Міністерство освіти і науки України
Донбаська державна машинобудівна академія

КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ І ПРОГРАМУВАННЯ

(Частина 1. Розробка програм на мові СІ)

Методичні вказівки до виконання лабораторних робіт для студентів спеціальності 123

Краматорськ 2019

Міністерство освіти і науки України
Донбаська державна машинобудівна академія

КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ І ПРОГРАМУВАННЯ
(Частина 1. Розробка програм мовою СІ)

Методичні вказівки до виконання лабораторних робіт
для студентів спеціальності
123

Затверджено
на засіданні кафедри
"Автоматизація виробничих процесів"
Протокол № від

Краматорськ 2019

УДК 681.5:681.3

Методичні вказівки для лабораторних робіт для студентів спеціальності 123/ Сост.: Е.В.Пищулина - Краматорськ: ДГМА, 2019. – 97 с.

Укладачі: Е.В.Пищулина, ст.препод.

Відп. за випуск Г. П. Клименко

ЗМІСТ

ЛАБОРАТОРНА РОБОТА 1

СТВОРЕННЯ КОНСОЛЬНОГО ЗАСТОСУВАННЯ

Мета роботи — Вивчити основні етапи створення консольних застосувань, навчитися оформляти графічну схему алгоритму, освоїти методику відладки і тестування програми за допомогою інтегрованого середовища пакету Visual C++ 6.0.

Розробка алгоритму рішення

Створення додатка будь-якого типу вимагає виконання певної послідовності дій, а саме:

1. На основі постановки завдання розробити алгоритм рішення.
2. Визначити необхідні модулі і функції.
3. Написати код додатка, використовуючи возсожности мови програмування.
4. Відлагодити програму.
5. Протестувати програму.

Розглянемо особливості кожного з цих етапів на прикладі наступного старовинного завдання.

Нехай дані 12 монет, відомо, що одна з них фальшива, причому невідомо, легше вона або важче за інших. Потрібно не більше зама за три зважування виявити фальшиву монету, використовуючи тільки важільні ваги (без гирь).

Слід зауважити, що на основі одного прикладу неможливо утямити усі особливості створення додатка. Проте основні, базові принципи будуть в лабораторній роботі представлені досить повно.

Передусім необхідно вирішити завдання. Розіб'ємо монети на три групи по чотири хвилини. Зважимо будь-кого дві груп. Можливі два випадки:

а) Весы урівноважені. Означає фальшива монета знаходиться в третій групі.

б) Весы неурівноважені. Тоді третя група не містить фальшивої монети.

Розглянемо випадок а). Розділимо третю групу на дві, одну групу покладемо на чашку вагів, на другу чашку покладемо будь-які дві справжні монети з восьми вже перевірених. Якщо ваги урівноважені, то монета знаходиться серед неперевірених, інакше вона знаходиться на вагах. Вибравши будь-яку з підозрюваних і одну хорошу, остаточно робимо вивід, яка монета фальшива.Рассмотрим случай а). Разделим третью группу на две, одну группу положим на чашку весов, на вторую чашку положим любые две настоящие монеты из восьми уже проверенных. Если весы уравновешены, то монета находится среди непроверенных, иначе она находится на весах. Выбрав любую из подозреваемых и одну хорошую, окончательно делаем вывод, какая монета фальшивая.

Випадок б) є важчим для аналізу. Одна група монет є важчою (ваги не урівноважені). Цю групу ділимо на дві по дві монети і до них додаємо по одній з легкої групи. Залишаться дві легкі монети. Якщо ваги урівноважені, то монета залишилася в групі неперевірених легких монет. Означає треба вибрати будь-яку з них і порівняти з хорошою.Случай б) является более трудным для анализа. Одна группа монет является более тяжелой (весы не уравновешены). Эту группу делим на две по две монеты и к ним добавляем по одной из легкой группы. Останутся две легкие монеты. Если весы уравновешены, то монета осталась в группе непроверенных легких монет. Значит нужно выбрать любую из них и сравнить с хорошей.

Якщо ваги неурівноважені, то фальшива монета може бути важкою і вона на нижній чаші вагів, або вона легка, тоді вона знаходиться на верхній чаші вагів. Виберемо дві важкі монети і зважимо. Якщо ваги будуть урівноважені, то фальшива монета легка і була раніше додана в іншу групу з двох важких і однією легкою монет, інакше вона важка і знаходиться на нижній чаші вагів.Если весы неуравновешены, то фальшивая монета может быть тяжелой и она на нижней чаше весов, либо

она легкая, тогда она находится на верхней чаше весов. Выберем две тяжелые монеты и взвесим. Если весы будут уравновешены, то фальшивая монета легкая и была ранее добавлена в другую группу из двух тяжелых и одной легкой монет, иначе она тяжелая и находится на нижней чаше весов.

Після того, як завдання вирішене, необхідно визначити, що повинна отримувати програма, що повинно бути на її виході і як буде осуществляться введення і вивід. В даному випадку нам необхідно ввести номер фальшивої монети і вказати, легка вона або важка.

Потім слід вирішити, як організувати введення. Є наступні варіанти:

- Організувати можливість введення даних безпосередньо в програмі.
- Отримати дані з командного рядка.
- Частину даних отримати з командного рядка, а частину запросити в програмі. Для однозначності приймемо, що «вага» вказується в командному рядку, а номер проситься у користувача.

Відразу необхідно передбачити перевірку введення даних, продумати питання відладки і тестування програми.

Функціональна декомпозиція

У найзагальнішому випадку програма - це послідовність операцій над структурами даних, які реалізують алгоритм рішення задачі. Для більшості завдань цей алгоритм досить об'ємний і складений. Для розділення і структуризації програм, для полегшення їх розуміння і супроводу використовуються прийоми структурного програмування. Ці прийоми концентруються на організації послідовності операцій в програмі. Один з основних прийомів полягає в розбитті програми на функції і модулі. Розділення програми на модулі дозволяє досягти коректності рішення задачі за рахунок більшої ясності, скоротити витрати по реалізації і супроводу, а також добитися повторного використання вже відлагодженого коду.В самом общем случае программа - это последовательность операций над структурами данных, которые реализуют алгоритм решения задачи. Для большинства задач этот алгоритм достаточно объемен и сложен. Для разделения и структурирования программ, для облегчения их понимания и сопровождения используются приемы структурного программирования. Эти приемы концентрируются на организации последовательности операций в программе. Один из основных приемов заключается в разбивке программы на функции и модули. Разделение программы на модули позволяет достичь корректности решения задачи за счет большей ясности, сократить расходы по реализации и сопровождению, а также добиться повторного использования уже отлаженного кода.

Розбиття програми на модулі, або функціональну декомпозицію, розпочинають із загального опису того, що робить програма. Потім виділяють окремі кроки алгоритму, які реалізують за допомогою функцій. Окремі функції, у свою чергу, також розділяються на дрібніші. У результаті полуячается ієрархія функцій, в якій функції більш високого рівня абстакции надають роботу функціям нижнього рівня. Цей метод відомий також як проектування зверху «вниз», оскільки він розпочинається з високорівневого опису програми, спесуаясь потім до усе більш дрібних деталей реалізації.Разбиение программы на модули, или функциональную декомпозицию, начинают с общего описания того, что делает программа. Затем выделяют отдельные шаги алгоритма, которые реализуют с помощью функций. Отдельные функции, в свою очередь, также разделяются на более мелкие. В итоге полуячается иерархия функций, в которой функции более высокого уровня абстакции предоставляют работу функциям нижнего уровня. Этот метод известен также как проектирование «сверху вниз», поскольку он начинается с высокоуровневого описания программы, спесуаясь затем до все более мелких деталей реализации.

Стосовно постановки лабораторній роботі, завдання полягає в тому, щоб отримати вхідні дані - «вага» і номер фальшивої монети, організувати три зважування і видати результат. Це можна було б представити в наступному виді.

```
// Глобальні дані
...
int main(iby argc, char *argv[]) {  // Стандартний заголовок функції mainint main(iby argc, char *argv[])  {  // Стандартный заголовок функции main
  // Локальні дані
...
  input();          // Отримати початкові даніinput();          // Получить исходные данные
  global_weighting();  // Організувати три «зважування»global_weighting();  // Организовать три «взвешивания»
  output();         // Вивести на екран номер фальшивої монетиoutput();         // Вывести на экран номер фальшивой монеты
}
```

Проте перша і остання функція в цьому завданні виконуються точно по одному разу, тому додаткові витрати на їх виклик в програмі не є обгрунтованими. В той же час функція зважування викликатиметься кілька разів, тому її можна виділити. Ця функція приймає на вході масиви монет на лівій і правій чашах вагів, а також число монет, що знаходяться на кожній з них. Однако первая и последняя функция в данной задаче выполняются точно по одному разу, поэтому дополнительные затраты на их вызов в программе не являются обоснованными. В то же время функция взвешивания будет вызываться несколько раз, поэтому ее можно выделить. Эта функция принимает на входе массивы монет на левой и правой чашах весов, а также число монет, находящихся на каждой из них.

Аналізуючи завдання, можна помітити, що функція повинна повернути одне з трьох значень - ваги урівноважені, важче ліва чаша і важче права чаша. Оскільки у разі урівноваження в чашах немає фальшивої монети, то це значення закодуємо нулем, - 1 означатиме, що важче ліва чаша, і 1, якщо права.

Лістинг 1Листинг 1

```
int weighting(int aLeft[], int *aRight, int iNumber) {
  int i;int i;
  int iWeightLeft  = 0, // Загальна вага монет на лівій чаші вагівint iWeightLeft  = 0, // Общий вес монет на левой чаше весов
    iWeightRight = 0; // Загальна вага монет на правій чаші вагівiWeightRight = 0; // Общий вес монет на правой чаше весов

  // Підраховуємо вагу монет окремо на лівій і правій чашах
  for(i = 0; i < iNumber; i++)
  {
   iWeightLeft  += aCoin[*aLeft + i - 1];
   iWeightRight += aCoin[aRight[i] - 1];
  }

  if(iWeightLeft == iWeightRight)
   return 0; // На вагах немає фальшивої монетиreturn 0; // На весах нет фальшивой монеты
  else  // Фальшива монета серед зважуванихelse  // Фальшивая монета среди взвешиваемых
   if(iWeightLeft < iWeightRight)
   return 1;return 1;
  else
   return - 1;return -1;
}
```

Тут представлені усі основні конструкції, що управляють : послідовність, цикл і вибір. Відмітимо, що в параметрах виклику функції представлені два способи передання масиву : за значенням (aLeft[]) і по посиланню, оскільки ім'я масиву є покажчиком (*aRight). Це зроблено в

учбових цілях, то ж відноситься і до циклу for.Здесь представлены все основные управляющие конструкции: последовательность, цикл и выбор. Отметим, что в параметрах вызова функции представлены два способа передачи массива: по значению (aLeft[]) и по ссылке, поскольку имя массива является указателем (*aRight). Это сделано в учебных целях, то же относится и к циклу for.

У лістингу 1 двічі робиться зважування (тобто порівняння ваги лівої і правої чаші вагів), тому можна виділити програмне введення номера фальшивої монети в окрему функцію (checkNumber), щоб уникнути дублювання коду, на неї ж покладається перевірка допустимості введеного значення (лістинг 2).

Лістинг 2Листинг 2

```
BOOL checkNumber(){
  printf("\nEnter false coin number: ");
  scanf("%d", &iBadCoin); // Пропускаємо операцію узяття адреси &scanf("%d", &iBadCoin); // Пропускаем операцию взятия адреса &
  if(iBadCoin < 1 || iBadCoin > 12) {if(iBadCoin < 1 || iBadCoin > 12)  {
   // Номер монети має бути від 1 до 12
   printf("Coin number must be from 1 to 12!\n");
   return FALSE;return FALSE;
  }
  return TRUE;return TRUE;
}
```

Таким чином, результатом проведеної функціональної декомпозиції є наступна ієрархія функцій (малюнок 1), три з яких - необхідно реалізувати, а дві бібліотечні.

Малюнок 1 - Результат виконання функціональної декомпозиції

Модульна композиція

Одиницею компіляції в мові З является файл або модуль. Звичайний спосіб доступу до зовнішніх визначень у файлі полягає в створенні окремого заголовного файлу, що містить зовнішні оголошення і підтримує необхідні лоя коректного використання визначення типів. Оголошення заголовчного файлу включаються у будь-який файл, що використовує зовнішні функції і об'єкти.

Розділення великої програми на модулі (файли) може полегшити процес створення, розуміння і супроводу програми. При цьому логічно пов'язані функції і структури даних можуть групуватися для кращої читабельності, а локальні модифікації обмежені модулем. Крім того, код, розділений на файли, легше переробляти для повторного використання в іншій програмі, або для створення бібліотек функцій.

У лабораторній роботі програма невелика і може бути реалізована у вигляді одного модуля, але для демонстрації різних аспектів програмування на мові З, розіб'ємо її на три модулі (малюнок 2).

Малюнок 2 - Розбиття програми на модулі

Після того, як проведена функціональна і модульна декомпозиції, тобто визначена структура програми, можна переходити до наступного етапу - написання коду.

Опис учбової програми

Тексти модулів приведені в лістингах 3-5.

Лістинг 3 - Основний модуль BadCoin.cЛистинг 3 - Основной модуль BadCoin.c

```
/* Програма визначення фальшивої монети.
```

Основний модуль BadCoin.c */Основной модуль BadCoin.c */

// У модулі є помилки

```
#include <conio.h>
#include <stdlib.h>
#include "BadCoin.h"

extern BOOL checkNumber();
extern int weighting(int aLeft[], int *aRight, int iNumber);

// Масив для зберігання монет
// Початково передбачається, що усі монети хороші
int aCoin[12] = {0, 0, 0, 0, int aCoin[12] = {0, 0, 0, 0,
                 0, 0, 0, 0,
                 0, 0, 0, 0};

// Зайве використання ключового слова externЛишнее использование ключевого слова extern
extern int iBadCoin;  // Номер фальшивої монетиextern int iBadCoin; // Номер фальшивой
монеты

int main(int argc, char *argv[])
{
  BOOL bWeight;  // TRUE - якщо фальшива монета важка iBOOL bWeight; // TRUE - если
фальшивая монета тяжелая и
                 // FALSE - якщо вона легкаFALSE - если она легкая
  int iResult;    // Результат зважуванняint iResult;  // Результат взвешивания
  // Масиви для зберігання монет на лівій
  int aLeft[4]  = {1, 2, 3, 4};int aLeft[4]  = {1, 2, 3, 4};
  //  і правою чашах вагів
  int aRight[4] = {5, 6, 7, 8};
  switch(argc)
  {
   case 1: printf("Program needs input data"!);
           do
            // Має бути натиснута або клавіша (h), якщо фальшива монетаДолжна быть нажата либо
клавиша (h), если фальшивая монета
            // важка, або клавіша (l), якщо вона легкатяжелая, либо клавиша (l), если она легкая
            printf("\nIf coin heavy press (h), otherwise - (l) : ");
           while((iResult = getchar()) != 'h' && iResult != 'l');

           bWeight = (iResult == 'h')? TRUE: FALSE;
           if(!checkNumber())
            return - 1;
           break;
     case 2: bWeight = (argv[1][0] == 'h');
           if(!checkNumber())
            return - 1;
           break;
     case 3: bWeight = (argv[1][0] == 'h');
           if((iBadCoin = atoi(argv[2])) < 1 || iBadCoin > 12)
            return - 1;
```

```
            break;
        default: printf("Too much data!\n");
            return - 2;return -2;

    // Пропустили закриваючу дужку
    }

    // Запам'ятовуємо фальшиву монету
    if(bWeight == TRUE)
      aCoin[iBadCoin - 1] = 1;
    else
      aCoin[iBadCoin - 1] = - 1;aCoin[iBadCoin - 1] = -1;
  // Введення даних закінчене. Приступаємо до визначення фальшивої монети.
    printf("\n\nWeighting 1");
    printf("\nCoins 1-4 on the left, coins 5-6 on the right");
    // Перше зважування
    iResult = weighting(aLeft, aRight, 4);
    if(!iResult)if(!iResult)
    { /* Фальшива монета знаходиться серед чотирьох решти */
      aLeft[0]  = 1, aLeft[1]  = 2;
      aRight[0] = 9, aRight[1] = 10;

      printf("\n\nWeighting 2");
      printf("\nCoins 1,2 on the left, coins 9,10 on the right");

      // Друге зважування
      iResult = weighting(aLeft, aRight, 2);
      if(!iResult)if(!iResult)
      { // Фальшива монета знаходиться серед двох що залишилися
        aLeft[0]  = 1;  // Можна опустити, оскільки значення не змінилосяaLeft[0]  = 1; // Можно
опустить, т.к. значение не изменилось
        aRight[0] = 11;

        printf("\n\nWeighting 3");
        printf("\nCoins 1 on the left, coins 11 on the right");

        // Третє зважування
        iResult = weighting(aLeft, aRight, 1);
        if(!iResult)  // Фальшивою є монета, що залишилася.if(!iResult) // Фальшивой является
оставшаяся монета.
          iResult = 12;iResult = 12;
        else              // Фальшива монета на вагах.else        // Фальшивая монета на весах.
          iResult = 11;iResult = 11;
      }
      else
      { /* Фальшива монета серед двох на правій чаші вагів. */
        aLeft[0]  = 1;  // Можна опустити, оскільки значення не змінилосяaLeft[0]  = 1; // Можно
опустить, т.к. значение не изменилось
        aRight[0] = 9;
        printf("\n\nWeighting 3");
        printf("\nCoins 1 on the left, coins 9 on the right");
```

```
        // Третє зважування
        iResult = weighting(aLeft, aRight, 1);
        if(!iResult)  // Фальшивою є монета, що залишилася.if(!iResult) // Фальшивой является
оставшаяся монета.
          iResult = 10;iResult = 10;
        else              // Фальшива монета на вагах.else        // Фальшивая монета на весах.
          iResult = 9;iResult = 9;
      }
    }
    else
    { /* Чотири монети, що залишилися, хороші.
      Розбиваємо вісім монет на три групи:
      перша і друга - дві важкі і одна легка;
      третя - дві легені */
      if(iResult == - 1)  // Важкі монети були на лівій чаші вагівif(iResult == -1) // Тяжелые монеты
были на левой чаше весов
      {
        aLeft[0]  = 1, aLeft[1]  = 2, aLeft[2]  = 5;
        aRight[0] = 3, aRight[1] = 4, aRight[2] = 6;

        printf("\n\nWeighting 2");
        printf("\nCoins 1,2,5 on the left, coins 3,4,6 on the right");

        // Друге зважування
        iResult = weighting(aLeft, aRight, 3);
        if(!iResult)if(!iResult)
        { // Фальшива монета 7 або 8
          aLeft[0]  = 1;
          aRight[0] = 7;

          printf("\n\nWeighting 3");
          printf("\nCoins 1 on the left, coins 7 on the right");

          // Третє зважування
          iResult = weighting(aLeft, aRight, 1);
          if(!iResult)  // Фальшивою є монета, що залишилася.if(!iResult) // Фальшивой является
оставшаяся монета.
            iResult = 8;iResult = 8;
          else              // Фальшива монета на вагах.else        // Фальшивая монета на весах.
            iResult = 7;iResult = 7;
        }
        else
          if(iResult == - 1)if(iResult == -1)
          { // Фальшива монета серед двох важких на нижній  Фальшивая монета среди двух тяжелых
на нижней
            // чи легка на верхній чаші вагів.
            aLeft[0]  = 1;
            aRight[0] = 2;

            printf("\n\nWeighting 3");
            printf("\nCoins 1 on the left, coins 2 on the right");
```

```
            // Третє зважування
            iResult = weighting(aLeft, aRight, 1);
            if(!iResult)  // Фальшивою є монета, що залишилася.if(!iResult) // Фальшивой является
оставшаяся монета.
              iResult = 5;
            else if(iResult == - 1)  // Фальшива монета на вагах.else if(iResult == -1) // Фальшивая монета
на весах.
              iResult = 1;
            else
              iResult = 2;
          }
          else
          {
           aLeft[0]  = 3;
           aRight[0] = 4;
           printf("\n\nWeighting 3");
           printf("\nCoins 3 on the left, coins 4 on the right");

           // Третє зважування
           iResult = weighting(aLeft, aRight, 1);
           if(!iResult)  // Фальшивою є монета, що залишилася.if(!iResult) // Фальшивой является
оставшаяся монета.
             iResult = 6;
           else if(iResult == - 1)  // Фальшива монета на вагах.else if(iResult == -1) // Фальшивая монета
на весах.
             iResult = 3;
           else
             iResult = 4;iResult = 4;
          }
        }
        else  // Важкі монети на правій чашіelse // Тяжелые монеты на правой чаше
        {
         aLeft[0]  = 5, aLeft[1]  = 6, aLeft[2]  = 1;
         aRight[0] = 7, aRight[1] = 8, aRight[2] = 2;

         printf("\n\nWeighting 2");
         printf("\nCoins 1,5,6 on the left, coins 2,7,8 on the right");

         // Друге зважування
         iResult = weighting(aLeft, aRight, 3);
         if(!iResult)if(!iResult)
         { // Фальшива монета 3 або 4
           aLeft[0]  = 1;
           aRight[0] = 3;
           printf("\n\nWeighting 3");
           printf("\nCoins 1 on the left, coins 3 on the right");

           // Третє зважування
           iResult = weighting(aLeft, aRight, 1);
           if(!iResult)  // Фальшивою є монета, що залишилася.if(!iResult) // Фальшивой является
оставшаяся монета.
             iResult = 4;iResult = 4;
```

```
        else                // Фальшива монета на вагах.else        // Фальшивая монета на весах.
          iResult = 3;iResult = 3;
        }
        else
         if(iResult == - 1)if(iResult == -1)
         { // Фальшива монета серед двох важких на нижній  Фальшивая монета среди двух тяжелых на нижней
          // чи легка на верхній чаші вагів.
          aLeft[0]  = 5;
          aRight[0] = 6;
          printf("\n\nWeighting 3");
          printf("\nCoins 5 on the left, coins 6 on the right");

          // Третє зважування
          iResult = weighting(aLeft, aRight, 1);
          if(!iResult)  // Фальшивою є монета, що залишилася.if(!iResult) // Фальшивой является оставшаяся монета.
            iResult = 2;
          else if(iResult == - 1)  // Фальшива монета на вагах.else if(iResult == -1) // Фальшивая монета на весах.
            iResult = 5;
          else
            iResult = 6;
         }
         else
         {
          aLeft[0]  = 7;
          aRight[0] = 8;

          printf("\n\nWeighting 3");
          printf("\nCoins 7 on the left, coins 8 on the right");

          // Третє зважування
          iResult = weighting(aLeft, aRight, 1);
          if(!iResult)  // Фальшивою є монета, що залишилася.if(!iResult) // Фальшивой является оставшаяся монета.
            iResult = 1;
          else if(iResult == - 1)  // Фальшива монета на вагах.else if(iResult == -1) // Фальшивая монета на весах.
            iResult = 7;
          else
            iResult = 8;iResult = 8;
         }
       }
      }
      // Виводимо результат на екран
      printf("\n\nBad coin is %d\n\n", iResult);
      getchar();
      return 0;
    }
```

Лістинг 4 - Модуль Auxiliary.cЛистинг 4 - Модуль Auxiliary.c

/* Модуль Auxiliary.c, що містить допоміжні функції */Модуль Auxiliary.c, содержащий вспомогательные функции */

// У модулі є помилки

#include "BadCoin.h"include "BadCoin.h"

// Робимо доступними глобальні змінні, визначені в модулі BadCoin.cДелаем доступными глобальные переменные, определенные в модуле BadCoin.c

```
extern int aCoin[12];
int iBadCoin;

BOOL checkNumber()
{
 printf("\nEnter false coin number: ");
 scanf("%d", &iBadCoin);  // Пропускаємо операцію узяття адреси &scanf("%d", &iBadCoin); // Пропускаем операцию взятия адреса &
 if(iBadCoin < 1 || iBadCoin > 12)if(iBadCoin < 1 || iBadCoin > 12)
 {
  // Номер монети має бути від 1 до 12
  printf("Coin number must be from 1 to 12!\n");
  return FALSE;
 }
 return TRUE;
}

int weighting(int aLeft[], int *aRight, int iNumber)
{ /* Повертає:
    0, якщо ваги урівноважені;
   -1, якщо важче ліва чаша і
    1, якщо права
 */
 int i;int i;
 int iWeightLeft  = 0,  // Загальна вага монет на лівій чаші вагівint iWeightLeft  = 0, // Общий вес монет на левой чаше весов
     iWeightRight = 0;  // Загальна вага монет на правій чаші вагівiWeightRight = 0; // Общий вес монет на правой чаше весов

 // Підраховуємо вагу монет окремо на лівій і правій чашах
 for(i = 0; i < iNumber; i++)
 {
  iWeightLeft  += aCoin[*aLeft + i - 1];
  iWeightRight += aCoin[aRight[i] - 1];
 }

 if(iWeightLeft == iWeightRight)
  return 0;  // На вагах немає фальшивої монетиreturn 0; // На весах нет фальшивой монеты
 else if(iWeightLeft < iWeightRight)
  return 1;                                  // Важче права чашаreturn 1;                  // Тяжелее правая чаша
 else
```

return - 1; // Важче ліва чаша**return -1;** // Тяжелее левая чаша
}

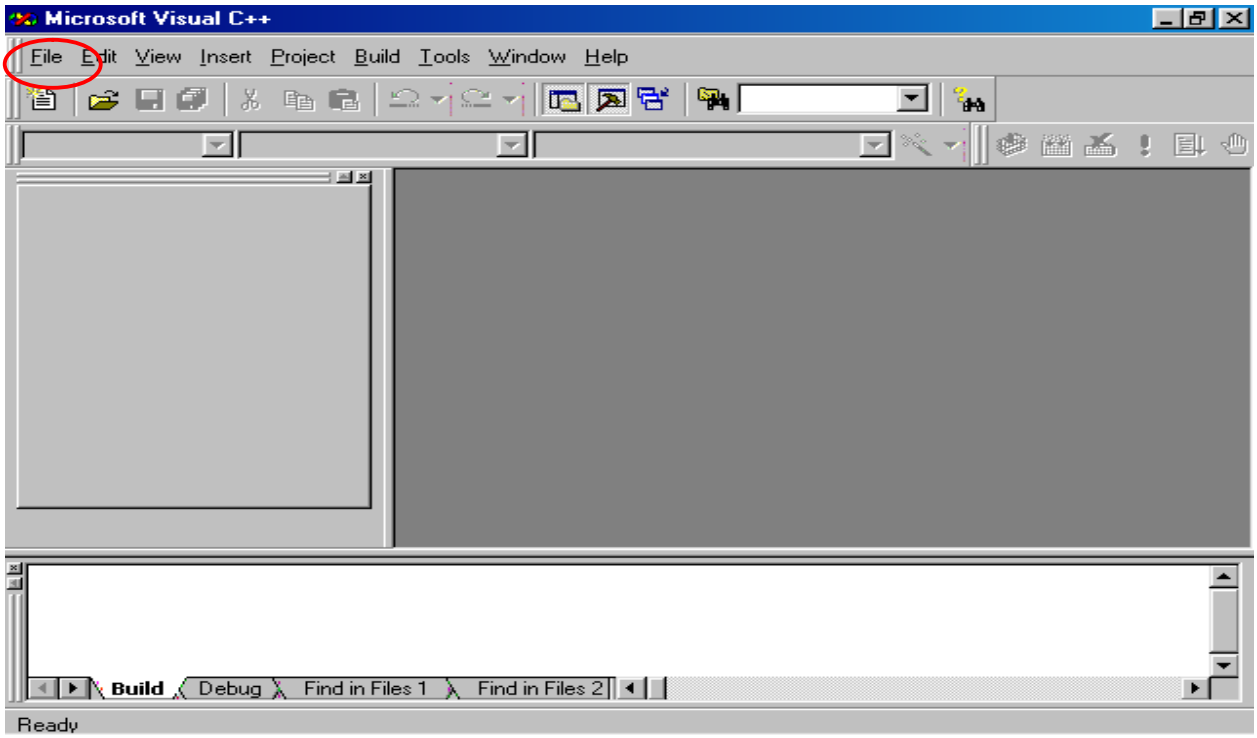
Лістинг 5 - Основний заголовний модуль BadCoin.h

```
/* Програма визначення фальшивої монети.
Основний заголовний модуль BadCoin.h  */Основной заголовочный модуль BadCoin.h  */

// У модулі є помилки

#ifndef _BAD_COINifndef _BAD_COIN
#define _BAD_COINdefine _BAD_COIN

// Ці параметри використовуються в обох модулях
#define TRUE 1
#define FALSE 0
// Пропускаємо крапку з комоюПропускаем точку с запятой
typedef char BOOL;
#include <stdio.h>
#endif  // _BAD_COINendif // _BAD_COIN
```

Створення проекту

Після того, як складений код програми, створюється нове застосування з використанням інструментальних засобів інтегрованого середовища розробки - IDE.

1. Запустіть додаток **Visual C++ 6**, клацнувши по значку на робочому столі або вибравши команду з меню Старт. У вікні **IDE** (малюнок 3), що з'явилося, виконаєте команду **File ->New...,** внаслідок чого на екрані з'явиться діалогове вікно New (малюнок 4).Запустіть



додаток **Visual C++ 6**, клацнувши по значку на робочому столі або вибравши команду з меню Старт. У вікні **IDE** (малюнок 3), що з'явилося, виконаєте команду **File ->New...,** внаслідок чого на екрані з'явиться діалогове вікно New (малюнок 4).Запустіть додаток **Visual C++ 6**, клацнувши по значку на робочому столі або вибравши команду з меню Старт. У вікні **IDE**

(малюнок 3), що з'явилося, виконаєте команду **File ->New...,** внаслідок чого на екрані з'явиться діалогове вікно New (малюнок 4).Запустите приложение **Visual C++ 6**, щелкнув по значку на рабочем столе или выбрав команду из меню Старт. В появившемся окне **IDE** (рисунок 3) выполните команду **File->New...,** в результате чего на экране появится диалоговое окно New (рисунок 4).

Малюнок 3 - Початковий екран середовища розробки **Visual C**++Рисунок 3 – Начальный экран среды разработки **Visual C**++

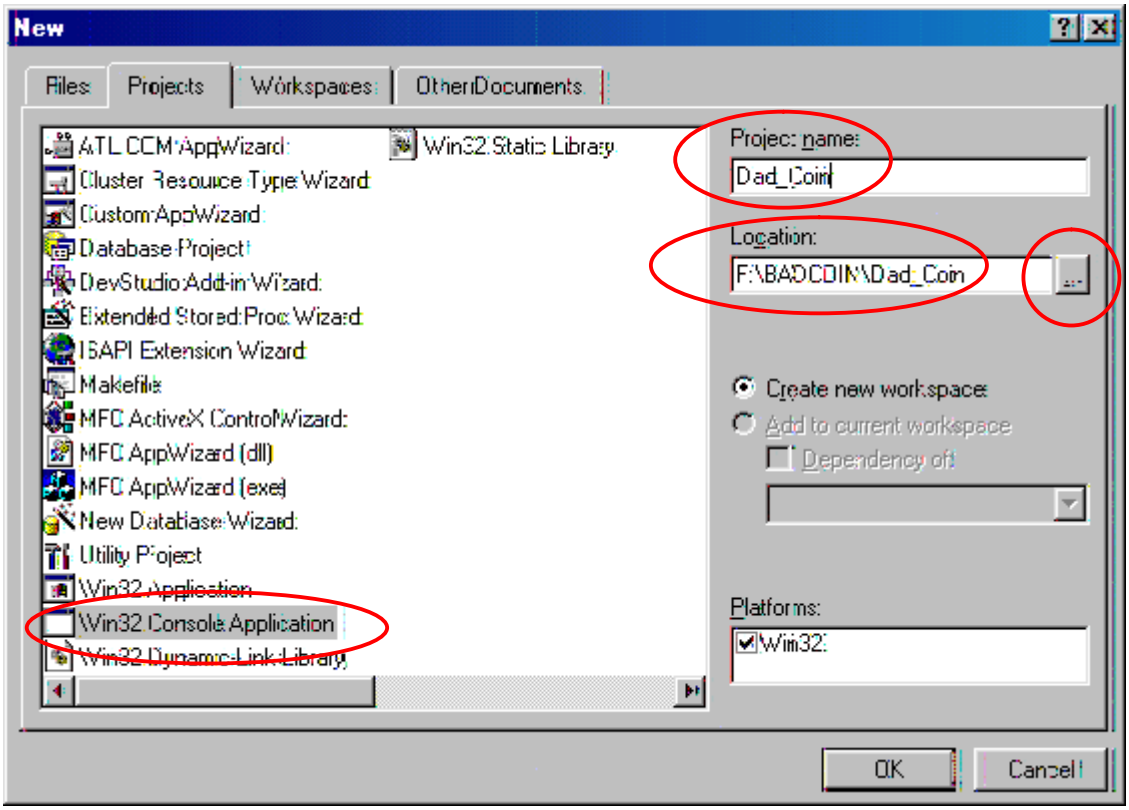


Рисунок 4 – Окно **New** для выбора типа создаваемого проекта

2. Оскільки створюється консольне застосування - аналог програми для MS - **DOS**, то слід вибрати в лівому списку пункт **Win32 Console Application**.Поскольку создается консольное приложение – аналог программы для MS-**DOS**, то следует выбрать в левом списке пункт **Win32 Console Application**.

У полі **ProjectName** введіть назву проекту - **BadCoin**, а в полі **Location** необхідно ввести інформацію про розташування робочої теки. Найпростіше зробити за допомогою кнопки перегляду, розташованої зліва від цього поля. Після клацання на цій кнопці у вікні вибору теки (малюнок 5) знайдіть свою робочу теку. При роботі в локальній мережі кафедри вона розташовується на мережевому диску **Z**. Потім клацніть на кнопці **OK**.В поле **ProjectName** введите название проекта – **BadCoin**, а в поле **Location** необходимо ввести информацию о расположении рабочей папки. Проще всего сделать при помощи кнопки просмотра, расположенной слева от этого поля. После щелчка на этой кнопке в окне выбора папки (рисунок 5) найдите свою рабочую папку. При работе в локальной сети кафедры она располагается на сетевом диске **Z**. Затем щелкните на кнопке **OK**.

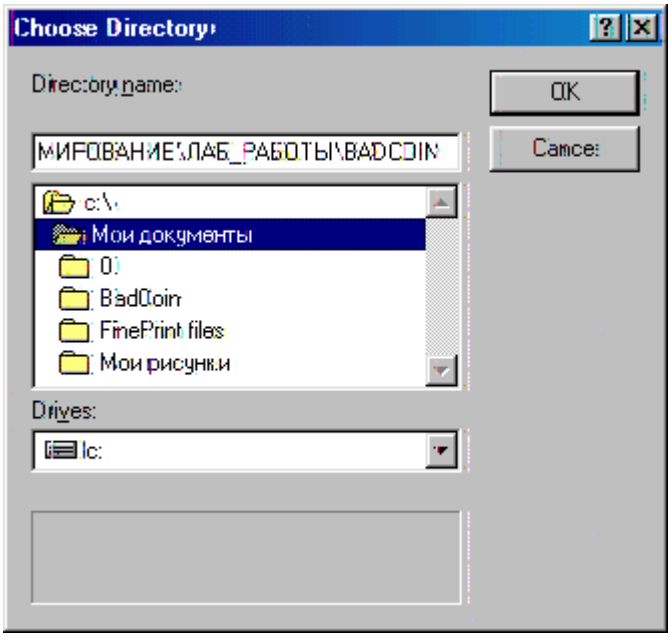


Рисунок 5 – Окно для выбора папки размещения файлов проекта

3. На экране появится окно мастера создания консольного приложения (рисунок 6).

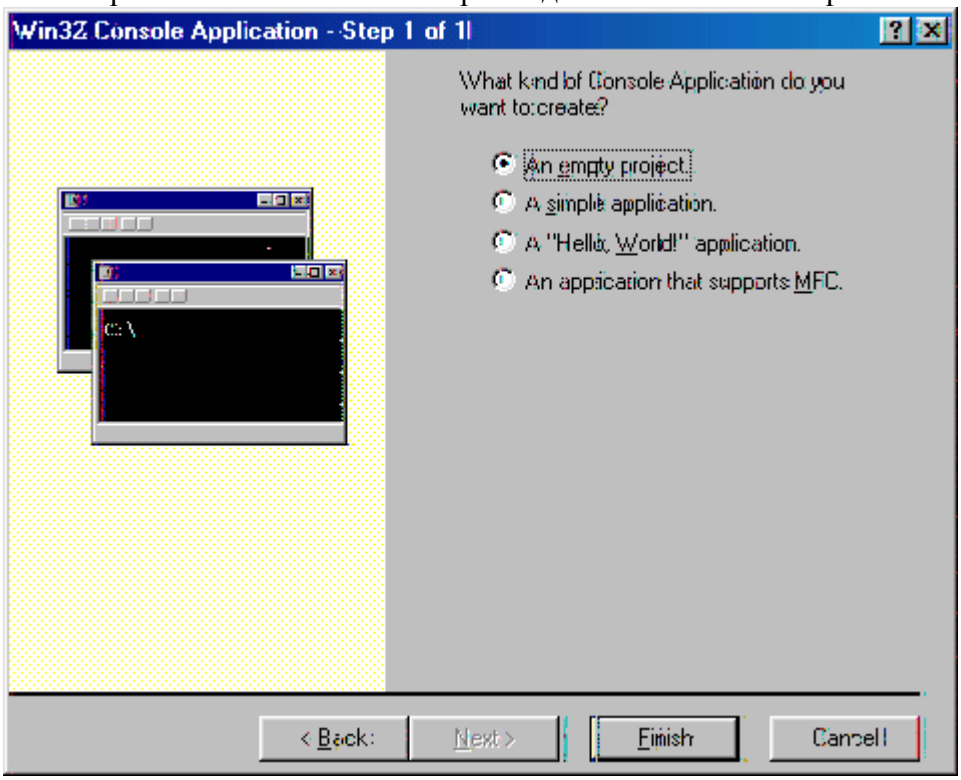


Рисунок 6 – Диалоговое окна мастера создания консольного приложения

При виборі перемикача **As Empty project** (порожній проект) не створюється ніяких додаткових файлів- тільки файл проекту. Для того, щоб наповнити створений проект, необхідно в нього додати файли, содержашие текст програми.При выборе переключателя **As Empty project** (пустой проект) не создается никаких дополнительных файлов- только файл проекта. Для того чтобы наполнить созданный проект, необходимо в него добавить файлы, содержашие текст программы.

4. Це можна зробити двома способами:

- додати в проект вже існуючий файл, створений раніше в текстовому редакторові;
- створити новий порожній файл і вставити його в проект.

Для додавання в проект існуючого файлу необхідно вибрати команду **Project ->Add to Project** (додати в проект) ->**Files...** В результаті цих дій на екран буде виведено діалогове вікно **Insert Files into Project**, показане на малюнку 7.Для добавления в проект существующего файла необходимо выбрать команду **Project->Add to Project** (добавить в проект)->**Files...** В результате этих действий на экран будет выведено диалоговое окно **Insert Files into Project**, показанное на рисунке 7.

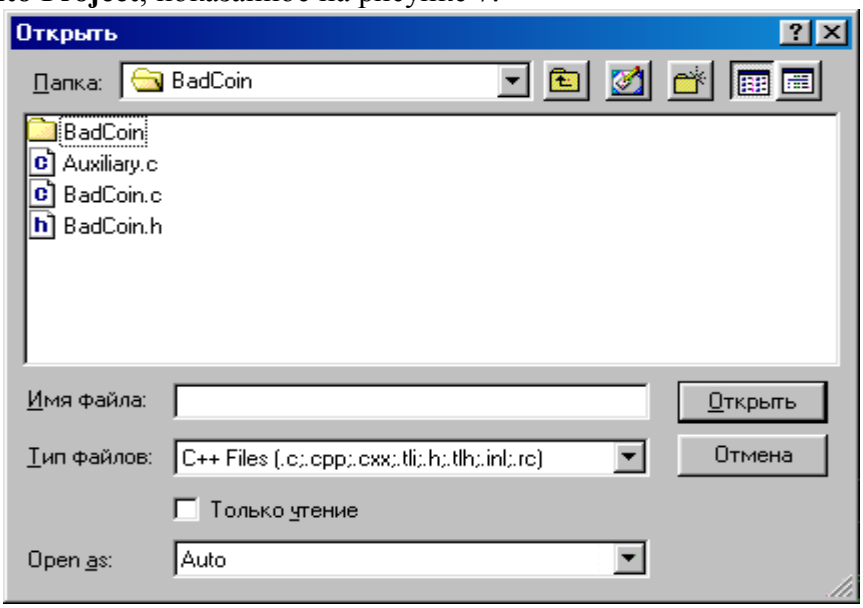


Рисунок 7 – Диалоговое окно добавления файлов в проект

Потім слід вибрати файли, двічі клацнувши по імені файлу, або виділивши потрібні файли і натиснувши кнопку **ОКИ**.Затем следует выбрать файлы, дважды щелкнув по имени файла, либо выделив нужные файлы и нажав кнопку **ОК**.

5. Создание нового файла делается кнопкой **New** на панели инструментов, или командой меню **File ->New**..., комбинацией ли клавиш **<Ctrl>+<N>**. В результате появится диалоговое окно **New**, открытое на вкладке **Files**, где представлены все типы файлов, которые можно создавать (рисунок 8).Создание нового файла производится кнопкой **New** на панели инструментов, или командой меню **File ->New**..., или комбинацией клавиш **<Ctrl>+<N>**. В результате появится диалоговое окно **New**, открытое на вкладке **Files**, где представлены все типы файлов, которые можно создавать (рисунок 8).Створення нового файлу робиться кнопкою **New** на панелі інструментів, або командою меню **File ->New**..., чи комбінацією клавіш **<Ctrl>+<N>**. В результаті з'явиться діалогове вікно **New**, відкрите на вкладці **Files**, де представлені усі типи файлів, які можна створювати (малюнок 8).Создание нового файла производится кнопкой **New** на панели инструментов, или командой меню **File->New**..., или комбинацией клавиш **<Ctrl>+<N>**. В результате появится диалоговое окно **New**, открытое на вкладке **Files**, где представлены все типы файлов, которые можно создавать (рисунок 8).

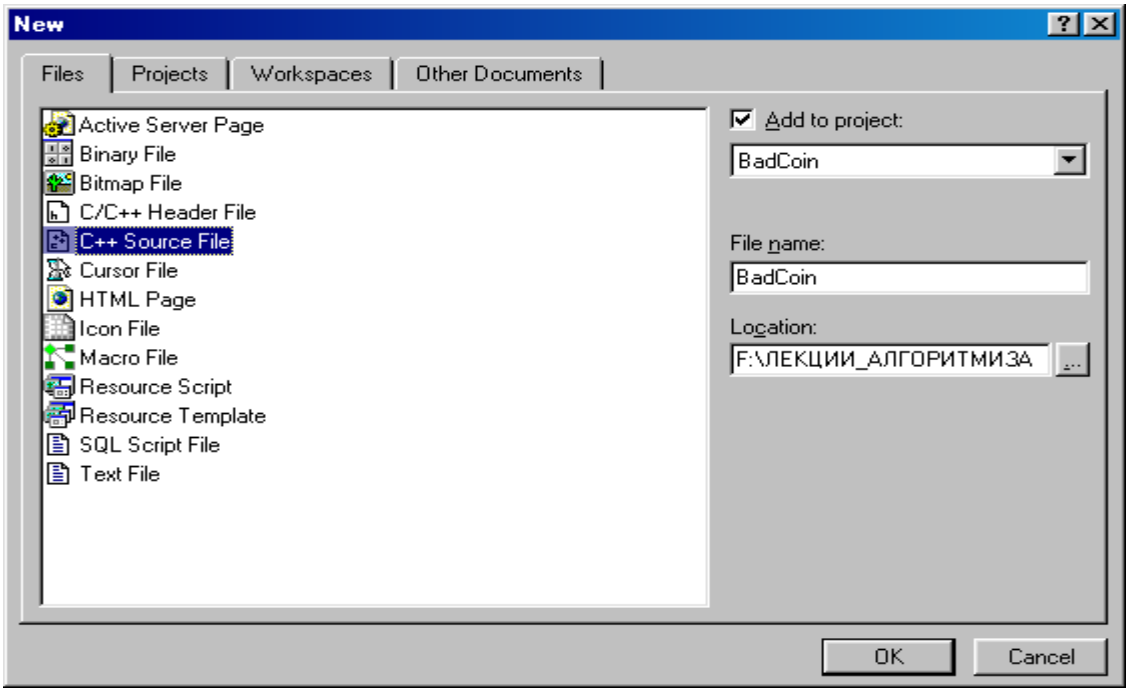
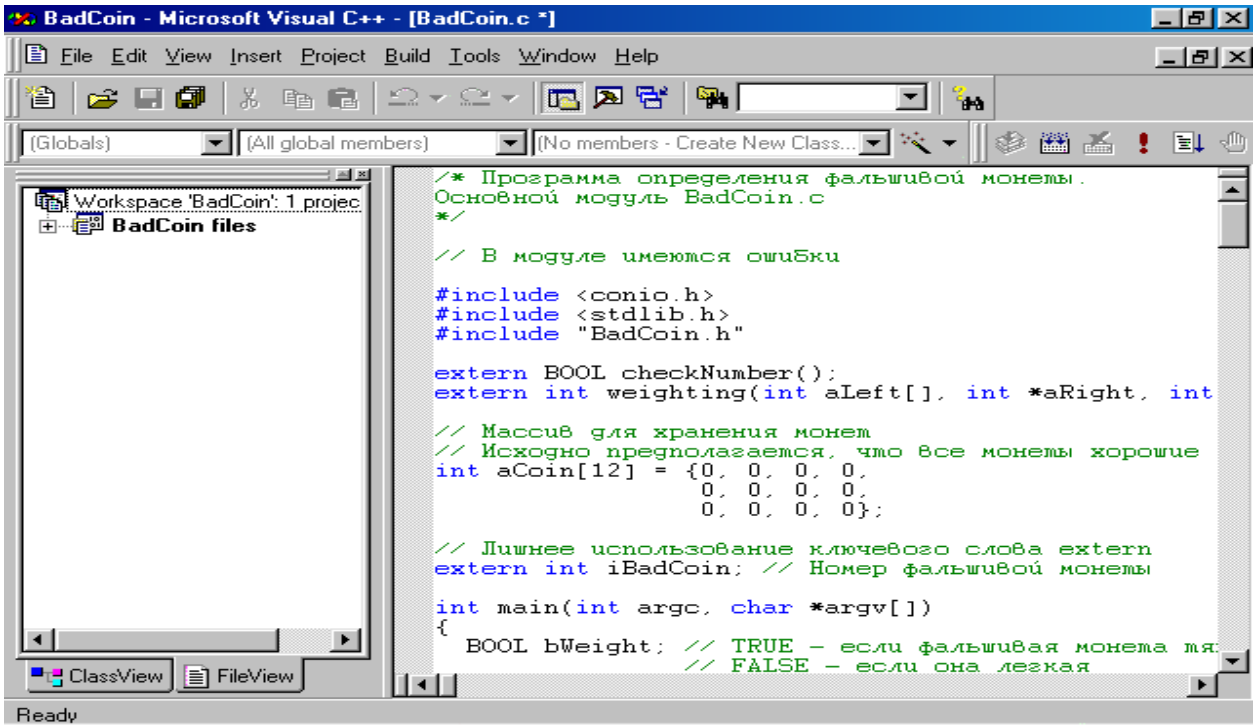


Рисунок 8 – Окно для добавления файла в проект

Прапорець **Add to Project** (додати в проект) має бути встановлений, щоб створюваний файл автоматично був доданий в проект. Виберіть тип створюваного файлу - **C/C++ Header File** або **C++ Source File,** а в полі **File name** - ім'я файлу, потім натисніть кнопку **OK**.Флажок **Add to Project** (добавить в проект) должен быть установлен, чтобы создаваемый файл автоматически был добавлен в проект. Выберите тип создаваемого файла – **C/C++ Header File** или **C++ Source File,** а в поле **File name** - имя файла, затем нажмите кнопку **OK**.



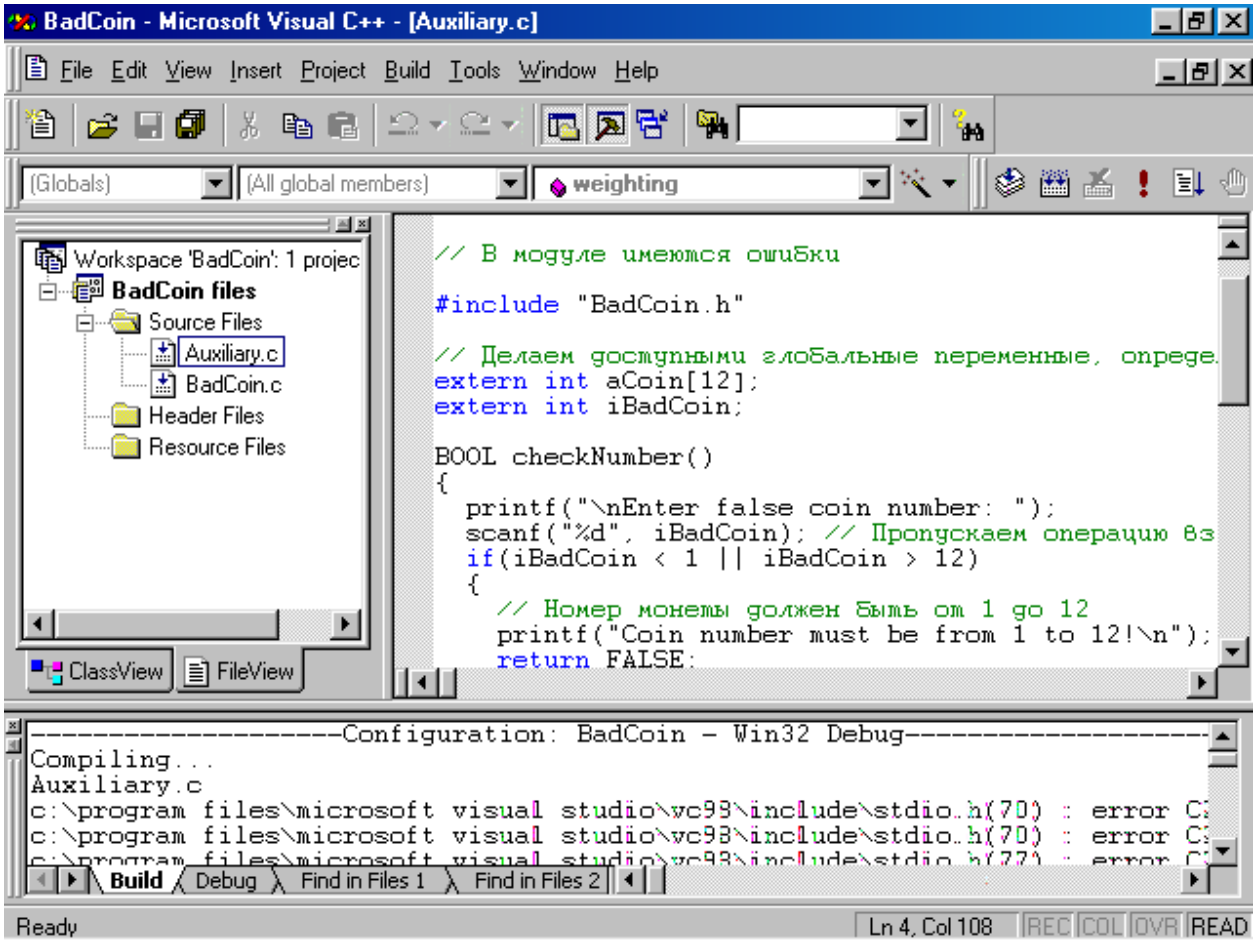
6. Відкриється вікно з порожнім полем для редагування тексту (малюнок 9).

Малюнок 9 - Вікно редагування **Visual C++** з видимим текстом.Рисунок 9 – Окно редактирования **Visual C++** с видимым текстом.

6. Після того, як в проекті опиняться усі потрібні файли, починається етап відладки програм. На цьому етапі треба усунути усі синтаксичні і велику частину логічних помилок. Для цього необхідно скомпілювати кожен створений модуль. Це можна зробити двома способами:

- Скомпілювати кожен модуль, що має розширення .з, використовуючи команду **Build ->Compile ім'я файлу**. Цю команду зручно використати у разі великих проектів, щоб сконцентруватися на конкретному модулі. При цьому слід мати на увазі, що міжмодульні зв'язки не перевіряються. Для запуску компонувальника створення виконуваного файлу необхідно виконати одну з команд, приведених нижче.Скомпилировать каждый модуль, имеющий расширение .c, используя команду **Build->Compile имя файла**. Эту команду удобно использовать в случае больших проектов, чтобы сконцентрироваться на конкретном модуле. При этом следует иметь в виду, что межмодульные связи не проверяются. Для запуска компоновщика создания исполняемого файла необходимо выполнить одну из команд, приведенных ниже.
- Скомпілювати і скомпоновать усі модулі проекту (для опису цих дій використовуються терміни «зборка» або «побудова»), скориставшись для цього командами **Build ->Build ім'я файлу** або **Build ->Rebuild All**. Єдиною відмінністю цих команд є те, що команда **Rebuild All** не перевіряє дати створення файлв проекту.Скомпилировать и скомпоновать все модули проекта (для описания этих действий используются термины «сборка» или «построение»), воспользовавшись для этого командами **Build->Build имя файла** или **Build->Rebuild All**. Единственным отличием этих команд является то, что команда **Rebuild All** не проверяет даты создания файлв проекта.

7. Якщо програма містить синтаксичні помилки, при виконанні компіляції смороду автоматичний відображаються у вікні, за умовчанням розташованому в нижній частині вікна інтегрованого середовища розробки (**IDE**) (малюнок 10).Якщо програма містить синтаксичні



помилки, при виконанні компіляції вони автоматично відображаються у вікні, за умовчанням розташованому в нижній частині вікна інтегрованого середовища розробки (**IDE**) (малюнок

10).Якщо програма містить синтаксичні помилки, при виконанні компіляції вони автоматично відображаються у вікні, за умовчанням розташованому в нижній частині вікна інтегрованого середовища розробки (**IDE**) (малюнок 10).Если программа содержит синтаксические ошибки, при выполнении компиляции они автоматически отображаются в окне, по умолчанию расположенном в нижней части окна интегрированной среды разработки (**IDE**) (рисунок 10).

Малюнок 10 - Екран IDE Visual C++ з перерахуванням помилок

Помилки в початкових файлах проекту були внесені свідомо, тому, якщо текст був набраний вірно, то в списку помилок можна помітити, що вони були виявлені в стандартному заголовному файлі **stdio.h**, де їх не повинно було бути. Насправді помилка зовсім у іншому місці. Це пояснюється тим, що компілятор не завжди правильно локалізує помилку, особливо, якщо це пов'язано з пропущеною дужкою або крапкою з комою.Ошибки в исходных файлах проекта были внесены сознательно, поэтому, если текст был набран верно, то в списке ошибок можно заметить, что они были обнаружены в стандартном заголовочном файле **stdio.h**, где их не должно было быть. На самом деле ошибка совсем в другом месте. Это объясняется тем, что компилятор не всегда правильно локализует ошибку, особенно, если это связано с пропущенной скобкой или точкой с запятой.

8. Слід знайти перше входження файлу **stdio.h** в компільованому модулі і уважно пошукати пропущений символ над цим рядком. У нашому випадку помилки припустилася в рядкуСледует найти первое вхождение файла **stdio.h** в компилируемом модуле и внимательно поискать пропущенный символ над этой строкой. В нашем случае ошибка допущена в строке

typedef char BOOLtypedef char BOOL

модуля **BadCoin.h,** що стоїть, безпосередньо перед рядкоммодуля **BadCoin.h,** стоящей, непосредственно перед строкой

#include <stdio.h>include <stdio.h>

Оператор **typedef** вимагає наявності у кінці оголошення нового типу крапки з комою; поставте її і наново скомпілюйте програму.Оператор **typedef** требует наличия в конце объявления нового типа точки с запятой; поставьте ее и заново скомпилируйте программу.

9. Цього разу виявлена єдина помилка "**fatal error C1004 : unexpected end of file found" (**несподіваний кінець файлу). Ця помилка також пов'язана з пропущеною дужкою або крапкою з комою. Двічі клацніть на рядку з повідомленням про помилку, і помилковий рядок буде знайдений (малюнок 11). Помилка знову не локалізована — шукайте пропущену закриваючу фігурну дужку. Вона може бути у будь-якому місці файлу. В даному випадку пропущена закриваюча дужка в операторові switch.На этот раз обнаружена единственная ошибка "**fatal error C1004: unexpected end of file found**" (неожиданный конец файла). Эта ошибка также связана с пропущенной скобкой или точкой с запятой. Дважды щелкните на строке с сообщением об ошибке, и ошибочная строка будет найдена (рисунок 11). Ошибка опять не локализована — ищите пропущенную закрывающую фигурную скобку. Она может быть в любом месте файла. В данном случае пропущена закрывающая скобка в операторе **switch**.

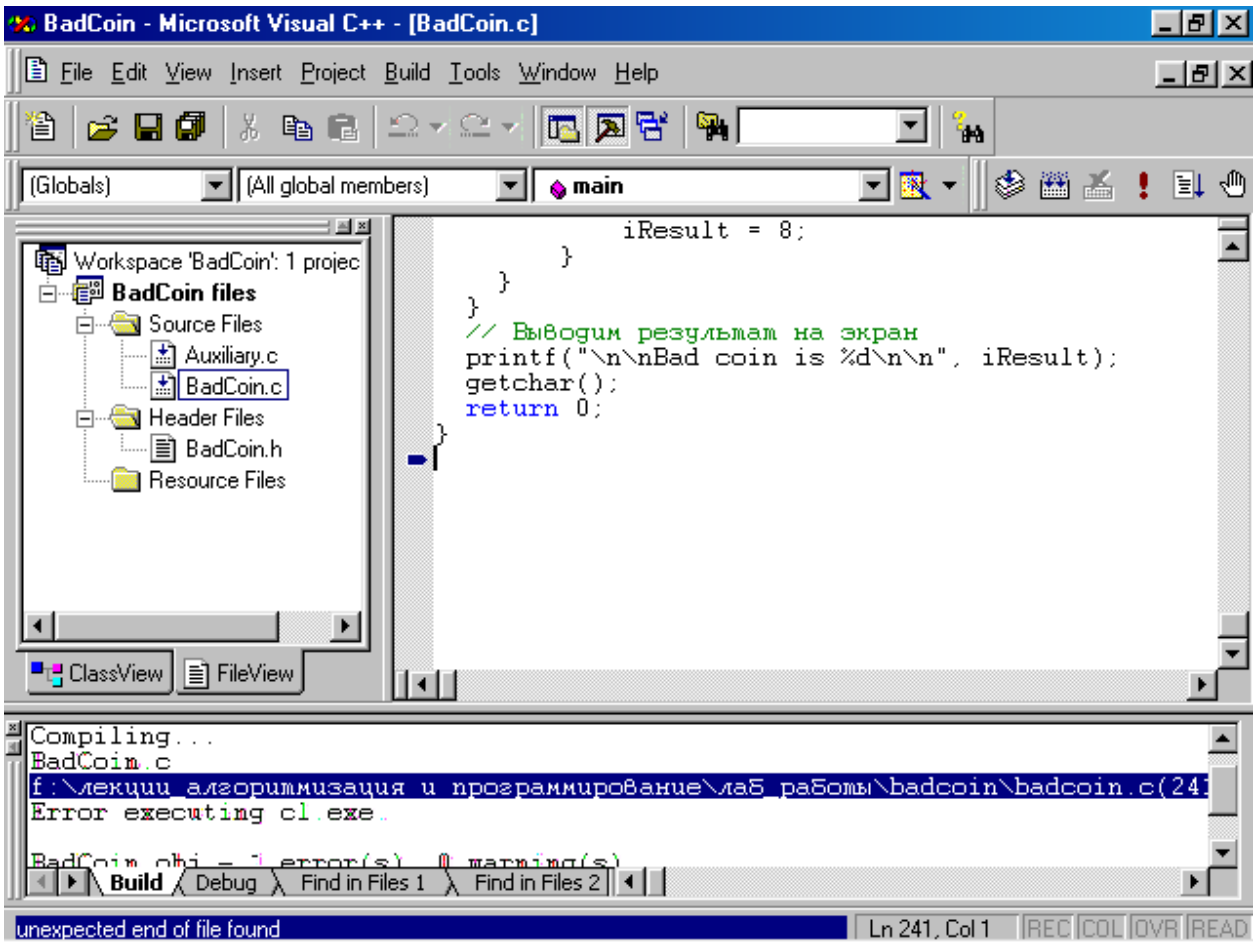


Рисунок 11 – Окно с нелокализованной ошибкой

10. Поставте пропущену закриваючу фігурну дужку і виконайте чергову зборку.

Синтаксичних помилок більше немає, проте з'явилося повідомлення про помилку на етапі компонування (малюнок 12). Такого роду помилки Visual **C**++ також не локалізує, тому шукати їх слід самостійно.

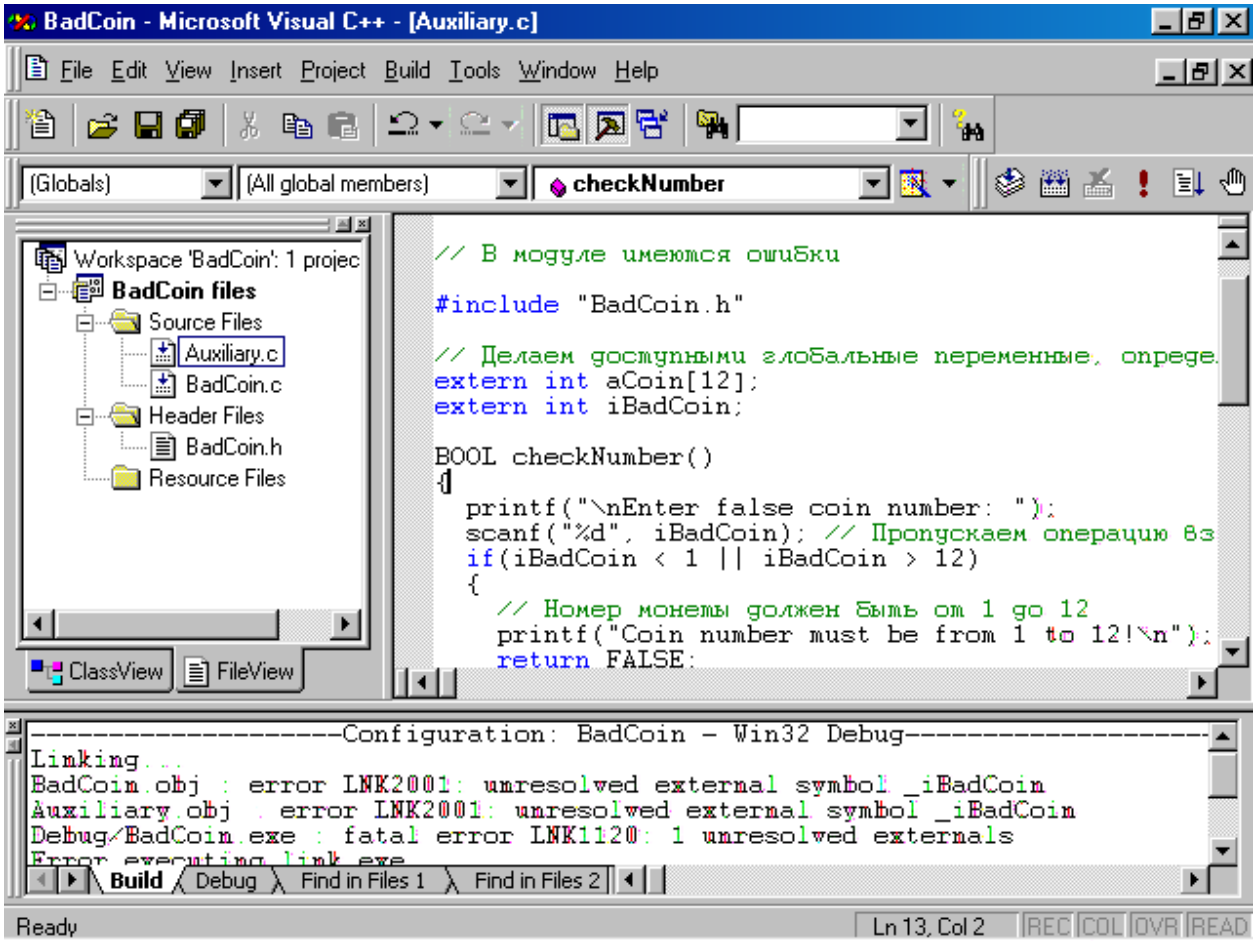


Рисунок 12 – Ошибки, выявленные компоновщиком

В даному випадку є підказка: одна і та ж помилка виявлена в двох файлах. Швидше за все, це оголошення змінної без її визначення. Дійсно, в обох файлах є рядок

extern int iBadCoin;extern int iBadCoin;

Видаліть слово extern у будь-якому з них і ще раз "зберіть" програму.Удалите слово extern в любом из них и еще раз "соберите" программу.

11. Помилок більше немає, запустите програму, виконавши команду **Build|Execute ім'я_файлу.** Надрукуйте дані (малюнок 13), натисніть клавішу <ENTER>.Ошибок больше нет, запустите программу, выполнив команду **Build|Execute имя_файла.** Напечатайте данные (рисунок 13), нажмите клавишу <ENTER>.

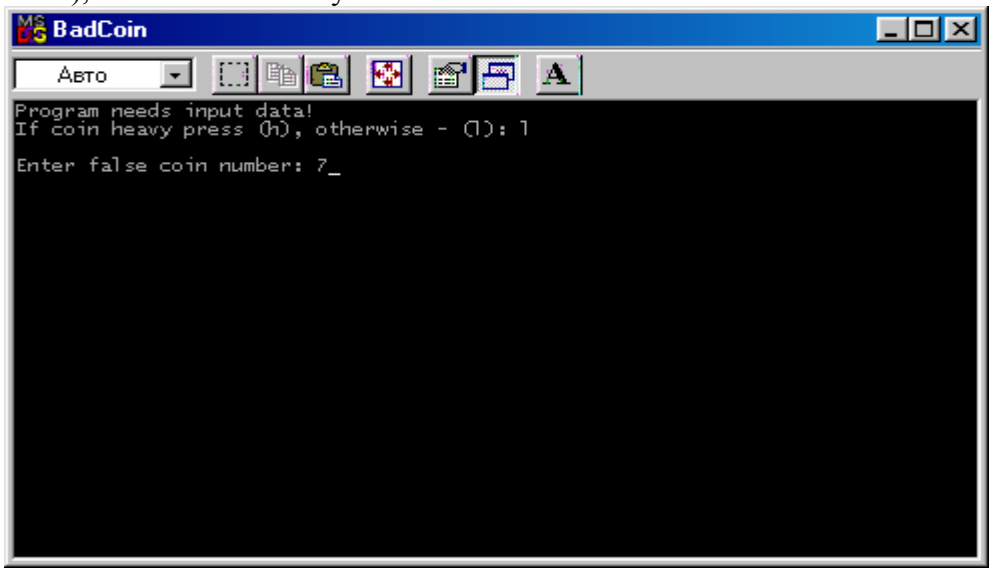


Рисунок 13 – Начало выполнения программы

12. Программа останавливается с выводом сообщения об ошибке (рисунок 14), хотя программа и не содержит синтаксических ошибок, но приложение не работает. Вызвано это так называемыми логическими ошибками, для обнаружения которых можно использовать разные методы (например, закомментировать фрагменты программы).

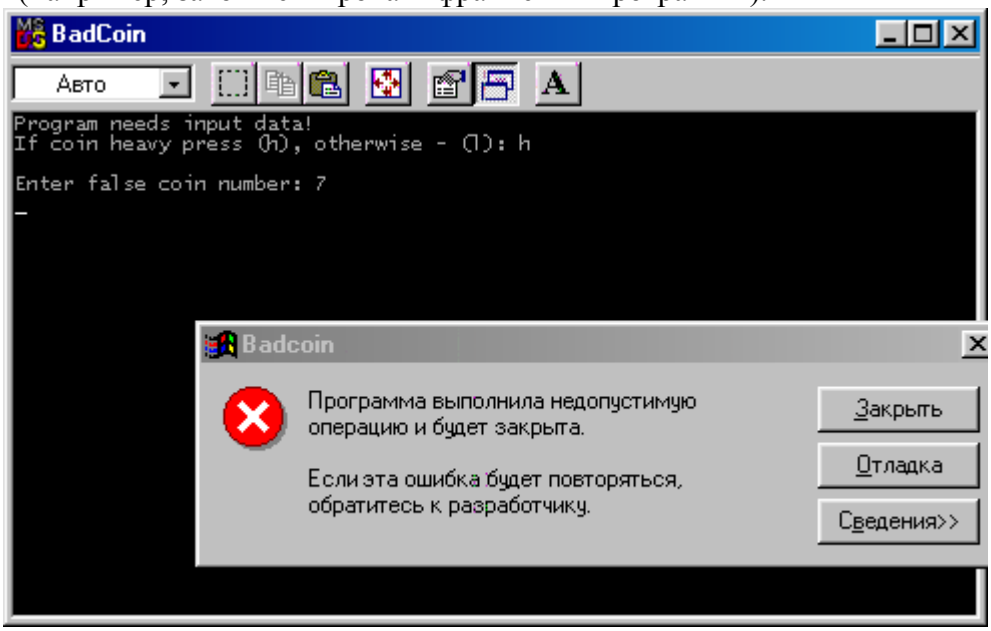


Рисунок 14 – Отладчик сообщает об ошибке

Проте краще всього скористатися наявним в інтегрованого середовища розробки (**IDE**) вбудованим відладчиком. Основні можливості з тих, що надаються їм, які рассмотриваются в лабораторній роботі, наступні:Однако лучше всего воспользоваться имеющимся в интегрированной среды разработки (**IDE**) встроенным отладчиком. Основные возможности из предоставляемых им, которые рассмотриваются в лабораторной работе, следующие:

- покрокове виконання програми;
- перегляд значень змінних у будь-якій точці програми.

Для покрокового виконання відладчик надає декілька команд. Для запуску виконуваного файлу в режимі відладки є команда **Go** (виконати). Проте, якщо просто виконати цю команду, робота програми не відрізнятиметься від запуску в звичайному режимі, просто при її завершенні на вкладці **Debug** в нижній частині вікна **IDE** з'явиться інформація про параметри завершення роботи програми. Щоб перейти в режим покрокового виконання, заздалегідь необхідно встановити так звані точки останову (**breakpoints**), які можна розглядати як стоп-сигнал для відладчика. Вони зазвичай встановлюються в місці, яке викликає сумніву в правильності виконання. При цьому передбачається, що усі оператори, передуючі цьому рядку, виконуються правильно.Для пошагового выполнения отладчик предоставляет несколько команд. Для запуска исполняемого файла в режиме отладки имеется команда **Go** (выполнить). Однако, если просто выполнить эту команду, работа программы не будет отличаться от запуска в обычном режиме, просто при ее завершении на вкладке **Debug** в нижней части окна **IDE** появится информация о параметрах завершения работы программы. Чтобы перейти в режим пошагового выполнения, предварительно необходимо установить так называемые точки останова (**breakpoints**), которые можно рассматривать как стоп-сигнал для отладчика. Они обычно устанавливаются в месте, которое вызывает сомнения в правильности выполнения. При этом предполагается, что все операторы, предшествующие этой строке, выполняются правильно.

13. У нашому випадку можна припустити, грунтуючись на місці появи помилки, що вона пов'язана з введенням номера фальшивої монети. Таке місце в програмі одне — у функції **checkNumber**(), а ще точніше — у використовуваній там функції **scanf**(). На цьому рядку і

встановите точку останову, Зробити це можна декількома способами. Найпростіший полягає в тому, що курсор встановлюється на рядок, на якому треба зупинити виконання програми і натискається функціональна клавіша <**F9**>.В нашем случае можно предположить, основываясь на месте появления ошибки, что она связана с вводом номера фальшивой монеты. Такое место в программе одно — в функции **checkNumber**(), а еще точнее — в используемой там функции **scanf**(). На этой строке и установите точку останова, Сделать это можно несколькими способами. Самый простой заключается в том, что курсор устанавливается на строку, на которой нужно остановить выполнение программы и нажимается функциональная клавиша <**F9**>.

При будь-якому способі рядок у вікні редагування з точкою останову буде відмічений червоним гуртком в крайній лівій позиції.При любом способе строка в окне редактирования с точкой останова будет отмечена красным кружком в крайней левой позиции.

14. Запустимо програму в режимі відладки, виконавши команду **Go,** або натиснувши функціональну клавішу <**F5**>. Усі оператори програми, передуючі точці останову, виконуватимуться в звичайному режимі і тільки в рядку 14 (у вас це може бути інший рядок, якщо ви вставили або, навпаки, прибрали деякі порожні рядки), виконання програми призупиниться (малюнок 15).Запустим программу в режиме отладки, выполнив команду **Go,** либо нажав функциональную клавишу <**F5**>. Все операторы программы, предшествующие точке останова, будут выполняться в обычном режиме и только в строке 14 (у вас это может быть другая строка, если вы вставили или, наоборот, убрали некоторые пустые строки), выполнение программы приостановится (рисунок 15).

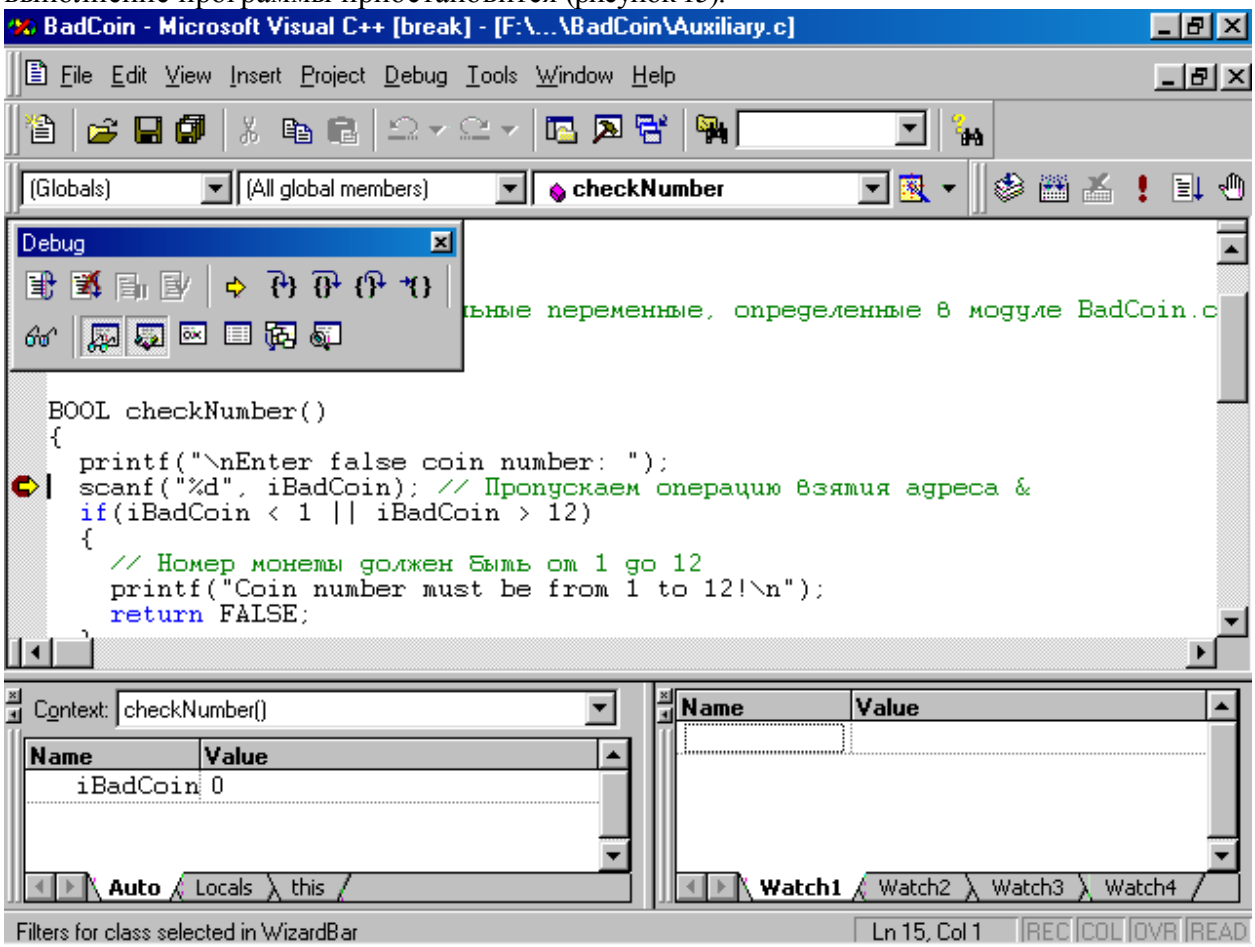


Рисунок 15 – До точки останова программа выполнилась успешно

Зовнішній вигляд IDE в режимі відладки істотно змінився. По-перше, змінився склад основного меню, по-друге, рядок, який виконуватиметься наступною, відмічений жовтою (за умовчанням) стрілкою; і, нарешті, з'явилися два нові вікна — **Variables** (змінні) і **Watch** (спостереження), які дозволяють переглядати і міняти вміст змінних.Внешний вид IDE в

режиме отладки существенно изменился. Во-первых, изменился состав основного меню, во-вторых, строка, которая будет выполняться следующей, отмечена желтой (по умолчанию) стрелкой; и, наконец, появилось два новых окна — **Variables** (переменные) и **Watch** (наблюдение), которые позволяют просматривать и менять содержимое переменных.

15. Отже, до цього моменту програма працює (поки не можна говорити, що повністю правильно, оскільки ще не було проведено тестування). Для покрокового виконання у відладчику є команди:Итак, до этого момента программа работает (пока нельзя говорить, что полностью правильно, т.к. еще не было проведено тестирование). Для пошагового выполнения в отладчике имеются команды:

Step Over (крок через)<**F10**> — виконує поточний оператор або функ-: цию і переходить до наступного рядка.**Step Over** (шаг через) <**F10**> — выполняет текущий оператор или функ-: цию и переходит к следующей строке.

Step Into (крок всередину)<**F11**>— виконує поточний оператор мови З або переходить до першого оператора функції.**Step Into** (шаг внутрь) <**F11**>— выполняет текущий оператор языка С или переходит к первому оператору функции.

Step Out (крок зовні)**<Shift>+<F11>** — завершує виконання поточної функції і переходить до рядка такою, що безпосередньо йде за її викликом.**Step Out** (шаг вне) **<Shift>+<F11>** — завершает выполнение текущей функции и переходит к строке непосредственно следующей за ее вызовом.

Run to Cursor (виконати до курсора)<Ctrl>+<F10> — виконує програму до рядка, де у нинішній момент знаходиться курсор.**Run to Cursor** (выполнить до курсора) <Ctrl>+<F10> — выполняет программу до строки, где в текущий момент находится курсор.

Оскільки функція **scanf**() — бібліотечна, натисніть функціональну клавішу <**F10**> **(Step Over).** Тут необхідно перемкнутися у вікно виконання програми і ввести прошений номер фальшивої монети. Після введення і натиснення клавіші <ENTER> відладчик негайно видасть повідомлення про помилку, яке відноситься до тільки що виконаної функції scanf() (малюнок 16).Поскольку функция **scanf**() — библиотечная, нажмите функциональную клавишу <**F10**> **(Step Over).** Здесь необходимо переключиться в окно выполнения программы и ввести запрашиваемый номер фальшивой монеты. После ввода и нажатия клавиши <ENTER> отладчик немедленно выдаст сообщение об ошибке, которое относится к только что выполненной функции **scanf**() (рисунок 16).

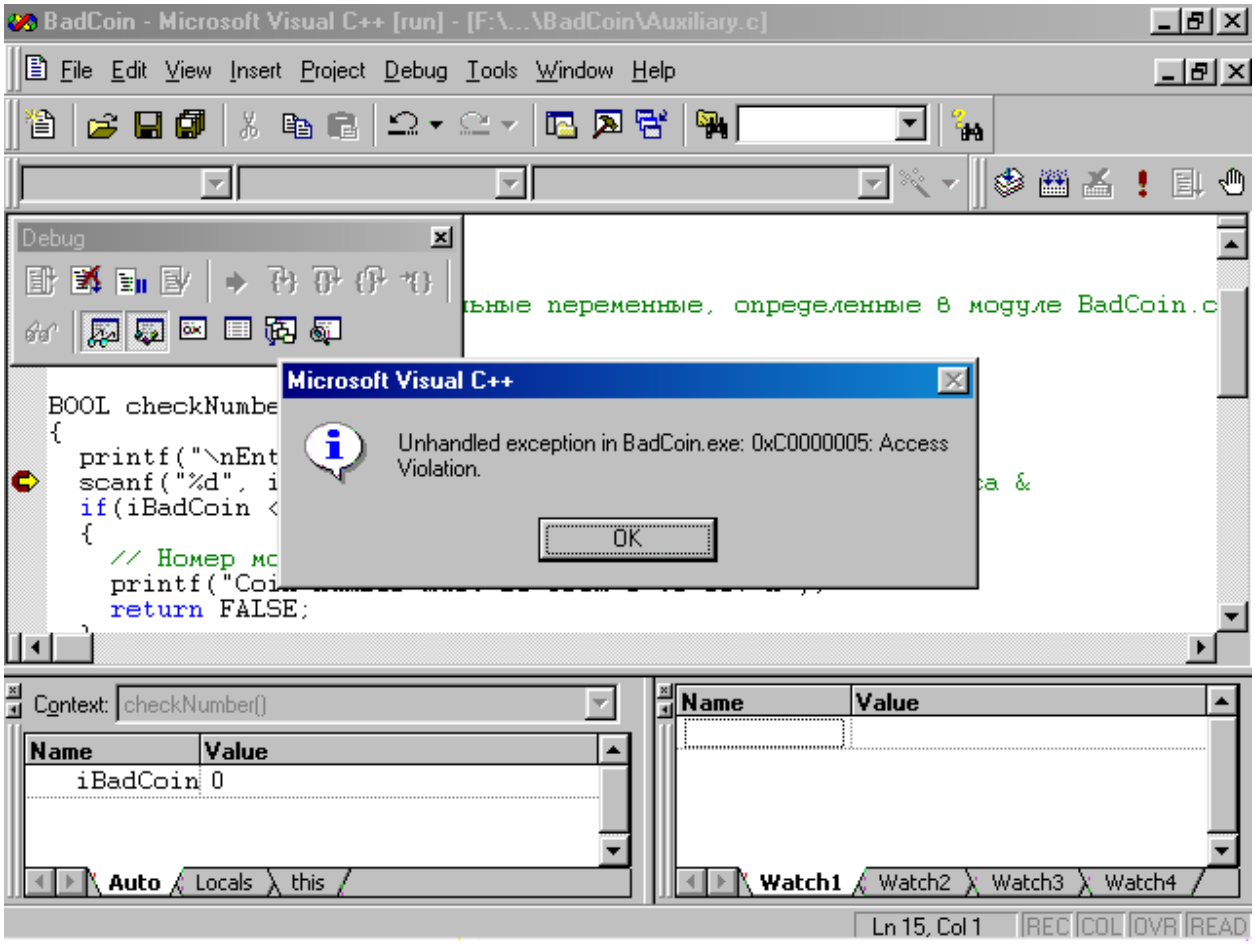


Рисунок 16 – Сообщение отладчика об ошибке

Це одна з найбільш поширених помилок. Річ у тому, що аргументи цієї функції, починаючи з другого, мають бути адресами (покажчиками) змінних, тип і кількість яких відповідає типам, заданим в рядку, що є першим аргументом. По цих адресах функція і заносить введені значення. У нас же замість цією вказано ім'я змінної :

scanf("%d", iBadCoin);scanf("%d", iBadCoin);

Зупиніть режим відладки командою **Stop Debugging** (перервати режим відладки) і внесіть виправлення — поставте перед змінною **iBadCoin** операцію узяття адреси (&), після чого запустите зборку програми командою **Build All.**Остановите режим отладки командой **Stop Debugging** (прервать режим отладки) и внесите исправления — поставьте перед переменной **iBadCoin** операцию взятия адреса (&), после чего запустите сборку программы командой **Build All.**

Зверніть увагу на вікно **Variables**, де відображена змінна **iBadcoin** і її поточне значення — воно співпадає з введеним. У цьому місці подивимося також на значення ще однієї змінної, яку ми ввели першою. Оскільки у вікні Variables відбиваються тільки змінні поточного блоку, то тут ви цю змінну не знайдете. Виконайте команду **Step Out**, щоб завершити виконання функції checkNumber() і повернутися у функцію main.Обратите внимание на окно **Variables**, где отображена переменная **iBadcoin** и ее текущее значение — оно совпадает с введенным. В этом месте посмотрим также на значение еще одной переменной, которую мы ввели первой. Поскольку в окне Variables отражаются только переменные текущего блока, то здесь вы эту переменную не найдете. Выполните команду **Step Out**, чтобы завершить выполнение функции checkNumber() и вернуться в функцию main.

Вміст вікна **Variables** змінився і тепер там представлені локальні змінні, задіяні у блоці оператора **switch**. Нас цікавить значення змінної **bweight**, яке має дорівнювати 1, якщо ви натиснули клавішу <h>, і 0, — якщо клавішу <**1**>. Таким чином, обидва введені параметри

мають правильні значення і можна спробувати виконати програму повністю, для чого натисніть клавішу <**F5**> (**Go**).

Тестування програми

Як і будь-який інший продукт виробництва, програма перед використанням має бути ретельно перевірена. Цей етап є чи не найскладнішим в усьому процесі створення програми — необхідно врахувати усі *можливі* варіанти її поведінки. Тому його треба починати не після завершення відладки, а одночасно з розробкою алгоритму.Как и любой другой продукт производства, программа перед использованием должна быть тщательно проверена. Этот этап является едва ли не самым сложным во всем процессе создания программы — необходимо учесть *все* возможные варианты ее поведения. Поэтому его нужно начинать не после завершения отладки, а одновременно с разработкой алгоритма.

Одним з шляхів перевірки або тестування програми є її виконання по одному разу з кожною з можливих комбінацій вхідних даних, В даному випадку ми так і поступимо, оскільки число варіантів, яке необхідно перевірити, невелике — всього 24, 12 номерів і дві "ваги" (це тільки "правильні" дані, а потрібно протестувати ще і неприпустимі). Проте у більшості випадків це досконало непрактично і очевидно, що необхідно як-небудь зменшити число необхідних перевірок.Одним из путей проверки или тестирования программы является ее выполнение по одному разу с каждой из возможных комбинаций входных данных, В данном случае мы так и поступим, поскольку число вариантов, которое необходимо проверить, невелико — всего 24, 12 номеров и два "веса" (это только "правильные" данные, а надо протестировать еще и недопустимые). Однако в большинстве случаев это совершенно непрактично и очевидно, что необходимо как-нибудь уменьшить число требуемых проверок.

Одним із способів такого зменшення є виконання випадкових тестів. Хоча випадкове тестування може виявити в програмі великі помилки, наприклад, програма постійно видає неправильний результат, дуже маловірогідно, щоб випадкове тестування виявило які-небудь нерегулярності в поведінці програми. Наприклад, програма, що використовує значення функції tg(x), може видавати ненадійні результати при x ~ тг/2, але випадкове тестування це навряд чи виявить. Для розробки тестів потрібно знати внутрішню структуру програми и-на її основі спробувати підібрати такий набір вхідних даних, який дозволить перевірити вагу гілки виконання програми. Звернемося приміром:Одним из способов такого уменьшения является выполнение случайных тестов. Хотя случайное тестирование может обнаружить в программе крупные ошибки, например, программа постоянно выдает неправильный результат, очень маловероятно, чтобы случайное тестирование обнаружило какие-либо нерегулярности в поведении программы. Например, программа, использующая значение функции tg(x), может выдавать ненадежные результаты при x ~ тг/2, но случайное тестирование это вряд ли обнаружит. Для разработки тестов надо знать внутреннюю структуру программы и-на ее основе попытаться подобрать такой набор входных данных, который позволит проверить вес ветви выполнения программы. Обратимся к примеру:

```
if (iBacICoin < 1 || iBadCoin > 12){
  printf("Coin number must be from 1 before 12!\n");
  return FALSE;return FALSE;
}
```

З цього фрагмента видно, що існують три випадки: два неприпустимих — iBadCoin < 1 і iBadCoin > 12, а також корртектный — 1 <= lEaciCoin <=12 . Таким чином, необхідно включити в тест дані, перевіряючі псові три випадки. При цьому будуть перевірені обидві гілки оператора ifИз этого фрагмента видно, что существуют три случая: два недопустимых — iBadCoin < 1 и iBadCoin > 12, а также корртектный — 1 <= lEaciCoin <=12 . Таким образом, необходимо включить в тест данные, проверяющие псе три случая. При этом будут проверены обе ветви оператора if,

А ось інший приклад:

```
do
  printf {"\nli' 'coin heavy press   (h),   otherwise -   (1) :);
whiie ( {iResuit - getch()} != 'h' && iResuit != 'l');
bWeight = (iResuit == 'h'? TRUE: FALSE;
...
if(bWeight = TRUE)
  aCoin[iBadCoin — 1] = 1;    // пряма гілкаaCoin[iBadCoin — 1] = 1;    // прямая ветвь
else
  aCoin[iBadCoin — 1] = — 1; // альтернативна гілкаaCoin[iBadCoin — 1] = -1; //
```

альтернативная ветвь

На перший погляд усе зрозуміло: якщо натиснута клавіша <h> чи <1>, те змінна bweight отримує, відповідно, значення true або false. І залежно від її значення деякому елементу масиву привласнюється або значення 1, або — 1. Знову три групи даних : <h>, <l> і усі інші клавіші. Перевірка покаже, що доки не натиснута одна з клавіш — <h> чи <1>, ми не вийдемо з циклу. Далі, залежно від введення, змінна bweight набуває або значення true, або — false. А ось далі ми маємо одну з труднообнаружимых помилок. Річ у тому, що ми ніколи не потрапимо в "альтернативну гілку", оскільки умова оператора if завжди буде істинною. Пов'язано це з тим, що замість оператора порівняння (==) ми використали оператор привласнення (=). І якщо ми просто запускатиме програму на "повному" наборі даних, то завжди отримуватимемо правильний номер фальшивої монети. Помилка виявиться тільки у тому випадку, якщо ми трохи змінимо програму і додатково визначатимемо, важче або легше виявлена фальшива монета — завжди отримуватимемо "важку" монету. У цьому ж варіанті помилку такого роду можна виявити тільки шляхом покрокового виконання програми.На первый взгляд все понятно: если нажата клавиша <h> или <1>, то переменная bweight получает, соответственно, значения true или false. И в зависимости от ее значения некоторому элементу массива присваивается либо значение 1, либо -1. Опять три группы данных: <h>, <l> и все остальные клавиши. Проверка покажет, что пока не нажата одна из клавиш — <h> или <1>, мы не выйдем из цикла. Далее, в зависимости от ввода, переменная bweight получает либо значение true, либо — false. А вот дальше мы имеем одну из труднообнаружимых ошибок. Дело в том, что мы никогда не попадем в "альтернативную ветвь", поскольку условие оператора if всегда будет истинным. Связано это с тем, что вместо оператора сравнения (==) мы использовали оператор присваивания (=). И если мы будет просто запускать программу на "полном" наборе данных, то всегда будем получать правильный номер фальшивой монеты. Ошибка выявится только в том случае, если мы немного изменим программу и будем дополнительно определять, тяжелее или легче обнаруженная фальшивая монета — всегда будем получать "тяжелую" монету. В данном же варианте ошибку такого рода можно выявить только путем пошагового выполнения программы.

Після виконання тестування програму можна вважати закінченою, хоча для складних програм відсутність помилок в роботі не означає, що програма правильна - помилки є присутніми, але себе ще не проявили. Проте, в лабораторній роботі розглядається проста програма, тому на цьому відладку і тестування програми можна вважати виконаною.После выполнения тестирования программу можно считать законченной, хотя для сложных программ отсутствие ошибок в работе не означает, что программа правильна – ошибки присутствуют, но себя еще не проявили. Однако, в лабораторной работе рассматривается простая программа, поэтому на этом отладку и тестирование программы можно считать выполненным.

Порядок виконання роботи

1. Прочитайте опис лабораторної роботи. Відповідно до умов вирішуваної задачі складіть граф-схему алгоритму. Він має бути виконаний акуратно.

Підготуйте письмовий звіт.

2. Покажіть поисьменный звіт викладачеві і з його дозволу приступайте до виконання лабораторної роботи.

3. Скопіюйте у свою теку заготівлі файлів, запустіть Visual C++ і створіть проект.Скопируйте в свою папку заготовки файлов, запустите Visual C++ и создайте проект.

4. Виконайте усі кроки, пов'язані з відладкою і тестуванням програми.

5. Захистіть звіт про виконану лабораторну роботу.Защитите отчет о выполненной лабораторной работе.

6. Закінчіть сеанс роботи з операційною системою.

Звіт повинен містити:

1) Найменування і мета роботи.
2) Опис вирішуваної задачі і граф-схему алгоритму.
3) Тексти програм з необхідними коментарями.
4) Висновки по лабораторній роботі.

Контрольні питання

1. Перерахуйте основні етапи розробки програм.

2. Що таке інтегроване середовище розробки (IDE)?Что такое интегрированная среда разработки (IDE)?

3. Поясніть алгоритм рішення задачі, що розглядається в лабораторній роботі?

4. Що таке функціональна декомпозиція і для чого вона використовується?

5. Чому програму розбивають на модулі?

6. Які модулі використані в лабораторній роботі?

7. Як повідомляється програмі про те, яка монета є фальшивою?

8. Поясніть призначення функції scanf() в лабораторній роботі.Объясните назначение функции scanf() в лабораторной работе.

9. Для чого в програмі потрібний заголовний модуль?

10. Як відбувається зборка програми в лабораторній роботі?

11. Як відбувається локалізація помилок в програмі за допомогою коментарів?

12. Перерахуйте види помилок в програмах.

13. Для чого потрібне тестування програми?

ЛАБОРАТОРНА РОБОТА 2

ПРОГРАМУВАННЯ ТИПОВИХ ЗАВДАНЬ

Мета роботи — Вивчити основні етапи створення консольних застосувань, навчитися оформляти графічну схему алгоритму, освоїти методику відладки і тестування програми за допомогою інтегрованого середовища пакету Visual C++ 6.0.

Частина 1 - Програма лінійної структури

Завдання на виконання частини 1

Розробити програму, в якій використовується оператор привласнення і арифметичне вираження. Програма повинна містити:

1) оголошення констант і скалярних змінних типу float;объявление констант и скалярных переменных типа float;
2) привласнення змінним заданих початкових значень;
3) обчислення арифметичного вираження трьома способами :
   а) з використанням проміжних змінних;а) с использованием промежуточных переменных;
   б) за допомогою одного арифметичного вираження;б) с помощью одного арифметического выражения;

в) за допомогою вираження, розташованого в списку даних оператора printf;в) с помощью выражения, расположенного в списке данных оператора printf;

4) усі проміжні і остаточні результати вивести на екран з ідентифікацією виведених значень.все промежуточные и окончательные результаты вывести на экран с идентификацией выведенных значений.

Відразу необхідно передбачити перевірку введення даних, продумати питання відладки і тестування програми.

За результатами відладки програми виконати друк:

1) тексту програми з середовища за допомогою команди редактора для роботи з блоком;
2) результатів виконання програми з екрану за допомогою команди Shift+PrintScreen.

Приклад програми

Дана формула для обчислення значення змінної у виді:

де x = 0,8где x = 0,8

C = 0,7C = 0,7

Розробити програму для обчислення значення змінної Z з использо-ванием арифметичного вираження і оператора привласнення. Задати значення X при його оголошенні, а значенню З присвоїти значення за допомогою оператора привласнення в тексті програми.

Текст програми прикладу представлений в лістингу 1.

Лістинг 1

```
#include <conio.h>   // - підключення бібліотекinclude <conio.h>   // - подключение библиотек
#include <stdio.h>
#include <math.h>
void main(){
   float x = 0.8, c, z, a, b;
   clrscr();  // - очищення екрануclrscr();  // - очистка экрана
   printf("Початкові дані:");printf("Исходные данные:\n");
   printf ("x = %5.2f\n",  x);
   з = 0.71;
  printf("з - %5.2f", c);printf("c - %5.2f\n", c);
// 1-й спосіб обчислень
printf("Проміжні результати:  ");printf("Промежуточные результаты:  \n");
  a = log(x*x т c);
  printf ("      a = %5.2f",  a);
  b = 112.37  * a;
  printf("      b - %.2f",  b);
  z = b/3;
  printf("      z - %.6e\n'\  z);
  // 2-й спосіб обчислень
  z =  112.37 * log(x*x + c)/3;z =  112.37 * log(x*x + c)/3;
  printf("Результат за допомогою одного вираження:");printf("Результат с помощью одного выражения:\n");
  printf {"  z = 112.37 * log(x*x + c)/3 = %.6e\n", z);
// 3-й спосіб обчислень
  printf("Результат за допомогою вираження зі списку даних printf :");printf("Результат с помощью выражения из списка данных printf:\n");
  printf ("%28c  z = %.6e\n"/  ' ',  112.37 * log(x*x + c)/3);
  printf("Для завершення програми натисніть будь-яку клавішу");printf("\nДля завершения программы нажмите любую клавишу\n");
  getch(); //  - натиснути будь-яку клавішуgetch(); //  - нажать любую клавишу
}            // - кінець функції mainконец функции main
```

Контрольні питання

1) Назвіть етапи процесу обробки програм.
2) Що таке компіляція, компонування?
3) Поясніть оголошення змінних програми.
4) Що визначає тип даних?
5) Які основні характеристики даних типу float?
6) Поясніть призначення і форму оператора привласнення.
7) Яке призначення виразів?
8) Які ви знаєте типи виразів?
9) З яких елементів формуються вирази?
10) Поясніть правила виконання виразів.
11) Яке призначення арифметичного вираження?
12) Що може бути операндом арифметичного вираження?
13) Назвіть арифметичні операції в порядку убування їх пріоритету.
14) У якій послідовності виконуються операції арифметичного выра-
15) жения?
16) У яких конструкціях мови Сі можна використати арифметичне вираження?
17) Назвіть приклади функцій, які можна використати в арифметичних виразах.
18) Яке призначення оператора printf і його параметрів?
19) Що може бути елементом списку аргументів оператора printf?

Часть 2 - Алгоритм и программа разветвляющейся структуры для работы в режиме диалога

Целью работы является освоение:

1) объявления скалярных переменных целого и строкового типа;
2) операторов для ввода данных с клавиатуры:
   а) gets - для ввода данных строкового типа;
   б) scanf ~ для ввода данных целого типа;
3) вывода в Watch - окно просмотра введенных значений;
4) оператора printf для вывода данных целого и строкового типа на экран;
5) оператора if - разветвления вычислений.

ЗАДАНИЕ НА ВІПОЛНЕНИЕ РАБОТЫ

Разработать программу, в состав которой должны входить:
объявление скалярных переменных типа char и int;
выдача пользователю на экран запроса в виде текстовой строки;
ввод с клавиатуры ответа в виде текстовой строки;
ввод с клавиатуры ответа в виде целого значения;
вывод на экран введенных значений;
анализ введенного значения и выдача на экран ответа по результатам анализа.

В листинге 8.2 приведена программа примера. Из программы задаются пользователю вопросы, на которые он дает ответы. Значения ответов вводятся в переменные:

Name — имя пользователя, строковая переменная;
let - возраст пользователя, переменная типа int.

Затем введенные значения выводятся на экран. Кроме того, значение переменной let анализируется с помощью оператора if.

Листинг 2 - Диалог с пользователем.

```
#include <conio.h> // - подключение библиотек
#include <stdio.h>
void main() {
  char Name[15];   // - строковая переменная
  int let;  // - целая переменная
  clrscr();  // - очистка экрана
  printf("Здравствуйте!\n Скажите, пожалуйста, как вас зовут?\n");
  gets(Name);        // Ввод строки с клавиатуры
  printf("Ах - %s - какое красивое имя!\n", Name);
  printf("А сколько Вам лет, если не секрет?..\n");
  scanf ("%i", &let);   // - ввод числа с клавиатуры
  // Анализ значения let:
  if (let < 20)
    printf("О! %i - милый юный возраст!. \n"
    "До скорого свидания! Приятно было познакомиться.", let);
  else if (let < 50)
    printf("Да! - %i - это уже серьезно..\n", let);
  else
    printf("Ну что же, и в %i в жизни все еще очень"
          много интересного !\n", let );
  printf("\nДля завершения программы нажмите любую клавишу\n");
  getch();   // - приостановка до нажатия любой клавиши
}
```

Контрольные вопросы

1) Каковы основные характеристики данных типа int, char?
2) Каково назначение операторов gets, scanf и их параметров?
3) Как вывести (удалить) значение переменной программы в окно просмотра Watch?
4) Как перейти из окна редактора в окно просмотра (вывода результатов) и обратно?
5) Как изменить размер окна: редактора, просмотра, вывода результатов?
6) Как выполнить программу построчно, до заданной строки, до конца?

Часть 3 - Программа с разными типами выражений и с циклами

Целью работы является освоение:

1) операторов, в которых можно использовать выражения отношения и логические;
2) операндов, операций и результатов операций отношения, логических, над символами и строками и приоритетов операций;
3) функций для обработки строковых значений;
4) просмотра результатов выражений любого типа в процессе отладки программы.

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ РАБОТЫ

Разработать программу, в которой используются операторы присваивания и выражения: арифметические, отношения, логические. Программа должна содержать:

объявление скалярных переменных следующих типов: целого, вещественного и строкового;

присваивание переменным заданных начальных значений;

вычисление выражений отношения, логических, над символами и строками;

вывод на экран результатов выражений различных типов с идентификацией выведенных значений;

определение машинного представления значений целого типа с использова
нием операции сдвига.

По завершении отладки программы отпечатать программу из среды, а результаты - с экрана.

Пример программы

На рисунке 2 приведен фрагмент схемы алгоритма примера. В листинге 8.3 -текст программы примера.

Листинг 3. Программа с циклами и разными типами выражений

```
#include <string.h>     // - подключение библиотек
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
FILE *f;
void main() {
  int i = 12, i = 34, k = 0;
  float a = 12.34,  b = 0,  c = 1;
  char * d = "Borland ", *e = "C++";  // - инициализация строковых переменных
int L = strlen(d) + strien(e);// - определение длины суммы строк
char * h =  (char *)  malloc  (L),       // - запрос ОП для строки
* p = "aaa", * q = "aaaa";
// f = fopen("03.rez", %'");    // - для вывода в файл
f = fopen ("con", "w");     // - для вывода на экран
clrscrO; // - очистить экран
fprintf (f, "\nНачало работы программы\n"
"Целые числа:  i = %d j = %k - %d\n" "Вещественные числа : a = %f b - %f c = %f\n",
i, j, k, a, b, c) ; fprintf (f, "Работа с логическими значениями.\n"
"Логические операции:\n  ! - отрицание:\n" "!a = %d  !b = %d  !i = %d    - любые типы данных\п" 11 && - логическое умножение:\n" "i && j = Ы i && k - %d      - целые \n" "a && b = %d a && c = %d     - Еещественные\п" "  || - логическое сложение:\п" "i I| ; = %d i !| k = %d     - целые\n" "a I i b = %d a | I c = -od     - вещественные\п", !a,  !bf   !i,
i & & j,  i & & k,  a & & b,  a & & c, i I I j,  i I I k,  a || b,  a || c); fprintf(f, "Работа со строками\п"); strcpy(h,""");  /7 - очистка значения строки strcat(h,d);  // - сцепление строк h и d scrcat(h,e);  // - сцепление строк h и e fprintf (f, "\п Результат сцепления строк: h = %s\n", h); fprintf(f, "\n Результаты сравнения строк:\п");
fprintf (f, " p = q =: %d\n p = p ^ %d\n q = p = %d\nM,  // - вывод О
strcmp(p,q) ,strcmp(p,p") ,strcmp(q,p) ) ;  II - вывод числа
 ( O
printf("\nДля продолжения программы нажмите любую клавишу\n");
getch(); clrscr ();
fprintf(f, "Работа со сдвигами\п");
fprintf(f, "Двоичное представление целых чисел:\п");
for (k = 1; к <= 15; к++)  // - перебор чисел
{ j - 16384;  // - j = 214
fprintf (f, " к = %2i = ", к );
for (i = 15; i >= 1; i--)  // - перебор номеров разрядов
//  двоичного числа { if (к & j) fprintf (f, "1");     // - начало блока for i
else fprintf (f, "0"); i = j » 1;  }       // - сдвиг j вправо
fprintf (f, "\n");  II - переход на следующую строку
free(h);  // - освобождение ОП
printfC^fl завершения программы нажмите любую клавишу");
  fclose(f);
```

getch();

КОНТРОЛЬНІ ПИТАННЯ

Назвіть основні характеристики цих наступних типів : цілих, речових, логічних, символьних і строковихНазовите основные характеристики данных следующих типов: целых, вещественных, логических, символьных и строковых,

Яке призначення виразів : стосунки, логічних; що є їх результатом?Каково назначение выражений: отношения, логических; что является их результатом?

З яких елементів формуються названі твані виразів?

У яких конструкціях мови Сі ++ можна використати названі типи виразів?В каких конструкциях языка Си ++ можно использовать названные типы выражений?

Що може бути операндом названих типів виразів?

Назвіть приклади функцій, які можна використати у виразах названих типів.

Назвіть операції різних типів в порядку убування їх пріоритету.

У якій послідовності виконуються операції змішаних виразів?

Як проконтролювати значення змінних названих типів в процесі виконання програми?

Частина 4 - **Програма з використанням масивів і текстових файлів початкових даних і результатів**Часть 4 - **Программа с использованием массивов и текстовых файлов исходных данных и результатов**

Метою роботи є освоєння:

1) оголошення масивів речових і строкових даних;
2) основних характеристик масивів даних;
3) операторів fopen, for, fclose;
4) введення з текстового файлу цих масивів строкового і речового типу;
5) висновку в текстовий файл результатів виконання програми.

ЗАВДАННЯ НА ВИКОНАННЯ роботиЗАДАНИЕ НА ВЫПОЛНЕНИЕ работы

Розробити схему алгоритму і програму для введення початкових даних з текстового файлу, обробки речових даних і виведення результатів в текстовий файл. У програмі повинні використовуватися оператори fopen, for, fclose. Початкові дані мають бути у складі шапки таблиці, стовпця таблиці з текстовою інформацією, декількох стовпців таблиці з речовими значеннями. Програма повинна включати:Разработать схему алгоритма и программу для ввода исходных данных из текстового файла, обработки вещественных данных и вывода результатов в текстовый файл. В программе должны использоваться операторы fopen, for, fclose. Исходные данные должны быть в составе шапки таблицы, столбца таблицы с текстовой информацией, нескольких столбцов таблицы с вещественными значениями. Программа должна включать:

1) оголошення масивів строкових і речових даних;
2) відкриття текстових файлів для читання (введення) даних і для запису (висновку) даних;открытие текстовых файлов для чтения (ввода) данных и для записи (вывода) данных;
3) введення і виведення шапки таблиці;
4) введення початкових даних : строкових і речових;
5) виведення таблиці у вигляді шапки, рядків таблиці, розділених горизонтальними лініями, і стовпців, розділених вертикальними лініями;
6) обробку числових значень таблиці відповідно до заданого варіанту;
7) виведення результатів виконання програми :
   в процесі відладки програми - на екран;
   після відладки остаточні результати - в текстовий файл;
8) закриття файлів.

Приклад програми

Дані початкові показники, приведені в таблиці 1.

Тип	Количество	Загрузи» самолет*	
		Пассажир	Груз, т
Ил~6	**15**	**54**	3,4
Ан-	**16**	**25**	***2,***\
Ту-	**9**	106	**20,7**
Ил-	**5**	170	57,6

Для ввода данных с помощью программы эти показатели представлены в текстовом файле (физический файл Ir4.dat) в виде, приведенном на рис.8.3.

РАСЧЕТ ОТПРАВОК АЭРОПОРТА

Тип	Количество	Загрузка	
		Пассажир	

Рис. 8.3F. ИСХОДНЫЕ ДАННЫЕ ДЛЯ ВВОДА, ФАЙЛ LR4.DAT

В листинге 8.4 приведен текст программы примера.

Листинг 8.4. ИСПОЛЬЗОВАНИЕ МАССИВОВ И текстовых ФАЙЛОВ ИСХОДНЫХ ДАННЫХ И РЕЗУЛЬТАТОВ,

```
#include <conio.h> // - подключение библиотек
#include <stdic.h>
void main() {
  FILE *fid, *frz;
char a[8][57],// - массив для шапки
b[4][7]; // - для текста таблицы
float c[4][3]; // - массив для чисел
int i, j;  float max;
II frz = fopen(t'04.rez"/ "w11);    // - для вывода в файл
frz = fopen ("con", "w");     // - для вывода на' экран
clrscr(); // - очистить экран
fprintf(frz, "Исходные данные:\n");
•//Ввод и вывод шапки таблицы:
fid = fopen(Mlr4.dat"/ иг");  // - открытие fid ьля чтения for (i = 0; i < 8; i++ ) fgets(a[i], 57,
fid); for (i - 0; i < 6; i++ ) fputs(a[i], frz);
// Ввод исходных данных:
for ( i = 0; i < 4; i++ )
{    fscanfffici,  "%6c",  b[i]); b[i][6] =  f\0';
for  (j = 0;  j  < 3;  j++ )  fscanf(fid,  "%f",  &c[i][j]);
fscanf  {fid,  " \n",  Шах); // - пропуск \п из файла
/Вывод данных:-
for  ( i - 0;  i < 4;  i+т )
{ fprintf(frz, "! %s ! %9.0f j %9.0f   ! %1.2e !\n", b[i]  [i][0]  [i][l]  [i][2) )
]  [, if ( i < 3 ) fprintf(frz, "%s", a[6]);
else fprintf (frz, "Is", a[7]); ]
// Поиск и вывод максимального значения массива С:
  max = c[0][0];
  for (i = 0; i < 4; i++)
    for ( j = 0; j < 3; 1++ )
     if (c[i][j] > max)
       max = c[i][jl; fprintf (fr?, "max = %7.2e\nM, max);
  fclose(fid);
  fclose(frz);
  printf("Для завершения программы нажмите любую клавишу");
```

```
getch(); )
}
```

Контрольні питання

1) Поясніть оператори оголошення масивів строкових і речових даних.
2) У якій послідовності розташовуються в ОП елементи одновимірних масивів і який об'єм ОП вони займають?В какой последовательности располагаются в ОП элементы одномерных массивов и какой объем ОП они занимают?
3) Поясніть відповідність елементів схеми алгоритму і операторів програми.
4) Що таке логічний (фізичний) файл?
5) Поясніть правила введення з файлу строкових і речових даних.
6) Поясніть правила форматного виведення даних символьного, строкового і речового типів.Поясните правила форматного вывода данных символьного, строкового и вещественного типов.
7) Поясніть послідовність виконання операторів програми.

Порядок виконання роботи

1. Прочитайте опис лабораторної роботи. Відповідно до умов вирішуваної задачі складіть граф-схему алгоритму. Він має бути виконаний акуратно.

Підготуйте письмовий звіт.

2. Покажіть поисьменный звіт викладачеві і з його дозволу приступайте до виконання лабораторної роботи.

3. Скопіюйте у свою теку заготівлі файлів, запустіть Visual C++ і створіть проект.Скопируйте в свою папку заготовки файлов, запустите Visual C++ и создайте проект.

4. Виконайте усі кроки, пов'язані з відладкою і тестуванням програми.

5. Захистіть звіт про виконану лабораторну роботу.Защитите отчет о выполненной лабораторной работе.

6. Закінчіть сеанс роботи з операційною системою.

Звіт повинен містити:

1) Найменування і мета роботи.
2) Опис вирішуваної задачі і граф-схему алгоритму.
3) Тексти програм з необхідними коментарями.
4) Висновки по лабораторній роботі.

Контрольні питання

1. Перерахуйте основні етапи розробки програм.

2. Що таке інтегроване середовище розробки (IDE)?Что такое интегрированная среда разработки (IDE)?

3. Поясніть алгоритм рішення задачі, що розглядається в лабораторній роботі?

4. Що таке функціональна декомпозиція і для чого вона використовується?

5. Чому програму розбивають на модулі?

6. Які модулі використані в лабораторній роботі?

7. Як повідомляється програмі про те, яка монета є фальшивою?

8. Поясніть призначення функції scanf() в лабораторній роботі.Объясните назначение функции scanf() в лабораторной работе.

9. Для чого в програмі потрібний заголовний модуль?

10. Як відбувається зборка програми в лабораторній роботі?

11. Як відбувається локалізація помилок в програмі за допомогою коментарів?

12. Перерахуйте види помилок в програмах.

13. Для чого потрібне тестування програми?

Лабораторна робота 3

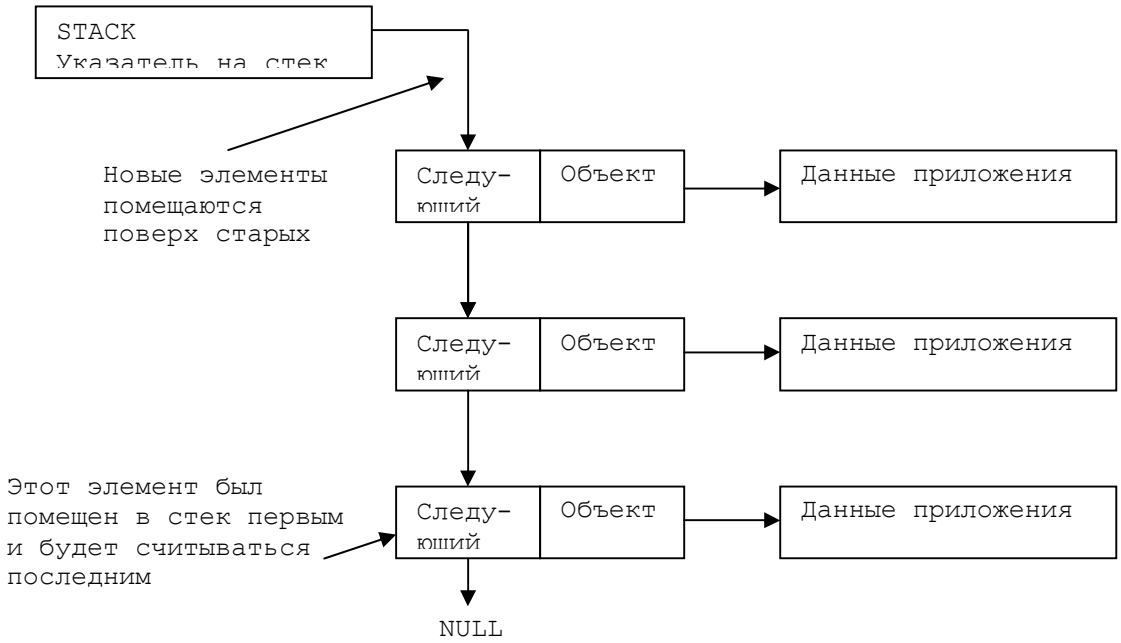
Структури даних типу списків

Мета роботи Вивчити структури даних типу списків : стеки, лінійні черги і списки, бінарні дерева; навчитися реалізовувати основні операції над списками.

Частина 1. Стеки

У лабораторній роботі розглядаються структури даних, які широко використовуються в системному і прикладному програмуванні.

Стік - це структура даних, яка підкоряється двом правилам. Перше правило полягає в тому, що нові елементи можна поміщати тільки на вершину стека; це називається занесенням (**pushing**) елементу. Другу правило полягає в тому, що брати елементи можна тільки з вершини стека; це називається витяганням (**popping**). У короткій формі ці правила містяться в акрониме (скороченні) LIFO (**Last In, First Out** - останнім прийшов, деремо пішов). Стік можна порівняти із стопкою тарілок - тарілку можна узяти згори, покласти тарілку зручніше вгору



стопки. Малюнок 1 ілюструє роботові стека.Стік - це структура даних, яка підкоряється двом правилам. Перше правило полягає в тому, що нові елементи можна поміщати тільки на вершину стека; це називається занесенням (**pushing**) елементу. Друге правило полягає в тому, що брати елементи можна тільки з вершини стека; це називається витяганням (**popping**). У короткій формі ці правила містяться в акрониме (скороченні) LIFO (**Last In, First Out** - останнім прийшов, першим пішов). Стек можна порівняти із стопкою тарілок - тарілку можна узяти згори, покласти тарілку зручніше вгору стопки. Малюнок 1 ілюструє роботу стека.Стік - це структура даних, яка підкоряється двом правилам. Перше правило полягає в тому, що нові елементи можна поміщати тільки на вершину стека; це називається занесенням (**pushing**) елементу. Друге правило полягає в тому, що брати елементи можна тільки з вершини стека; це називається витяганням (**popping**). У короткій формі ці правила містяться в акрониме (скороченні) LIFO (**Last In, First Out** - останнім прийшов, першим пішов). Стек можна порівняти із стопкою тарілок - тарілку можна узяти згори, покласти тарілку зручніше вгору стопки. Малюнок 1 ілюструє роботу стека.Стек – это структура данных, которая подчиняется двум правилам. Первое правило заключается в том, что новые элементы можно помещать только на вершину стека; это называется занесением (**pushing**) элемента. Второе правило состоит в том, что брать элементы можно только с вершины стека; это называется извлечением

(**popping**). В краткой форме эти правила содержатся в акрониме (сокращении) LIFO (**Last In, First Out** – последним пришел, первым ушел). Стек можно сравнить со стопкой тарелок – тарелку можно взять сверху, положить тарелку удобнее наверх стопки. Рисунок 1 иллюстрирует работу стека.

Малюнок 1 - Стік

Стеки часто використовуються компіляторами для організації виклику функцій, передачі ним аргументів і отримання результату, а також в додатках, зокрема, в калькуляторах.

Максимальне число елементів, які можна розмістити в стеку, не повинне обмежуватися програмним оточенням: у міру заштовхування в стек нових елементів і виштовхування старих, пам'ять під нього повинна динамічно проситися і звільнятися. Стан стека розглядається тільки по відношенню до його вершини, а не до усього вмісту.

Операції, що виконуються над стеком, мають спеціальні назви. При до-бавлении елементу в стек **s** елементу **sItem** визначена операція **push (s, sItem).** Аналогічним чином визначається операція виштовхування із стека — **pop(s)** 3 по якій із стека s "верхній" елемент віддаляється і повертається в якості значення функції. Отже, операція привласненняОперации, выполняемые над стеком, имеют специальные названия. При до-бавлении элемента в стек **s** элемента **sItem** определена операция **push (s, sItem).** Аналогичным образом определяется операция выталкивания из стека — **pop(s)**3 по которой из стека s "верхний" элемент удаляется и возвращается в качестве значения функции. Следовательно, операция присваивания

sItem = pop(s); sItem = pop(s);

видалить елемент із стека і присвоїть його значення змінної **sItem**. На применекие операції виштовхування із стеки існує єдине обмеження: вона не може застосовуватися до *порожнього* стека, т.е, до такого, який не містить жодного елементу.удалит элемент из стека и присвоит его значение переменной **sItem**. На применекие операции выталкивания из стеки существует единственное ограничение: она не может применяться к *пустому* стеку, т.е, к такому, который не содержит ни одного элемента.

Окрім цих двох основних операцій часто буває необхідно прочитати елемент у вершині стека, не витягаючи його звідти. Для цих цілей використовують операцію **peek(s).**Помимо этих двух основных операций часто бывает необходимо прочитать элемент в вершине стека, не извлекая его оттуда. Для этих целей используют операцию **peek(s).**

Найчастіше використовуються два підходи до реалізації стека : на базі масивів і динамічний. Оскільки перший підхід рідше використовується, в лабораторній роботі використані динамічні структури данихНаиболее часто используются два подхода к реализации стека: на базе массивов и динамический. Поскольку первый подход реже используется, в лабораторной работе использованы динамические структуры данных,

У лістингу.1 приведений заголовний файл для програм роботи із стеком, в якості базового вибраний для його елементів цілий тип.В листинге.1 приведен заголовочный файл для программ работы со стеком, в качестве базового выбран для его элементов целый тип.

Лістинг 1

```
/*Інтерфейс для роботи із стеком*/
#define STACK struct stack
STACK{
 int   info;
 STACK *next;
};

extern void push(STACK **ppStack, int nItem);
extern int  pop(STACK **ppStack, int *nError);
extern int  peek(STACK **ppStack, int *nError);
```

Зверніть увагу на друге поле структури — покажчик на структуру STACK. Такий спосіб рекурсивного визначення, при якому поле структури містить посилання на саму себе, є абсолютно допустимим. Компілятор відводь^ під покажчик next необхідна кількість пам'яті незалежно від того, на який об'єкт він вказує.Обратите внимание на второе поле структуры — указатель на структуру STACK. Такой способ рекурсивного определения, при котором поле структуры содержит ссылку на саму себя, является абсолютно допустимым. Компилятор отводи^ под указатель next требуемое количество памяти независимо от того, на какой объект он указывает.

Ще один момент, на який слід звернути увагу, полягає в тому, що у функціях push і pop використовуються подвійні посилання. Завдяки цьому кожна з них може повертати в якості результату своєї роботи покажчик на новий елемент STACK (використовується передача параметрів за адресою). Вхідним параметром вказаних функцій є покажчик на стек, з використанням якого обчислюється і повертається нова адреса елементу, що знаходиться на вершині. Після кожної операції заштовхування і виштовхування покажчик, пов'язаний з вершиною, міняється.Еще один момент, на который следует обратить внимание, заключается в том, что в функциях push и pop используются двойные ссылки. Благодаря этому каждая из них может возвращать в качестве результата своей работы указатель на новый элемент STACK (используется передача параметров по адресу). Входным параметром указанных функций является указатель на стек, с использованием которого вычисляется и возвращается новый адрес элемента, находящегося на вершине. После каждой операции вталкивания и выталкивания указатель, связанный с вершиной, меняется.

Функції **pop** і **peek** мають ще один параметр цілого типу, що передається через покажчик, — **error**. Якщо в стеку є хоч один елемент, то функції повертають error = 0. Якщо ж стек порожній, тобто не має сенсу видалити або прочитувати з нього що-небудь, **error** набуває значення 1. Реалізація функцій роботи із стеком представлена в лістингу 2.Функции **pop** и **peek** имеют еще один параметр целого типа, передаваемый через указатель, — **error**. Если в стеке есть хоть один элемент, то функции возвращают error = 0. Если же стек пуст, т.е. не имеет смысла удалить или считывать из него что-либо, **error** принимает значение 1. Реализация функций работы со стеком представлена в листинге 2.

Листинг 2

```
/* Реализация функций работы со стеком */
#include <malloc.h>
#include "stack.h"
void push(STACK **ppStack, int nItem) {
 STACK *pNewItem;
  // Запрашиваем память под структуру для элемента стека
 pNewItem = (STACK *)malloc(sizeof(STACK));
  // Заполняем поля структуры - информационное и
  // указателя на следующий элемент
 pNewItem->info = nItem;
 pNewItem->next = *ppStack;
  // Устанавливаем новый указатель на вершину стека
 *ppStack = pNewItem;
}
int pop(STACK **ppStack, int *nError) {
  // Запоминаем "старый" адрес вершины стека
 STACK *pOldItem = *ppStack;
 int   nOldInfo = 0;
 if(*ppStack) {
    // Если стек не пустой, извлекаем элемент ...
  nOldInfo = pOldItem->info;
```

```
    *ppStack = (*ppStack)->next;
     // и освобождаем память
    free(pOldItem);
    *nError = 0;
   }
   else
     // В противном случае ошибка
    *nError = 1;
   return nOldInfo;
  }
  int peek(STACK **ppStack, int *nError)  {
   if(*ppStack)  {
     // Если стек не пустой, то читаем информацию об элементе
     // не удаляя его из стека
    *nError = 0;
    return (*ppStack)->info;
   }
   else  {
     // В противном случае ошибка
    *nError = 1;
    return 0;
   }
  }
```

В листинге 3 приведена простейшая тестовая программа, иллюстрирующая работу со стеком.

Листинг 3

```
  /* Простейшая программа, иллюстрирующая работу со стеком */
  #include <stdio.h>
  #include "stack.h"
  STACK *s1, *s2;
  int main()  {
   int nError;
    // Помещаем в стек число 12
   push(&s1, 12);
   printf("\npeek(s1) = %d", peek(&s1, &nError));  // проверяем
  // Помещаем в стек число 13
   push(&s1, 13);
   printf("\npeek(s1) = %d", peek(&s1, &nError));  // проверяем
  // Помещаем в стек число 14
   push(&s1, 14);
   printf("\npeek(s1) = %d", peek(&s1, &nError));  // проверяем
   push(&s1, 15);
    // Помещаем в стек число 15
   printf("\npeek(s1) = %d\n", peek(&s1, &nError)); // проверяем
    // Извлекаем элементы из одного стека и помещаем в другой
   push(&s2, pop(&s1, &nError));
   push(&s2, pop(&s1, &nError));
   push(&s2, pop(&s1, &nError));
   push(&s2, pop(&s1, &nError));
    // Распечатываем результаты, одновреено освобождая стек
   printf("\npop(s2)  = %d", pop(&s2, &nError));
```

```
  printf("\npop(s2)  = %d", pop(&s2, &nError));
  printf("\npop(s2)  = %d", pop(&s2, &nError));
  printf("\npop(s2)  = %d\n", pop(&s2, &nError));
   getchar();
  return 1;
}
```

Результат работы программы представлен на рисунке 3.

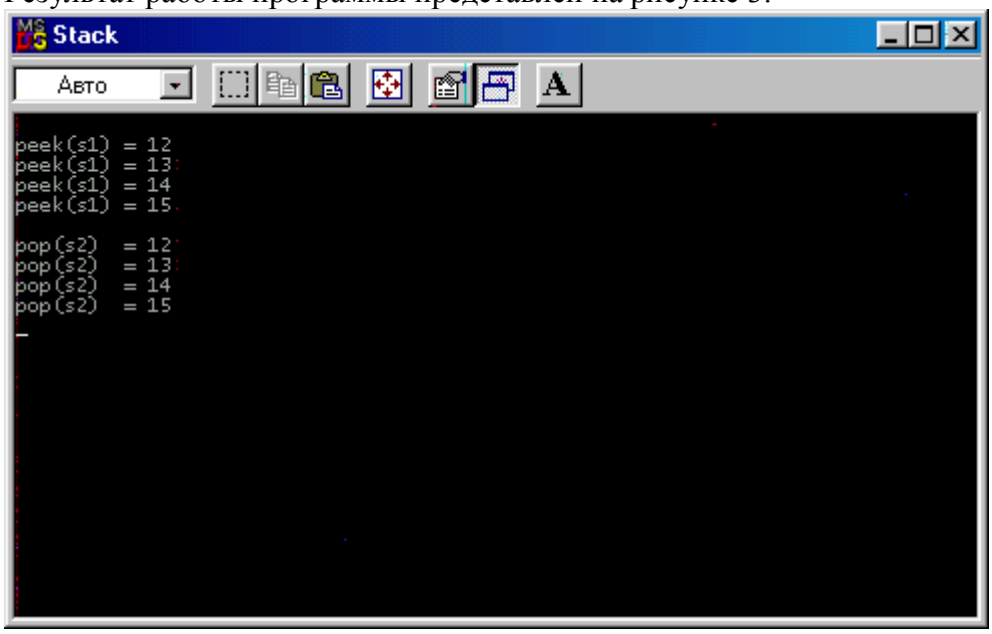
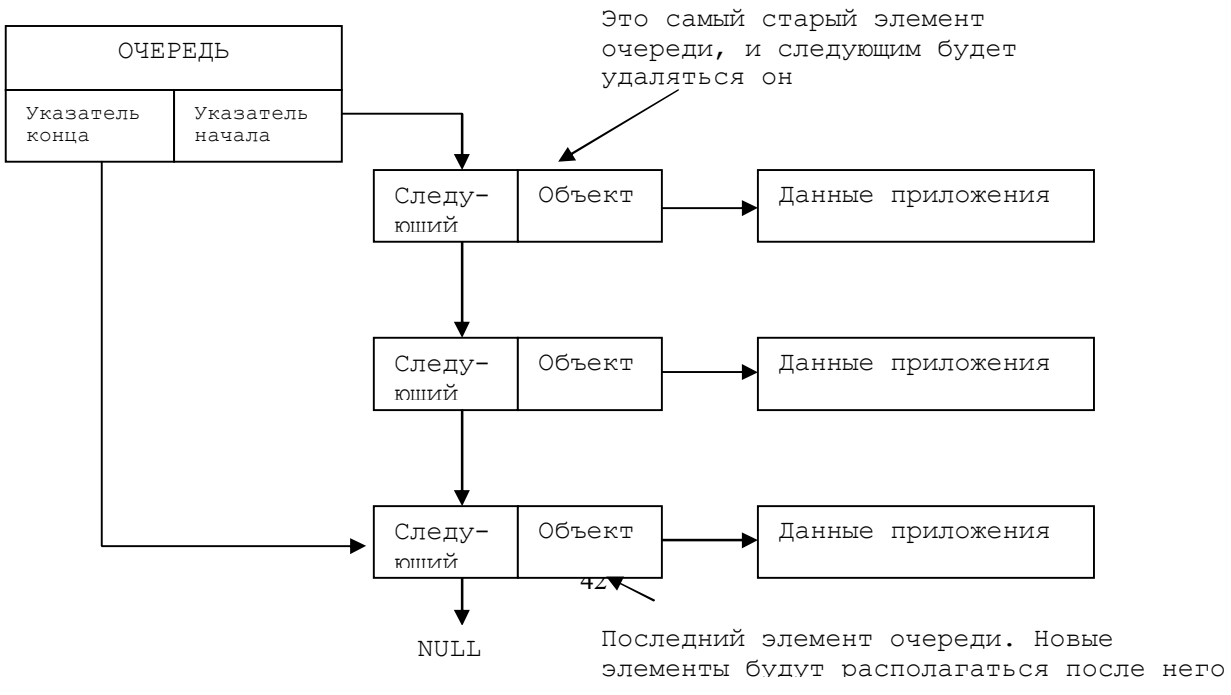


Рисунок 3 – Результат выполнения программы Stack

Часть 2. Очереди

Чергою називається впорядкований набір елементів, які можуть віддалятися з одного її кінця (що називається початком черги) і поміщатися в інший кінець цього набору (що називається кінцем черги).

У основних рисах черги (малюнок 2) схожі на стеки, але підкоряються іншому набору правил, відомому як **FIFO** (**First In, First Out** - перший прийшов, перший пішов). При такому режимі роботи додавати елементи можна тільки в кінець черги і видаляти елементи тільки з початку черги. Ця черга функціонує так само як, звичайна черга в магазині або на пошті.В



основных чертах очереди (рисунок 2) похожи на стеки, но подчиняются другому набору правил, известному как **FIFO** (**First In, First Out** – первый пришел, первый ушел). При таком режиме работы добавлять элементы можно только в конец очереди и удалять элементы только из начала очереди. Эта очередь функционирует точно так же как, обычная очередь в магазине или на почте.

Рисунок 2 – Очередь

Як і для стека, максимальне число елементів в черзі не повинне лімітуватися використовуваним програмним забезпеченням: пам'ять повинна проситися і звільнятися динамічно у міру того, як в чергу додаються нові і з неї видаляються елементи, що знаходяться там.Как и для стека, максимальное число элементов в очереди не должно лимитироваться используемым программным обеспечением: память должна запрашиваться и освобождаться динамически по мере того, как в очередь добавляются новые и из нее удаляются находящиеся там элементы.

Базові операції, що виконуються над чергою, очевидні;

- insert — додати в чергу новий елемент;insert — добавить в очередь новый элемент;
- remove — улолить з черги перший елемент.remove — улолить из очереди первый элемент.

Заголовний файл до програм, обслуговуючими чергу, приведений, в лістингу 4.

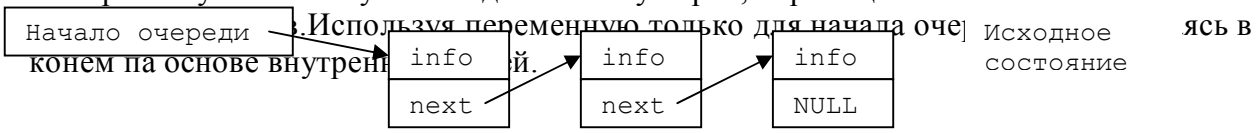
Листинг 4

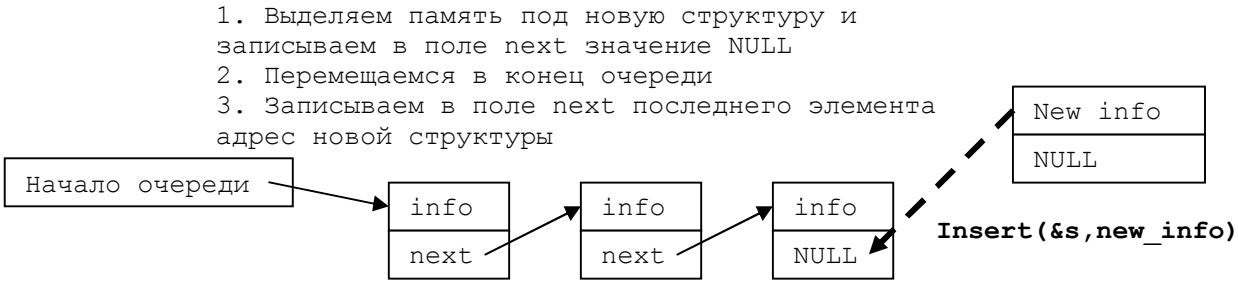
```
/*Интерфейс для работы с очередью*/
#define QUEUE struct queue

QUEUE{
 int   info;
 QUEUE *next;
};

extern void insertItem(QUEUE **ppQueue, int nItem);
extern int  removeItem(QUEUE **ppQueue, int *nError);
```

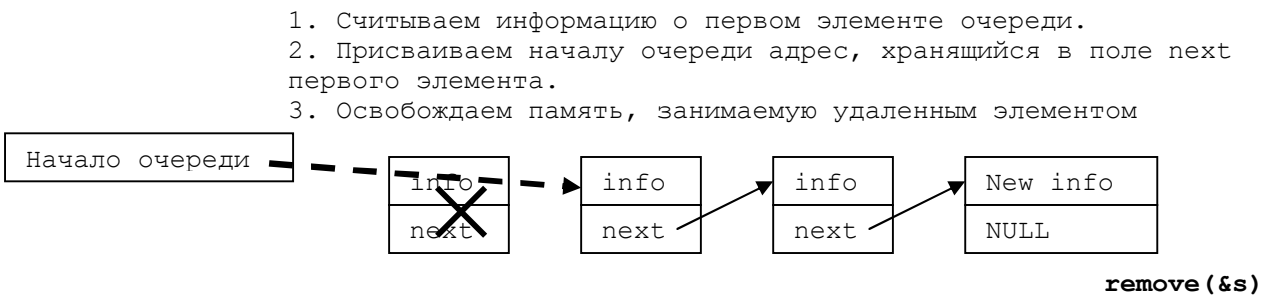
Представлені функції є аналогами функцій **push** і **pop**, вживаними ;лля роботи із стеком, і усе що говорилося там про передачу параметрів і індикації помилки, повною мірою відноситься і до представлених функцій.Представленные функции являются аналогами функций **push** и **pop**, применяемыми ;лля работы со стеком, и все что говорилось там о передаче параметров и индикации ошибки, в полной мере относится и к представленным функциям.

Роботу з чергою можна організувати днумя способами:

- використовуючи додаткові змінні дли почала і кінця черги.используя дополнительные переменные дли начала и конца очереди.
- Використовуючи змінну тільки для початку черги, переміщаючись в конем па основі .Используя переменную только для начала оче ясь в конем па основе внутрен й.







Принципы работы с очередью продемонстрированы на рисунке 3.
Рисунок 3 – Включение и исключение элементов из очереди

Реализация функций работы с очередью показана в листинге 5.

Листинг 5

```
/* Реализация функций работы с очередью */
#include <malloc.h>
#include "queue.h"

void insertItem(QUEUE **ppQueue, int nItem) {
 QUEUE *pNewItem;
 QUEUE *pCurItem = *ppQueue; // Текущий - начало очереди
 QUEUE *pPreItem = 0;
 while(pCurItem) {
  // Перемещаемся в конец очереди
  pPreItem = pCurItem;
  pCurItem = pCurItem->next;
 }

  // Запрашиваем память под структуру для элемента очереди
 pNewItem = (QUEUE *)malloc(sizeof(QUEUE));

  // Заполняем информационное поле
 pNewItem->info = nItem;
 if(pPreItem) {
    // Если очередь пуста, делаем новый элемент ее началом
  pNewItem->next = 0;
  pPreItem->next = pNewItem;
 }
 else {
    // В противном случае помещаем его в конец
  *ppQueue = pNewItem;
  (*ppQueue)->next = 0;
 }
}

int removeItem(QUEUE **ppQueue, int *nError) {
 QUEUE *pOldItem = *ppQueue;
 int   nOldInfo = 0;
 if(*ppQueue) {
    // Если очередь не пустая - извлекаем элемент
  nOldInfo = pOldItem->info;
  *ppQueue = (*ppQueue)->next;
    // и освобождаем память
  free(pOldItem);
  *nError = 0;
 }
 else
    // В противном случае ошибка
  *nError = 1;
 return nOldInfo;
```

```
}
```

Зверніть увагу, що код функції removeItem() повністю сопадает з кодом функції pop() для стека, тому детальніше розглянемо функцію insertItem.Обратите внимание, что код функции removeItem() полностью сопадает с кодом функции pop() для стека, поэтому более подробно рассмотрим функцию insertItem.

Спочатку створюється локальна змінна *pCurItem, що є покажчиком на структуру QUEUE, яка ініціалізувалася значенням початку черги, — *ppQueue. Потім описується локальна змінна *pPreItem того ж типу, яка ініціалізувалася значенням 0. І, нарешті, створиться локальна змінна для зберігання адреси нового елементу черги — *pNewItem

Далі організовується перегляд черги до досягнення її кінця :

```
while (pCurltem){
  pPreItem = pCurItem;
  pCurItem = pCurItem ->next;
}
```

Після чого покажчик pPreItem містить адресу останнього елементу Q'.JF.UF; у черзі, Далі, у фрагменті програмиПосле чего указатель pPreItem содержит адрес последнего элемента Q'.JF.UF; в очереди, Далее, в фрагменте программы

```
pNewIten: = (QUEUE *) malloc (sizeof (QUEUE));
pNewItem ->info = nItem;
if(pPreItem){
  pNewItem ->next = 0;
  pPreItem ->next = pNewItem;
) else {
  *ppQueue = pNewItein;
  (*ppQueue) ->next = 0;
}
```

проситься пам'ять під новий елемент черги **pNewItem**, полю **info** цієї структури привласнюється значення **nItem**. Якщо черга не порожня, то полю next нової структури привласнюється значення 0, яке показує, що це тепер останній елемент в черзі, а полю next елементу, що був останнім, привласнюється значення адреси нової структури **pNewItem**. Якщо ж черга порожня, то покажчик на початок черги набуває значення адреси нового елементу.запрашивается память под новый элемент очереди **pNewItem**, полю **info** этой структуры присваивается значение **nItem**. Если очередь не пуста, то полю next новой структуры присваивается значение 0, которое показывает, что это теперь последний элемент в очереди, а полю next бывшего последнею элемента присваивается значение адреса новой структуры **pNewItem**. Если же очередь пуста, то указатель на начало очереди принимает значение адреса нового элемента.

Як приклад, що ілюструє роботу з чергою, розглядається програма калькулятора (лістинг 6) з чотирма арифметичними діями і можливістю зміни пріоритету виконання операції за допомогою дужок трьох типів : (), [] і { }.В качестве примера, иллюстрирующего работу с очередью, рассматривается программа калькулятора (листинг 6) с четырьмя арифметическими действиями и возможностью изменения приоритета выполнения операции при помощи скобок трех типов: (), [] и { }.

Лістинг 6

```
/*  Тестова програма, що використовує змінні типу стік і черга.
Дозволяє вводити довільні числові вирази з використанням трьох типів дужок : (), [] і {}.
Після закінчення введення на екран виводиться результат обчислень  */
#include <stdio.h>include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
```

```
#include "stack.h"
#include "queue.h"

#define MAXSIZE 128
#define LF 10

STACK *s1, *s3;
QUEUE *s2;

int getExpression(char *str);
int parseExpression(char *str);
int priority(int nOper1, int nOper2);
int evaluate();

int main()
{
 char strExpression[MAXSIZE + 1];

  // Вводимо вираження
 printf("\nEnter expression: \n");
 getExpression(strExpression);
 printf("\nYou enter expression: %s\n", strExpression);

  // Розбираємо його
 parseExpression(strExpression);parseExpression(strExpression);

  // Обчислюємо результат і виводимо його на екран
 printf("\nResult = %d\n", evaluate());
 getchar();getchar();
 return 1;return 1;
}

 // Функція, що просить вираження у користувача
int getExpression(char *str) {
 int ch, nCount = 0;
 while((ch = getchar()) != LF) {
    // Поки не натиснута клавіша ENTER, вводимо символи
  if (isdigit(ch) || ch == '(' || ch == ')' ||
      ch == '{' || ch == '}' || ch == '[' || ch == ']' ||
      ch == '*' || ch == '/' || ch == '+' || ch == '-')
  {
   // Допустимими символами є: цифри (, ), [, ], {, }
   // і знаки арифметичних операцій
   if(nCount >= MAXSIZE) {if(nCount >= MAXSIZE)  {
    // Рядок не може містити більше MAXSIZE символівСтрока не может содержать
больше MAXSIZE символов
     *str = '';str = '\0';
     return 0;return 0;
   }
    // Додаємо символ до рядка
   *str++ = ch;str++ = ch;
   nCount++;nCount++;
```

```
   }
  }
   // Вставляємо завершуючий нуль
  *str = '';str = '\0';
  return 1;return 1;
}

 // Функція здійснює розбір вираження і перетворення
 // з інфіксної форми запису в постфіксну
int parseExpression(char *strExpression) {
  int  ch;
  int   nError, nNumeral;
  char *str = strExpression, aTmp[32];
  char *strTmp = aTmp;
  while((ch = *str) != 0) {
    // Поки не досягнутий кінець рядка, перевіряємо символи
   switch(ch){switch(ch) {
      // Якщо це відкриваюча дужка - поміщаємо її в стек
    case '(':
    case '{':
     // Якщо це закриваюча дужка - читаємо значення з вершини стека
    case '[': push(&s1, ch);
            str++;
            break;
    case ') ':
    case '}':
    case ']': nNumeral = peek(&s1, &nError);
            if(nError) {
              // Стік порожній
              printf("No openning brackets\n", ch);
              return 0;return 0;
            }
               // Поки у вершині стека дужка
            while(!nError && priority(nNumeral, ch)) {
                // Витягаємо елемент із стека і поміщаємо його в чергу
              insertItem(&s2, pop(&s1, &nError));
                // Читаємо наступний елемент
              nNumeral = peek(&s1, &nError);nNumeral = peek(&s1, &nError);
            }
               // Видаляємо дужку із стека
            pop(&s1, &nError);
            str++;
            break;
     // Якщо це знак операції - читаємо значення з вершини стека
    case '+':
    case '-':
    case '/':
    case '*': nNumeral = peek(&s1, &nError);
            while(!nError && priority(nNumeral, ch)) {
                // Витягаємо елемент із стека і поміщаємо його в чергу
              insertItem(&s2, pop(&s1, &nError));
                // Поміщаємо знак операції в стек
```

```
            nNumeral = peek(&s1, &nError);nNumeral = peek(&s1, &nError);
          }
            // Видаляємо елемент із стека
          push(&s1, ch);push(&s1, ch);
          str++;
          break;
      default:  while(isdigit(ch = *str)) {
              // Переписуємо цифри в тимчасовий рядок
          *(strTmp++) = ch;strTmp++) = ch;
          str++;str++;
          }
          *strTmp = '';strTmp = '\0';
            // Перетворимо рядок в число
          nNumeral = atoi(aTmp);
          strTmp = aTmp;
            // Вставляємо число в чергу
          insertItem(&s2, nNumeral);insertItem(&s2, nNumeral);
          break;
     }
    }
    nNumeral = peek(&s1, &nError);
    while(!nError) {while(!nError)  {
      // Витягаємо елементи, що залишилися, із стека і поміщаємо їх в чергу
     insertItem(&s2, pop(&s1, &nError));
     nNumeral = peek(&s1, &nError);
    }
    return 1;
  }
   // Функція порівнює пріоритети двох операцій
  int priority(int nOper1, int nOper2) {int priority(int nOper1, int nOper2)  {
   if((nOper1 == '(' || nOper1 == '{' || nOper1 == '[') ||
     (nOper2 == '(' || nOper2 == '{')|| nOper2 == '[')
     return 0;
   if((nOper2 == ')' || nOper2 == '}' || nOper2 == ']'))
     return 1;
   if((nOper1 == '+'|| nOper1 == '-')&&(nOper2 == '*'|| nOper2 == '/'))
     return 0;return 0;
    return 1;return 1;
  }
   // Функція витягає вираження з черги і обчислює його
  int evaluate() {
   int nValue, nResult, nTmp, nError;
   nValue = removeItem(&s2, &nError);
   while(!nError) {
     // Якщо лічений знак операції
    if(nValue!='+' &&nValue!='-' && nValue!='*' && nValue!='/')if(nValue!='+' &&nValue!='-'
&& nValue!='*' && nValue!='/')
       // поміщаємо її в стек
      push(&s1, nValue);push(&s1, nValue);
     else {else  {
       // Інакше виконуємо операцію
       // над двома верхніми елементами стека
```

```
        nTmp = pop(&s1, &nError);nTmp = pop(&s1, &nError);
        switch(nValue) {
          case '+': nResult = pop(&s1, &nError) + nTmp;
                break;
          case '-': nResult = pop(&s1, &nError) - nTmp;
                break;
          case '*': nResult = pop(&s1, &nError) * nTmp;
                break;
          case '/': nResult = pop(&s1, &nError) / nTmp;
                break;break;
        }
          // Поміщаємо результат назад в стек
         push(&s1, nResult);push(&s1, nResult);
       }
       // Витягаємо з черги наступний елемент
       nValue = removeItem(&s2, &nError);
     }
     return nResult;
   }
```

Приведені коментарі роблять зайвими які-небудь пояснення, тому відмітимо тільки два моменти:Приведенные комментарии делают излишними какие-либо пояснения, поэтому отметим только два момента:

На вході користувач вводить арифметичне вираження (можна з дужками) в інфіксній формі, яке потім перетвориться в постфіксну форму і після цього обчислюється.На входе пользователь вводит арифметическое выражение (можно со скобками) в инфиксной форме, которое затем преобразуется в постфиксную форму и после этого вычисляется.

Приведена програма має серйозний недолік — вона не перевіряє, чи є вхідний рядок коректним вираженням.Приведенная программа имеет серьезный недостаток — она не проверяет, является ли входная строка корректным выражением.

Результат работы программы представлен на рисунке 3.

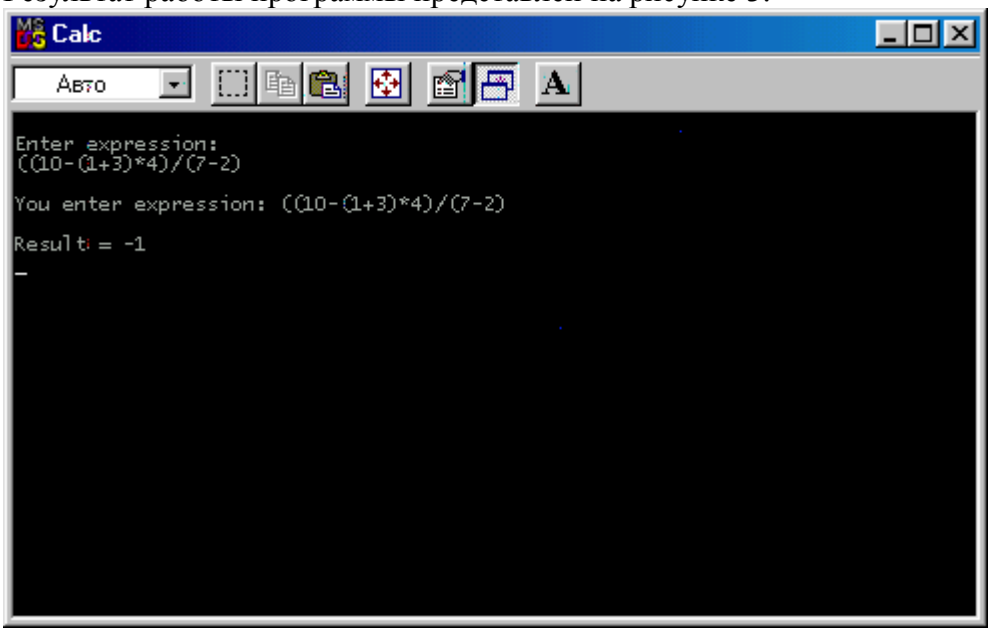


Рисунок 3 – Результат работы программы Calc

Окрім лінійної черги, робота якої розглядалася вище, часто використовується так звана циклічна черга, в якій останній елемент вказує на її початок. Така організація черги зручна, якщо вона реалізується на основі масиву.

Частина 3. Списки

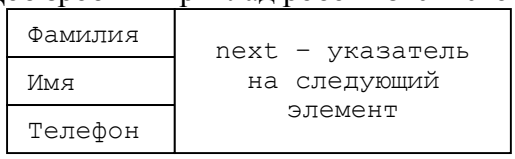
Розглянуті структури - стік і черга, є спеціальними різновидами загальнішої структури даних - узагальненого пов'язаного списку. Існує безліч способів його реалізації. Розглянемо лінійні (однонапрямлені) списки.

Заголовний файл для роботи з такими списками приведений в лістингу 7.

Лістинг 7

```
/*Інтерфейс для роботи зі списком*/
#define LIST struct list

LIST{
 char aLastName[16];
 char aFirstName[16];
 char aTelephoneNumber[16];
 LIST *pNext;
};
extern void insert(LIST **ppList, LIST *pItem);
extern void destroy(LIST **pList);
extern void display(LIST  *pList);
extern int removeItem(LIST **ppList, LIST *pItem);
```

Щоб зробити приклад роботи зі списком реалістичнішим і продемонструвати особливості роботи із структурами, кожен елемент списку (малюнок 4) є символьним рядком.Щоб зробити приклад роботи зі списком реалістичнішим і продемонструвати



особливості роботи із структурами, кожен елемент списку (малюнок 4) є символьним рядком.Щоб зробити приклад роботи зі списком реалістичнішим і продемонструвати особливості роботи із структурами, кожен елемент списку (малюнок 4) є символьним рядком.Чтобы сделать пример работы со списком более реалистичным и продемонстрировать особенности работы со структурами, каждый элемент списка (рисунок 4) представляет собой символьную строку.

Малюнок 4 - Приклад структури, що визначає елемент списку

Для роботи зі списком передбачені наступні функції:

- **insert** — додати новий елемент в список, зберігаючи встановлений порядок дотримання.**insert** — добавить новый элемент в список, сохраняя установленный порядок следования.
- **destroy** — зруйнувати список.**destroy** — разрушить список.
- **display** — вивести усі елементи списку.**display** — вывести все элементы списка.
- **remove** — видалити елемент списку.**remove** — удалить элемент списка.

Реалізація цих функцій представлена в лістингу 8.

Лістинг 8

```
/* Реалізація функцій роботи зі списком */
#include <string.h>
#include <malloc.h>
#include <stdio.h>
```

```
#include "list.h"

 // Статична функція створення елементу списку
static LIST* create(LIST *pItem)static LIST* create(LIST *pItem)
{
 LIST *pNewItem = (LIST *) malloc(sizeof(LIST));
 *pNewItem = *pItem;
 return pNewItem;
}

void insert(LIST **ppList, LIST *pItem){
 char aKey[16];
 LIST *pNewItem;
 LIST *pCurItem = *ppList;
 LIST *pPreItem = 0;LIST *pPreItem = 0;
  // В якості ключового поля в списку використовується прізвище.
 strcpy(aKey, pItem ->aLastName);strcpy(aKey, pItem->aLastName);
  // Шукаємо елемент, співпадаючий з ключовим
 while(pCurItem != 0 &&  strcmp(pCurItem ->aLastName, aKey)< 0) {
  pPreItem = pCurItem;
  pCurItem = pCurItem ->pNext;
 }
  // Тут pCurItem або 0, або дорівнює покажчику на знайдений елементЗдесь pCurItem либо
0, либо равен указателю на найденный элемент
  // Створюємо новий елемент і заповнюємо його поля
 pNewItem = create(pItem);
 pNewItem ->pNext = pCurItem;
  // Вставляємо новий елемент в список
 if(pPreItem == 0)if(pPreItem == 0)
    // або в початок
  *ppList = pNewItem;ppList = pNewItem;
 else
    // або після ключового
  pPreItem ->pNext = pNewItem;pPreItem->pNext = pNewItem;
}

int removeItem(LIST **ppList, LIST *pItem) {
 LIST *pCurItem = *ppList;
 LIST *pPreItem = 0;
  // Шукаємо заданий елемент
 while(pCurItem != 0 &&while(pCurItem != 0 &&
      strcmp(pCurItem ->aLastName, pItem ->aLastName)!= 0)  {
  pPreItem = pCurItem;
  pCurItem = pCurItem ->pNext;
 }
  // Якщо такого елементу немає, то і видаляти нічого
 if(pCurItem != 0)
  return 0;
 if(pPreItem == 0)
    // Видаляємо перший елемент списку
  *ppList = pCurItem ->pNext;ppList = pCurItem->pNext;
 else
```

```
      // Видаляємо елемент списку, починаючи з другого
     pPreItem ->pNext = pCurItem ->pNext;pPreItem->pNext = pCurItem->pNext;
    free(pCurItem);  // обов'язково звільняємо пам'ятьfree(pCurItem); // обязательно
освобождаем память
    return 1;
  }

  void destroy(LIST **ppList) {
   LIST *pCurItem = *ppList;
   LIST *pPreItem = 0;
    // Прохідний увесь список
   while(pCurItem != 0) {while(pCurItem != 0)  {
    pPreItem = pCurItem;
    pCurItem = pCurItem ->pNext;
     // і видаляємо кожен поточний елемент
    free(pPreItem);free(pPreItem);
   }
    // Нульовий покажчик на початок списку говорить, що список порожній
   *ppList = 0;
  }

  void display(LIST  *pList) {
   LIST *pCurItem = pList;LIST *pCurItem = pList;
    // Послідовно прохідний увесь список
   while(pCurItem) {while(pCurItem)  {
     // і виводимо інформацію про кожен елемент
    printf("\n%s, %s", pCurItem ->aLastName, pCurItem ->aFirstName);
    printf("    \t%s", pCurItem ->aTelephoneNumber);
    pCurItem = pCurItem ->pNext;
   }
   printf("\n\n");
  }
```

У лістингу 8 є статична функція **create**, яка призначена для того, щоб відводити пам'ять під новий елемент списку і передавати дані, поміщені але адресі **pItem** (покажчик на структуру), в новий елемент. У функції використовується оператор привласнення :В листинге 8 имеется статическая функция **create**, которая предназначена для того, чтобы отводить память под новый элемент списка и передавать данные, помещенные но адресу **pItem** (указатель на структуру), в новый элемент. В функции используется оператор присваивания:

*pNewItem = *pItem;pNewItem = *pItem;

для пересилки усієї інформації, розташованої але адресі **pItem**, в область пам'яті за адресою **pNewItem**.для пересылки всей информации, расположенной но адресу **pItem**, в область памяти по адресу **pNewItem**.

Вона введена виключно для демонстрації роботи із статичними функціями: її можна викликати тільки з перелічених вище за функмий роботу зі списком: безпосереднім виклик з іншого модуля заборонений. Це один з прийомів обмеження доступу -~ новий елемент списку можна створювати тільки за допомогою інтерфейсних функцій.Она введена исключительно для демонстрации работы со статическими функциями: ее можно вызвать только из перечисленных выше функмий работы со списком: непосредственным вызов из другого модуля запрещен. Это один из приемов ограничения доступа -~ новый элемент списка можно создавать только при помощи интерфейсных функций.

Зупинимося також на деяких аспектах функцій **insert** і **remove**, оскільки реалізація функцій **display** і **destroy** досить тривіальна і вже, в тому або іншому вигляді, розглядалася.Остановимся также на некоторых аспектах функций **insert** и **remove**, поскольку реализация функций **display** и **destroy** достаточно тривиальна и уже, в том или ином виде, рассматривалась.

Робота функції **insert** проілюстрована на малюнку 5.Работа функции **insert** проиллюстрирована на рисунке 5.

Так, щоб вставити елемент в список, необхідно спочатку знайти елемент зі значенням ключового поля, співпадаючим із заданим. Це здійснюється шляхом проходу *за списком*Таким образом, чтобы вставить элемент в список, необходимо сначала найти элемент со значением ключевого поля, совпадающим с заданным. Это осуществляется путем *прохода по списку,*

while (pCurItem != Про && strcmp (pCurItem > aLastName, aKey)< 0){while (pCurItem != О && strcmp (pCurItem > aLastName, aKey) < 0) {

```
  pPreItem = pCurItem;
  pCurItem = pCurItem ->pNext;
}
```

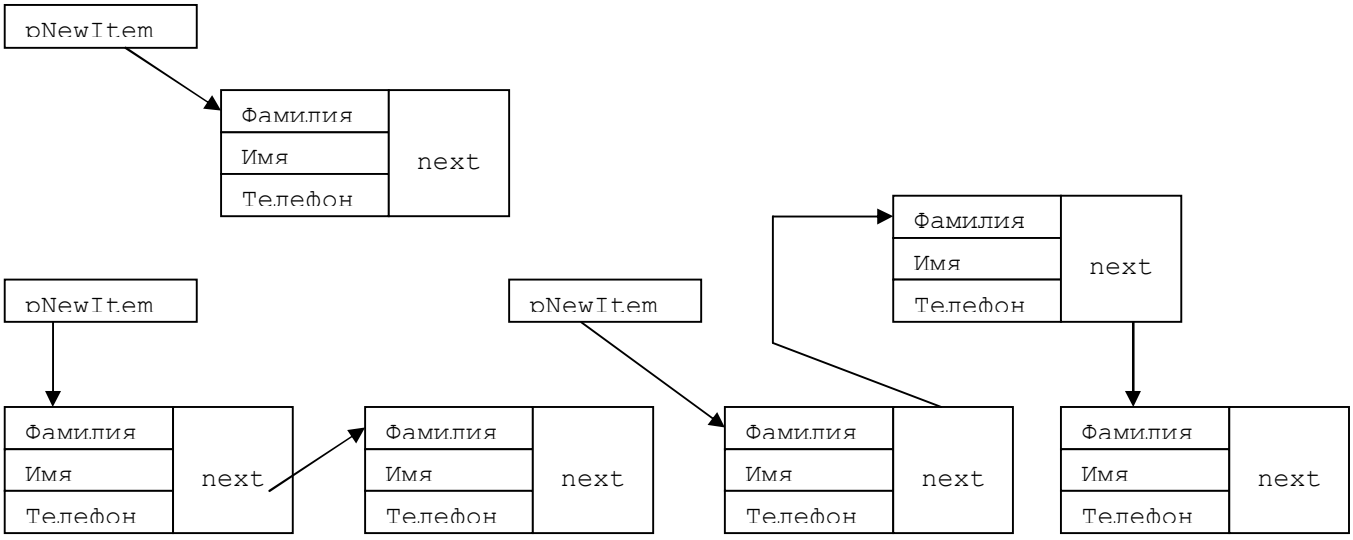


Рисунок 5 – Вставка элемента в список

Після цього вставляємо новий елемент (не має значення, коли він був створений, до пошуку або після) :После этого вставляем новый элемент (не имеет значения, когда он был создан, до поиска или после):

```
if (pPreItem == 0)
  *ppList = pNewItem;
else
  pPreItem ->pNext = pNewItem;
```

Зверніть увагу, що новий елемент вставляється в список *після* ключового. Тому важливо, порожній це список або ні, для усіх інших випадків вставка здійснюється однаково.Обратите внимание, что новый элемент вставляется в список *после* ключевого. Поэтому важно, пустой это список или нет, для всех остальных случаев вставка осуществляется одинаково.

Операція видалення елементу зі списку також досить проста (малюнок 6).

Передусім необхідно організувати прохід за списком, щоб знайти потрібний елемент, Якщо такого елементу немає, то, нічого не видаляючи, просто виходимо з функції:Прежде всего необходимо организовать проход по списку, чтобы найти нужный элемент, Если такого элемента нет, то, ничего не удаляя, просто выходим из функции:

```
if(pCurItem != 0)
```

```
return 0;
```

Рисунок 6 – Удаление элемента из списка

Рисунок 7 - Результат работы программы List

Листинг 9

```
/* Простая тестовая программа, демонстрирующая использование функций работы со списком */
#include <string.h>
#include <stdio.h>
#include <malloc.h>
#include "list.h"

LIST *pMyList;

void main() {
 LIST *pItem = (LIST *)malloc(sizeof(LIST));
  // Подготавливаем данные
 strcpy(pItem->aLastName, "Tihomirov");
 strcpy(pItem->aFirstName, "Yury");
 strcpy(pItem->aTelephoneNumber, "558-01-02");
 insert(&pMyList, pItem);              // вставляем элемент

  // Подготавливаем данные
 strcpy(pItem->aLastName, "Meshkov");
 strcpy(pItem->aFirstName, "Andrew");
 strcpy(pItem->aTelephoneNumber, "534-11-68");
 insert(&pMyList, pItem);              // вставляем элемент

  // Подготавливаем данные
 strcpy(pItem->aLastName, "Shvedov");
 strcpy(pItem->aFirstName, "Dmitry");
 strcpy(pItem->aTelephoneNumber, "165-65-56");
 insert(&pMyList, pItem);              // вставляем элемент

  // Подготавливаем данные
 strcpy(pItem->aLastName, "Bortnovsky");
 strcpy(pItem->aFirstName, "Sergew");
 strcpy(pItem->aTelephoneNumber, "168-05-56");
 insert(&pMyList, pItem);              // вставляем элемент

  // Распечатываем содержимое списка
 display(pMyList);
  // Удаляем элемент
 removeItem(&pMyList, pItem);
  // Распечатываем содержимое списка
 display(pMyList);
  // Разрушаем весь список
 destroy(&pMyList);
 getchar();
```

}

Окрім розглянутого лінійного списку існують і інші:

Циклічний* список** — поле ruxt останнього елементу містить покажчик назад на перший елемент. Такий список не має першого і останнього елементів. Проте в деяких випадках зручно використати зовнішній покажчик на останній елемент, що автоматично робить першим елемент, що йде за ним. Альтернативний варіант припускає використання покажчика на перший елемент.Циклический* список** — поле ruxt последнего элемента содержит указатель назад на первый элемент. Такой список не имеет первого и последнего элементов. Однако в некоторых случаях удобно использовать внешний указатель на последний элемент, что автоматически делает первым следующий за ним элемент. Альтернативный вариант предполагает использование указателя на первый элемент.

Двонаправлені списки — кожен елемент такого списку містить дна покажчика : один вказує на попередній елемент, а інший — на подальший. Такі списки можуть бути лінійними і циклічними.***Двунаправленные списки*** — каждый элемент такого списка содержит дна указателя: один указывает на предшествующий элемент, а другой — на последующий. Такие списки могут быть линейными и циклическими.

Мультисписки — структура даних, що складається з елементів, які можуть належати декільком спискам, не повторюючись в декількох елементах для кожного списку. Такі елементи мають покажчики на кожен список, якому вони належать. Списки в мультисписку можуть бути лінійними і циклічними, однонапрямленими і двонаправленими.***Мультисписки*** — структура данных, состоящая из элементов, которые могут принадлежать нескольким спискам, не повторяясь в нескольких элементах для каждого списка. Такие элементы имеют указатели на каждый список, которому они принадлежат. Списки в мультисписке могут быть линейными и циклическими, однонаправленными и двунаправленными.

Частина 4. Бінарні дерева

Бінарне дерево — ця кінцева безліч елементів, яка або порожньо, або міст! один елемент, що називається *коренем* дерева, а інші елементи великої кількості діляться на дві підмножини, що не перетинаються, кожне з яких саме є бінарним деревом, Ці підмножини називаються *лівим* і *правим піддеревами* початкового дерева. Кожен елемент бінарного дерева називається *розумом* дерева. Загальноприйнятий спосіб зображення бінарного дерева представлений на малюнку 8.*Бинарное дерево* — это конечное множество элементов, которое либо пусто, либо содержи! один элемент, называемый *корнем* дерева, а остальные элементы множества делятся на два непересекающихся подмножества, каждое из которых само является бинарным деревом, Эти подмножества называются *левым* и *правым поддеревьями* исходного дерева. Каждый элемент бинарного дерева называется *умом* дерева. Общепринятый способ изображения бинарного дерева представлен на рисунке 8.

Якщо A — корінь бінарного дерева, а B — корінь його лівого або правого поддерева, то говорять, що A є *батьком* B? а B — *лівий* або *правий син,* Два вузли є *братами,* якщо вони сини одного і того ж батька. Вузол, що не має синів (вузли D, G, H і I на малюнку 8), називається *листом,* Якщо кожен вузол бінарного дерева, що не є листом, має непорожні праве і ліве піддерева, то дерево називається *строго бінарним деревом.*Если A — корень бинарного дерева, а B — корень его левого или правого поддерева, то говорят, что A является *отцом* B? а B — *левый* или *правый сын,* Два узла являются *братьями,* если они сыновья одного и того же отца. Узел, не имеющий сыновей (узлы D, G, H и I на рисунке 8), называется *листом,* Если каждый узел бинарного дерева, не являющийся листом, имеет непустые правое и левое поддеревья, то дерево называется *строго бинарным деревом.*

Графічне уявлення (малюнок 8) наглядно і їм зручно користуватися при роботі з бінарними деревами. Проте слід пам'ятати, що пам'ять лінійна і з цієї точки зору можна вважати, що бінарне дерево є різновидом зв'язного списку. Тому основні операції над ними

багато в чому ті ж самі, що і лдя списків: елементи дерева можна додавати, видаляти, а також здійснювати до них доступ. Інтерфейс для програм роботи з бінарними деревами приведений в лістингу 10. Інформаційні поля можуть бути довільними. У лабораторній роботі, щоб не захаращувати функції додатковою обробкою "складних" полів, застосовуються два цілі поля.Графическое представление (рисунок 8) наглядно и им удобно пользоваться при работе с бинарными деревьями. Однако следует помнить, что память линейна и с этой точки зрения можно считать, что бинарное дерево представляет собой разновидность связного списка. Поэтому основные операции над ними во многом те же самые, что и лдя списков: элементы дерева можно добавлять, удалять, а также осуществлять к ним доступ. Интерфейс для программ работы с бинарными деревьями приведен в листинге 10. Информационные поля могут быть произвольными. В лабораторной работе, чтобы не загромождать функции дополнительной обработкой "сложных" полей, применяются два целых поля.

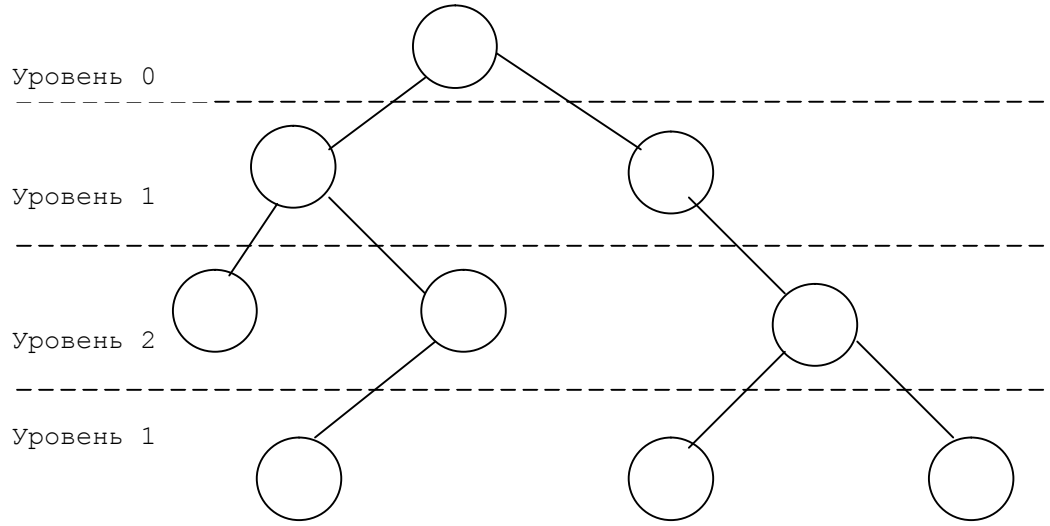


Рисунок 8 – Бинарное дерево

Листинг 10

```
/*Интерфейс для работы с деревом
Модуль Tree.h*/

#define TREE struct tree

TREE {
 int  iField;
 int  iCount;
 TREE *pLeft, *pRight;
};

extern void insert(TREE **ppRoot, TREE *pItem);
extern int remove(TREE **ppRoot, TREE *pItem);
extern void destroy(TREE *pRoot);
extern void display(TREE *pRoot);
```

Структура для роботи з бінарними деревами приведена на мал. 5.17. Зверніть увагу на її подібність із структури для роботи з двонаправленим списком (малюнок 9).Структура для работы с бинарными деревьями приведена на рис. 5.17. Обратите внимание на ее сходство со

структурой для работы с двунаправленным списком (рисунок 9).

Малюнок 9 - Структура вузла бінарного дерева

При роботі з деревами однієї з основних операцій являється *проходження дерева* — обхід усього дерева, при якому кожен вузол відвідується один раз, Існують три способи, що відрізняються порядком відвідування кореня і проходження його лівого і правого піддерев :При работе с деревьями одной из основных операций является *прохождение дерева* — обход всего дерева, при котором каждый узел посещается один раз, Существуют три способа, отличающиеся порядком посещения корня и прохождения его левого и правого поддеревьев:

Прямий порядок (низхідний)

- потрапити в корінь;
- пройти в прямому порядку ліве поддерево;
- пройти в прямому порядку праве.

Симетричний порядок (послідовний)

- пройти в симетричному порядку ліве поддерево;
- потрапити в корінь;
- пройти в симетричному порядку праве поддерево.

Зворотний порядок (висхідний)

- пройти в зворотному порядку ліве поддерево;
- пройти в зворотному порядку праве поддерево;
- потрапити в корінь.

Прямий порядок проходження вузлів для дерева на малюнку 9 - ABDEGCFHI, симетричний, - DBGEAHFIC, зворотний, - DGEBHIFCA.Прямой порядок прохождения узлов для дерева на рисунке 9 – ABDEGCFHI, симметричный – DBGEAHFIC, обратный – DGEBHIFCA.

Функції для роботи з бінарними деревами ті ж, що і для списку. Реалізація цих функцій приведена в лістингу 11.Функции для работы с бинарными деревьями те же, что и для списка. Реализация этих функций приведена в листинге 11.

Лістинг 11

```
/* Реалізація функцій роботи з деревом
   Модуль Tree.c */
#include <string.h>
#include <malloc.h>
#include <stdio.h>
#include "tree.h"

static TREE* create(TREE *pItem) {
 TREE *pNewItem = (TREE *) malloc(sizeof(TREE));
 *pNewItem = *pItem;
 return pNewItem;
}

void insert(TREE **ppRoot, TREE *pItem) {
  // В якості ключового поля в списку використовується прізвище
 TREE *pParent = 0, *pCurItem = *ppRoot;
 TREE *pNewItem;
 int   nFound = 0;int   nFound = 0;
  // Проходимо дерево, поки не досягнутий лист або не знайдений вузол
 while(pCurItem &&!nFound) {while(pCurItem && !nFound)  {
    // Поки не досягнутий лист дерева або не знайдений вузол
  if(pItem ->iField == pCurItem ->iField) {
```

```
      // Вузол знайдений - встановлюємо прапор завершення пошуку
     nFound = 1;nFound = 1;
      // і збільшуємо значення лічильника числа входжень
     pCurItem ->iCount++;pCurItem->iCount++;
    }
    else {else {
      // Запам'ятовуємо поточний вузол
     pParent = pCurItem;
     if(pItem ->iField < pCurItem ->iField)
        // Якщо новий елемент "менший", ніж що зберігається у вузлі
        // йдемо по лівій гілці
      pCurItem = pCurItem ->pLeft;
     else
        // інакше - по правій
      pCurItem = pCurItem ->pRight;
    }
   }

   // Якщо елементу немає в дереві
  if(!nFound){if(!nFound) {
   // Такого елементу в дереві немає - створюємо новий вузол
   pNewItem = create(pItem);
   pNewItem ->pLeft = pNewItem ->pRight = 0;
   if(pParent == 0) {if(pParent == 0)  {
      // Перший вузол дерева - створюємо його і робимо коренем
    *ppRoot = create(pItem);
    (*ppRoot) ->pLeft = (*ppRoot) ->pRight = 0;
   }
   else {else {
      // Якщо новий елемент менший, ніж що зберігається в поточному вузлі
     if(pItem ->iField < pParent ->iField)
        // вставляємо його в ліву гілку
      pParent ->pLeft = pNewItem;pParent->pLeft = pNewItem;
     else
        // інакше - в праву
      pParent ->pRight = pNewItem;
   }
  }
 }
 int remove(TREE **ppRoot, TREE *pItem) {
  TREE *pPreItem = 0,
      *pPresent = *ppRoot;
  TREE *pReplace
      *pParentpParent,
      *pTmp;pTmp;
  int   nFound = 0;int   nFound = 0;

   // Шукаємо вузол із заданим ключовим полем
  while(pPresent &&!nFound) {
   if(pItem ->iField == pPresent ->iField)
     nFound = 1;
   else {
```

```
    pPreItem = pPresent;
    if(pItem ->iField < pPresent ->iField)
      pPresent = pPresent ->pLeft;
    else
      pPresent = pPresent ->pRight;
   }
}
if(nFound){
  if(pPresent ->pLeft == 0)
      // Якщо немає лівого поддерева, то елемент, що видаляється
      // замінюємо на праве поддерево
    pReplace = pPresent ->pRight;pReplace = pPresent->pRight;
  else

      // Якщо немає правого поддерева, то елемент, що видаляється
      // замінюємо на ліве поддерево
    if(pPresent ->pRight == 0)if(pPresent->pRight == 0)
      pReplace = pPresent ->pLeft;
    else {
      pParent = pPresent;
      pReplace = pPresent ->pRight;
      pTmp = pPresent ->pLeft;

        // Шукаємо найлівіший лист
      while(pTmp != 0) {while(pTmp != 0)  {
        pParent = pReplace;
        pReplace = pTmp;
        pTmp = pReplace ->pLeft;
      }
        // Якщо це корінь
      if(pParent != pPresent){if(pParent != pPresent) {
          // Замінюємо на знайдений лист
        pParent ->pLeft = pReplace ->pRight;
        pReplace ->pRight = pPresent ->pRight;
      }
      pReplace ->pRight = pPresent ->pRight;
    }
    if(pPreItem == 0)
        // Якщо віддаляється корінь
        // те вибраний раніше син стає коренем дерева
      *ppRoot = pReplace;ppRoot = pReplace;
    else
        // Інакше замінюємо значення
        // відповідного покажчика
      if(pPresent == pPreItem ->pLeft)if(pPresent == pPreItem->pLeft)
        pPreItem ->pLeft = pReplace;
      else
        pPreItem ->pRight = pReplace;
   // Не забуваємо звільнити пам'ять, займану видаленим вузлом
  free(pPresent);
}
return 1;
```

```
}

void destroy(TREE *pRoot) {
 if(pRoot) {if(pRoot)  {
    // Рекурсивно проходимо ліве поддерево
  destroy(pRoot ->pLeft);destroy(pRoot->pLeft);
    // Рекурсивно проходимо праве поддерево
  destroy(pRoot ->pRight);
    // Видаляємо вузол
  free(pRoot);
 }
 pRoot = 0;
}
void display(TREE *pRoot) {
 if(pRoot) {if(pRoot)  {
    // Рекурсивно проходимо ліве поддерево
  display(pRoot ->pLeft);display(pRoot->pLeft);

    // Виводимо інформацію про вузол
  printf("\n%d", pRoot ->iField);

    // Рекурсивно проходимо праве поддерево
  display(pRoot ->pRight);display(pRoot->pRight);
 }
}
```

Функція destroy для проходження дерева використовує висхідний спосіб про= ходу дерева, "сходячи" спочатку вниз по лівому поддереву, а функція display — симетричний (послідовний) обхід бінарного дерева. У обох випадках використовується рекурсивний виклик відповідної функції.Функция destroy для прохождения дерева использует восходящий способ об= хода дерева, "спускаясь" сначала вниз по левому поддереву, а функция display — симметричный (последовательный) обход бинарного дерева. В обоих случаях используется рекурсивный вызов соответствующей функции.

Розглянемо тепер детальніше функцію **insert**, яка додає в дерево новий вузол, опускаючи пояснення з приводу визначення локальних змінних. Передусім організовуємо пошук вузла, що додається, — можливо, він вже є присутнім в дереві:Рассмотрим теперь более подробно функцию **insert**, которая добавляет в дерево новый узел, опуская пояснения по поводу определения локальных переменных. Прежде всего организуем поиск добавляемого узла — может быть, он уже присутствует в дереве:

```
*pCurItem = *ppRoot;  // Задаємо покажчик початку пошукурCurItem = *ppRoot;  // Задаем указатель начала поиска
nFound = 0;          // і прапор завершення пошукунFound = 0;          // и флаг завершения поиска
while(pCurItem &&!nFound) {while(pCurItem && !nFound)  {
  // Поки не досягнутий лист дерева або не знайдений вузол
 if(pItem ->iField == pCurItem ->iField) {
    // Вузол знайдений - встановлюємо прапор завершення пошуку
  nFound = 1;nFound = 1;
    // і збільшуємо значення лічильника числа входжень
  pCurItem ->iCount++;pCurItem->iCount++;
 }
 else {else {
  // Запам'ятовуємо поточний вузол
   pParent = pCurItem;
```

```
      if(pItem ->iField < pCurItem ->iField)
        // Якщо новий елемент "менший", ніж що зберігається у вузлі
        // йдемо по лівій гілціидем по левой ветви
        pCurItem = pCurItem ->pLeft;
      else
        // інакше - по правійиначе - по правой
        pCurItem = pCurItem ->pRight;
    }
  }
```

Пошук закінчується або досягши листа дерева, або якщо вузол з таким полем вже є. Після завершення пошуку необхідно вставити в дерево новий вузол (малюнок 10), якщо такого ще там немає (нагадуємо, що в змінній pparcnt зберігається покажчик на батька вузла, що вставляється) :Поиск заканчивается либо при достижении листа дерева, либо если узел с таким полем уже есть. После завершения поиска необходимо вставить в дерево новый узел (рисунок 10), если такого еще там нет (напоминаем, что в переменной pparcnt хранится указатель на отца вставляемого узла):

```
  if(!nFound){if(!nFound) {
    // Такого елементу в дереві немає - створюємо новий вузол
    pNewItem = create(pItem);
    pNewItem ->pLeft = pNewItem ->pRight = 0;
    if(pParent == 0) {if(pParent == 0)  {
      // Перший вузол дерева - створюємо його і робимо коренем
      *ppRoot = create(pItem);
      (*ppRoot) ->pLeft = (*ppRoot) ->pRight = 0;
    } else {else {
      // Якщо новий елемент менший, ніж що зберігається в поточному вузлі
      if(pItem ->iField < pParent ->iField)
        pParent ->pLeft = pNewItem;       // вставляємо його в ліву гілкурParent->pLeft = pNewItem;
// вставляем его в левую ветвь
      else
        pParent ->pRight = pNewItem;      // інакше - в правурParent->pRight = pNewItem;   // иначе - в
правую
    }
  }
```

Очевидно, що "зовнішній вигляд" дерева залежить від порядку включенні в нею елементів. Але у будь-якому випадку це буде впорядковане бінарне дерево, в якому номер лівого сина завжди менший, а правого — більше, ніж номер батька.Очевидно, что "внешний вид" дерева зависит от порядка включении в нею элементов. Но в любом случае это будет упорядоченное бинарное дерево, в котором номер левого сына всегда меньше, а правого — больше, чем номер отца.

Підведемо підсумок: для того, щоб вставити в дерево новий вузол, необхідно знайти для нього батька (що іноді називається *батьком)* і зробити його лівим або правим сином, залежно від значення ключового поля нового вузла.Подведем итог: для того чтобы вставить в дерево новый узел, необходимо найти для него отца (иногда называемого *родителем)* и сделать его соответственно левым или правым сыном, в зависимости от значения ключевого поля нового узла.

Розглянемо видалення вузла з бінарного дерева. В цьому випадку усе дещо складніше. Природно, передусім необхідно знайти вузол, який вимагається видалити. Робиться це точно так, як і при вставці (функція **insert**). Результатом пошуку може бути один з трьох варіантів :Рассмотрим удаление узла из бинарного дерева. В этом случае все несколько сложнее. Естественно, прежде всего необходимо найти узел, который требуется удалить. Делается это

точно так же, как и при вставке (функция **insert**). Результатом поиска может быть один из трех вариантов:

- Вузла із заданим ключем в дереві немає.
- Вузол із заданим ключем має не більше одного нащадка.
- Вузол із заданим ключем має двох нащадків.

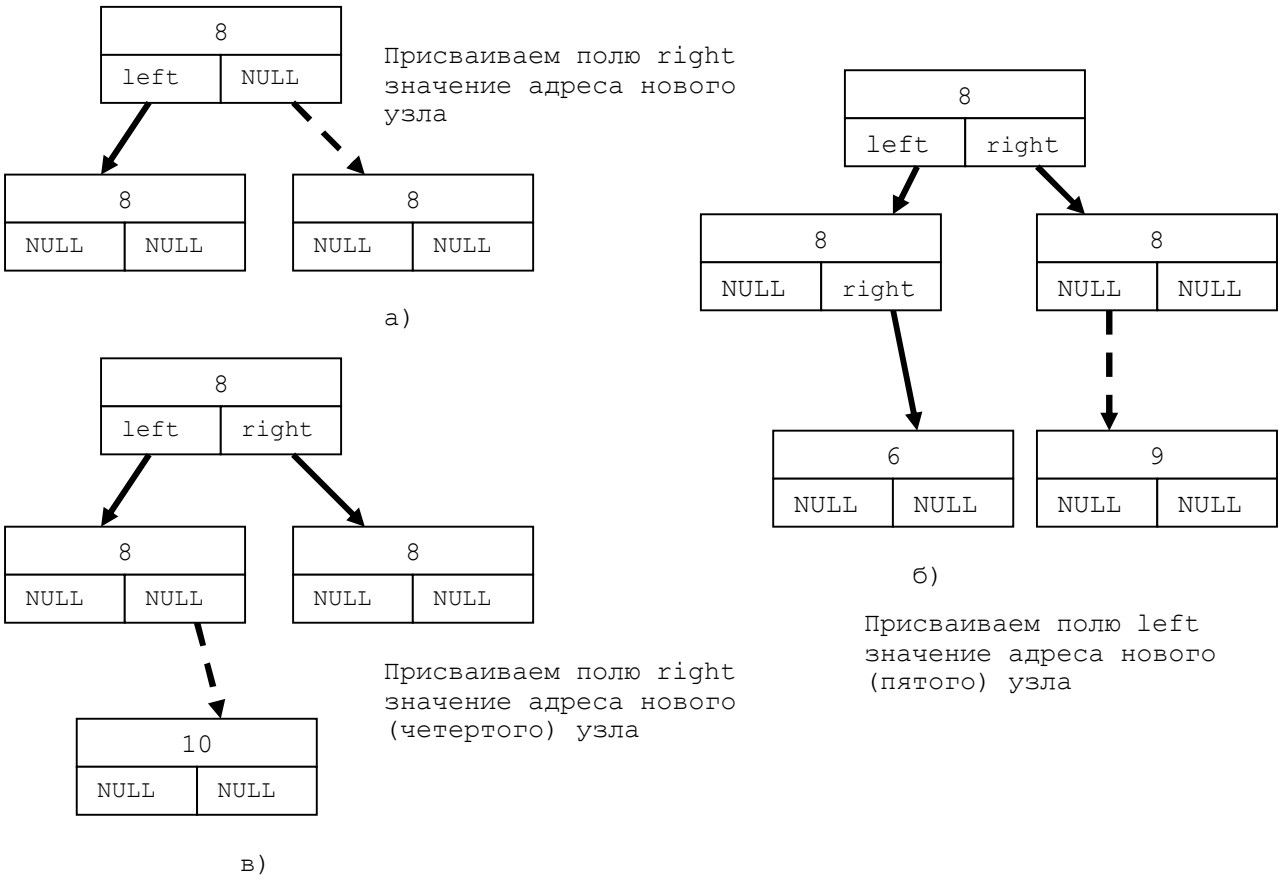


Рисунок 10 –Включение узлов в упорядоченное дерево

Действия в первом случае очевидны — завершаем работу функции. Второй случай (исключаемый элемент— лист или узел с одним потомком) также достаточно прост:

```
*pPreItem = 0;
if(nFound) {
if(pPresent->pLeft == 0)
  // Если нет левого поддерева, то удаляемый элемент
  // заменяем на правое поддерево
 pReplace = pPresent->pRight;
else
  // Если нет правого поддерева, то удаляемый элемент
  // заменяем на левое поддерево
 if(pPresent->pRight == 0)
  pReplace = pPresent->pLeft;
 else {
 . . .  // Здесь код, обрабатывающий случай наличия обоих поддереаьев
}
if(pPreItem == 0)
   // Если удаляется корень,
   // то выбранный ранее сын становится корнем дерева
```

```
    *ppRoot = pReplace;
  else
    // В противном случае заменяем значение
    // соответствующего указателя
   if(pPresent == pPreItem->pLeft)
     pPreItem->pLeft = pReplace;
   else
    pPreItem->pRight = pReplace;
    // Не забываем освободить память, занимаемую удаленным узлом
   free(pPresent);
  }
```

Трудность возникает, если нужно удалить узел с двумя потомками. В этом случае удаляемый узел нужно заменить либо на самый правый элемент его левого поддерава, либо на самый левый элемент его самого правого поддерева, причем они длжны иметь максимум одного потомка:

```
  pParent = pPresent;
  pReplace = pPresent->pRight;
  pTmp = pPresent->pLeft;

   // Ищем самый левый лист
  while(pTmp != 0)  {
   pParent = pReplace;
   pReplace = pTmp;
   pTmp = pReplace->pLeft;
  }
   // Если это корень
  if(pParent != pPresent) {
     // Заменяем на найденный лист
   pParent->pLeft = pReplace->pRight;
   pReplace->pRight = pPresent->pRight;
  }
  pReplace->pRight = pPresent->pRight;
```

Рассмотренный алгоритм удаления узла дерева иллюстрируется на рисунке 11, где приведено исходное дерево (а), из которого последовательно удаляются вершины с ключами 13, 15, 5 и 10. Код простой тестовой функции приведен в листинге 12.

Листинг 12

```
  #include <string.h>
  #include <stdio.h>
  #include <malloc.h>
  #include "tree.h"

  TREE *pMyTree;

  void main()  {
   TREE *pItem = (TREE *)malloc(sizeof(TREE));
   TREE *pItem1 = (TREE *)malloc(sizeof(TREE));

   pItem->iField = 10;
   insert(&pMyTree, pItem);

   pItem->iField = 5;
```

```
insert(&pMyTree, pItem);

pItem->iField = 3;
insert(&pMyTree, pItem);

pItem->iField = 15;
insert(&pMyTree, pItem);

pItem1->iField = 8;
insert(&pMyTree, pItem1);

pItem1->iField = 13;
```

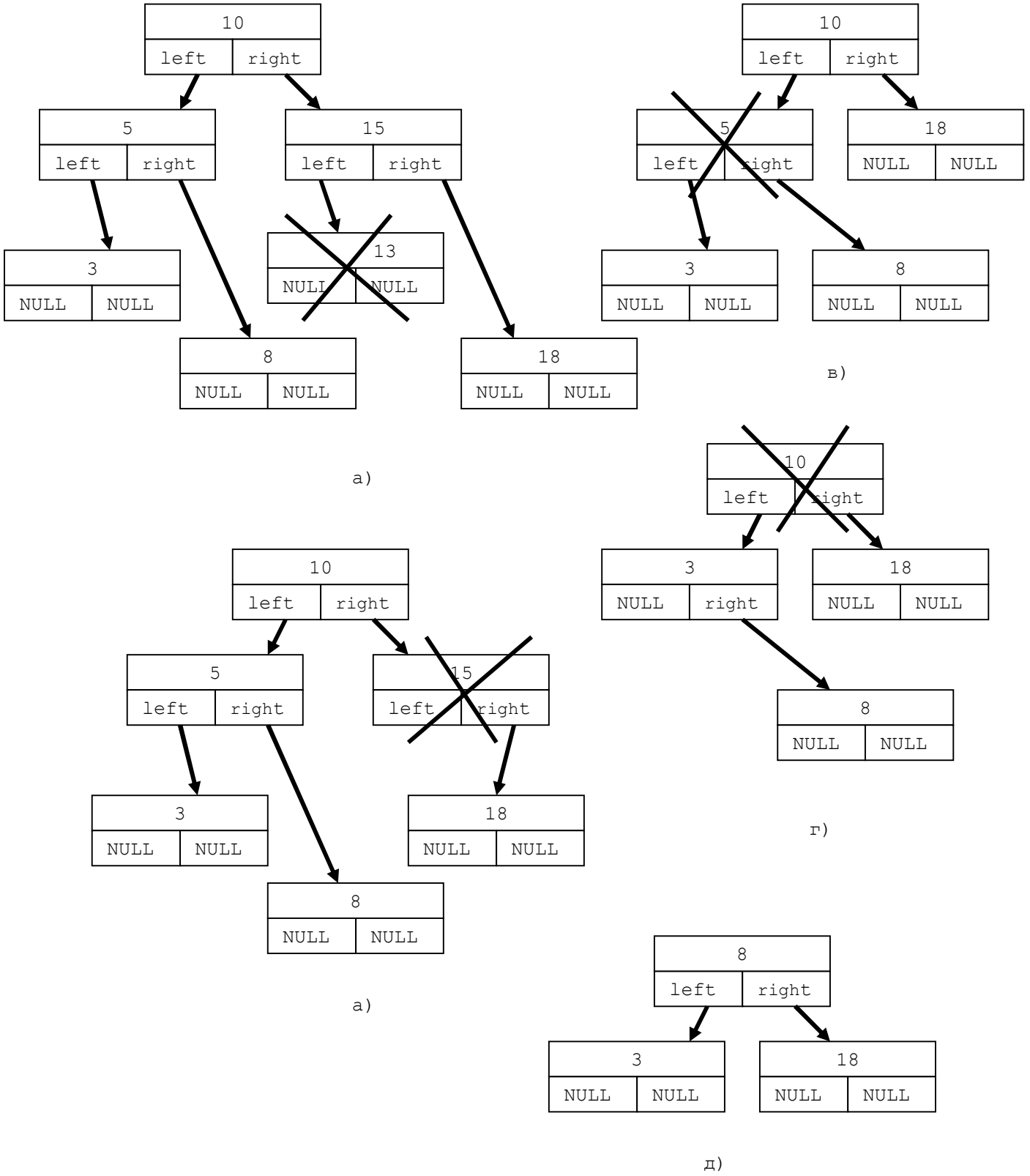


Рисунок 11 - Исключение узлов из дерева

insert(&pMyTree, pItem1);

pItem1->iField = 18;
insert(&pMyTree, pItem1);

display(pMyTree);

```
    printf("\n");

    pItem->iField = 13;
    remove(&pMyTree, pItem);
    display(pMyTree);
    printf("\n");

    pItem1->iField = 15;
    remove(&pMyTree, pItem1);
    display(pMyTree);
    printf("\n");

    pItem->iField = 5;
    remove(&pMyTree, pItem);
    display(pMyTree);
    printf("\n");

    pItem->iField = 10;
    remove(&pMyTree, pItem);
    display(pMyTree);
    printf("\n");

    destroy(pMyTree);
    getchar();
  }
```

Порядок виконання роботи

1. Уважно прочитайте і вивчіть опис лабораторної роботи. Підготуйте письмовий звіт.

2. Покажіть поисьменный звіт викладачеві і з його дозволу приступайте до виконання лабораторної роботи.

3. У особистому каталозі створіть теку «Лаб_робота_2», відкрийте її. Для кожної з частин лабораторної роботи створіть наступні підкаталоги: Stack, Calc, List, Tree.В личном каталоге создайте папку «Лаб_работа_2», откройте ее. Для каждой из частей лабораторной работы создайте следующие подкаталоги: Stack, Calc, List, Tree.

4. У підкаталог Stack скопіюйте файли Stack.c, MainStack.c і Stack.h з підкаталогу «Частина_1» теки з описом лабораторної роботи.В подкаталог Stack скопируйте файлы Stack.c, MainStack.c и Stack.h из подкаталога «Часть_1» папки с описанием лабораторной работы.

5. У підкаталог Calc скопіюйте файли Calc.c, Queue.c, Stack.c, Stack.h і Queue.h з підкаталогу «Частина_2» теки з описом лабораторної роботи.В подкаталог Calc скопируйте файлы Calc.c, Queue.c, Stack.c, Stack.h и Queue.h из подкаталога «Часть_2» папки с описанием лабораторной работы.

6. У підкаталог List скопіюйте файли List.c. MainList.c і List.h з підкаталогу «Частина_3» теки з описом лабораторної роботи.В подкаталог List скопируйте файлы List.c. MainList.c и List.h из подкаталога «Часть_3» папки с описанием лабораторной работы.

7. У підкаталог Tree скопіюйте файли Tree.c, MainTree.c і Tree.h з підкаталогу «Частина_4» теки з описом лабораторної роботи.В подкаталог Tree скопируйте файлы Tree.c, MainTree.c и Tree.h из подкаталога «Часть_4» папки с описанием лабораторной работы.

8 Запустіть на виконання Visual C++, створіть новий проект і включіть в нього файли з теки Stack. Виконайте зборку програми Stack, відлагодьте і протестуйте її, працюючу програму покажіть викладачеві.Запустите на выполнение Visual C++, создайте новый проект и включите в него файлы из папки Stack. Выполните сборку программы Stack, отладьте и протестируйте ее, работающую программу покажите преподавателю.

9. Виконайте усі дії, пов'язані з відладкою і тестуванням інших частин лабораторної роботи.

10. Закінчіть сеанс роботи з операційною системою.

11. Захистіть звіт про виконану лабораторну роботу.Защитите отчет о выполненной лабораторной работе.

Звіт повинен містити:

1) Найменування і мета роботи.
2) Опис вирішуваної задачі для кожної з чотирьох частин лабораторної роботи.
3) Тексти програм з необхідними коментарями.
4) Висновки по лабораторній роботі.

Контрольні питання

1. Дайте визначення стеку.
2. Поясніть операції вставки і видалення елементів в стек
3. Яка структура даних називається чергою?
4. За яких умов черга стає циклічною?
5. Які операції в лабораторній роботі виконуються зі списками?
6. Які списки досліджуються в лабораторній роботі?
7. Перерахуйте типи списків.
8. Як реалізована операція видалення елементу зі списку в лабораторній роботі?
9 Яка структура даних називається бінарним деревом?
10. Поясніть, як реалізована операція вставки елементу бінарного дерева в лабораторній роботі?
11. Поясніть, як реалізована операція видалення елементу бінарного дерева в лабораторній роботі?
12. Які існують способи обходу бінарного дерева?
13. Що таке рівень вузла у бінарному дереві? Як він визначається?

Лабораторна робота 3

Створення Windows -приложения

Мета роботи — Вивчити структуру типової Windows -программы, основні функціональні частини функції WinMain, основні структури, використовувані при створенні додатків. Навчитися управляти зовнішнім виглядом вікна. Освоїти техніку програмування додатків, працюючими з повідомленнями Windows.

Короткий огляд системи Windows

На відміну від традиційного процедурного, програмування для Windows є об'єктно-орієнтованим. Система Windows є набором об'єктів, найважливішим з яких є вікно.В отличие от традиционного процедурного, программирование для Windows является объектно-ориентированным. Система Windows представляет собой набор объектов, самым важным из которых является окно.

Вікно першим з'являється при запуску системи або будь-якого застосування і останнім зникає при завершенні роботи. Спілкування з об'єктом-вікном можливо за допомогою повідомлень. Механізм повідомлень в Windows реалізується за допомогою черги повідомлень, підтримуваною операційною системою. Кожне повідомлення потрапляє в таку чергу, де і чекає подальшої обробки. Повідомлення визначається зовнішньою подією, яка описує деяку зміну, що сталася в тій, що оточує додаток середовищу.Окно первым появляется при запуске системы или любого приложения и последним исчезает при завершении работы. Общение с объектом-окном возможно при

помощи сообщений. Механизм сообщений в Windows реализуется при помощи очереди сообщений, поддерживаемой операционной системой. Каждое сообщение попадает в такую очередь, где и ожидает последующей обработки. Сообщение определяется внешним событием, которое описывает некоторое изменение, произошедшее в окружающей приложение среде.

Кожне вікно в Windows має спеціальну функцію, яка викликається для обслуговування повідомлень. Операційна система може тільки послати повідомлення окну, а воно вже саме вирішує, що з ним робити. Точніше, програміст визначає реакцію вікна на отримувані повідомлення. Впродовж цього процесу вікно діє як автономний об'єкт, що живе своїм власним життям.Каждое окно в Windows имеет специальную функцию, которая вызывается для обслуживания сообщений. Операционная система может только послать сообщение окну, а оно уже само решает, что с ним делать. Точнее, программист определяет реакцию окна на получаемые сообщения. В течение этого процесса окно действует как автономный объект, живущий своей собственной жизнью.

Перше, з чим стикається користувач, включивши комп'ютер зі встановленою версією Windows, — це графічний призначений для користувача інтерфейс. Немає ніякого командного рядка або чогось подібного. Розробники системи прагнули не лише до того, щоб система була зручна і інтуїтивно зрозуміла, але і щоб додатки і системи третіх фірм по своєму інтерфейсу якомога менше відрізнялися від програм, розроблених фірмою Microsoft. Для цього операційна система підтримує безліч об'єктів, на основі яких можна реалізувати велику частину того, що може знадобитися розробникам. Перерахуємо деякі з них.Первое, с чем сталкивается пользователь, включив компьютер с установленной версией Windows, — это графический пользовательский интерфейс. Нет никакой командной строки или чего-то подобного. Разработчики системы стремились не только к тому, чтобы система была удобна и интуитивно понятна, но и чтобы приложения и системы третьих фирм по своему интерфейсу как можно меньше отличались от программ, разработанных фирмой Microsoft. Для этого операционная система поддерживает множество объектов, на основе которых можно реализовать большую часть того, что может потребоваться разработчикам. Перечислим некоторые из них.

Елементи управління

Вікно (window) — прямокутна область екрану для організації обміну між користувачем і додатком. Вікно спільно використовує екран з іншими вікнами, у тому числі і інших застосувань. Одноразово користувач може здійснювати введення тільки в одне вікно і додаток, до якого це вікно відноситься.*Окно* (window) — прямоугольная область экрана для организации обмена между пользователем и приложением. Окно совместно использует экран с другими окнами, в том числе и других приложений. Единовременно пользователь может осуществлять ввод только в одно окно и приложение, к которому это окно относится.

Блок діалогу (Dialog Box) — тимчасове вікно, що створюється .для запиту введення користувача. Зазвичай блоки діалогу використовуються для отримання додаткової інформації, містять деяку кількість елементів управлінні, що дозволяють вводити текст, встановлювати перемикачі і так далі. Діалоги можуть бути як модальні, так і немодальні. Модальні блоки діалогу використовуються у разі, коли для продовження роботи від користувача потрібно введення деяких даних або відповідь на питання. Інші елементи (вікна) додатка у цей момент недоступні. Немодальні діалоги можуть використовуватися для постійною відображення на екрані, наприклад, для швидкої зміни якого-небудь параметра, з негайною реакцією додатка на зроблені зміни.*Блок диалога* (Dialog Box) — временное окно, создаваемое .для запроса ввода пользователя. Обычно блоки диалога используются для получения дополнительной информации, содержат некоторое количество элементов управлении, позволяющих вводить текст, устанавливать переключатели и т.д. Диалоги могут быть как модальные, так и немодальные. Модальные блоки диалога используются в случае, когда для продолжения работы от пользователя требуется ввод некоторых данных или ответ на вопрос. Другие элементы (окна) приложения в этот момент недоступны. Немодальные диалоги могут использоваться для постоянною отображения на экране, например, для быстрого изменения какого-либо параметра, с немедленной реакцией приложения на предпринятые изменения.

Елемент управління (Control) — дочірнє вікно, яке додаток використовує для взаємодії з іншим вікном, обеспечиная прості операції введення/виводу. Найчастіше

елементи управління використовуються р блоках діалогу, хоча їх можна використати і при роботі з вікнами інших типів. Далі перераховані усі зумовлені елементи управління.*Элемент управления* (Control) — дочернее окно, которое приложение использует для взаимодействия с другим окном, обеспечиная простейшие операции ввода/вывода. Чаще всего элементы управления используются р блоках диалога, хотя их можно использовать и при работе с окнами других типов. Далее перечислены все предопределенные элементы управления.

Кнопка (BUTTON) зазвичай повідомляє, що користувач вибрав цей елі мент. Розрізняють декілька видів кнопок; командна (звичайна), така, що спрацьовує по натисненню (PUSHBUTTON), кнопка вибору — перемикач (RAD10BUTTON), прапорець (CHECKBOX) і рамка (GROUP BOX).*Кнопка* (BUTTON) обычно уведомляет, что пользователь выбрал этот эле мент. Различают несколько видов кнопок; командная (обычная), срабатывающая по нажатию (PUSHBUTTON), кнопка выбора — переключатель (RAD10BUTTON), флажок (CHECKBOX) и рамка (GROUP BOX).

Редактор (EDIT) — простий редактор, що дозволяє користувачеві переглядати і редагувати текст.*Редактор* (EDIT) — простейший редактор, позволяющий пользователю просматривать и редактировать текст.

Список (L1STBOX) — елемент відображення списку елементів. З його допомогою користувач може вибрати один або декілька елементів списку.*Список* (L1STBOX) — элемент отображения списка элементов. С его помощью пользователь может выбрать один или несколько элементов списка.

Комбінований список (COMBOBOX) — комбінація списку і однорядкового елементу редагування. Дозволяє користувачеві вибрати і редагувати елемент списку.*Комбинированный список* (COMBOBOX) — комбинация списка и однострочного элемента редактирования. Позволяет пользователю выбрать и редактировать элемент списка.

Смуга прокрутки (SCROLLBAR) — елемент управління, що дозволяє користувачеві вибрати напрям і відстань прокрутки інформації в пов'язаному з елементом вікні.*Полоса прокрутки* (SCROLLBAR) — элемент управления, позволяющий пользователю выбрать направление и расстояние прокрутки информации в связанном с элементом окне.

Статичний елемент — мітка (STATIC) — елемент, що виключає дію на нього з боку користувача, зазвичай використовується як напис для інших елементів управління.*Статический элемент — метка* (STATIC) — элемент, исключающий воздействие на него со стороны пользователя, обычно используется в качестве надписи для других элементов управления.

Операційна система надає декілька шляхів для створенні призначених для користувача елементів управління :

1. Для кнопок, списків і комбінованих списків можна використати стиль *самоотображения* (owner - draw), який має на увазі написання коду для самостійного прорисовывании елементу.Для кнопок, списков и комбинированных списков можно использовать стиль *самоотображения* (owner-draw), который подразумевает написание кода для самостоятельного прорисовывании элемента.

2. Спадкоємство зумовленого елементу управління із заміною функції вікна, в якій можна передбачити будь-які можливості, що цікавлять, залишаючи можливість використання частини коду стандартної реалізації.

3. Створення власного класу вікна.

Окрім зумовлених елементів управління (CONTROLS), 32-розрядні операційні системи Windows підтримують велике колпчество додаткових елементів управління (COMMON CONTROLS). Полдержка здійснюється за допомогою спеціальної динамічно підвантажуваної бібліотеки. Як і інші елементи управлінням додаткові є дочірні вікна, які додаток використовує для взаємодії з іншими вікнами, забезпечуючи прості операції введення/виводу. Відмінність від звичайних (стандартних) елементів управління состо- ит н більшій їх різноманітності і функціональності. Крім того, цілий ряд таких елементів активно використовується окрім блоків діалогу у вікнах інших типів. Далі перераховані усі наявні додаткові елементи управлінняКроме предопределенных элементов управления (CONTROLS), 32-разрядные операционные системы Windows поддерживают большое колпчество дополнительных элементов управления (COMMON CONTROLS). Полдержка осуществляется посредством специальной динамически подгружаемой библиотеки. Как и остальные элементы управление дополнительные представляют собой дочерние окна, которые приложение использует для взаимодействия с другими окнами, обеспечивая простейшие операции ввода/вывода. Отличие от обычных (стандартных) элементов управления состо- ит н большем их

разнообразии и функциональности. Кроме того, целый ряд таких элементов активно используются помимо блоков диалога в окнах других типов. Далее перечислены все имеющиеся дополнительные элементы управления,

Змінюваний список (DRAG LIST BOX) — спеціальний тип списку, що дозволяє перетягувати свої елементи, міняючи порядок відображення.*Изменяемый список* (DRAG LIST BOX) — специальный тип списка, позволяющий перетаскивать свои элементы, меняя порядок отображения.

Заголовок (HEADER) — дочірнє вікно, що зазвичай розташовується над колонками тексту або цифр. Містить заголовок для кожного стовпця і дозволяє міняти ширину стовпців.*Заголовок* (HEADER) — дочернее окно, обычно располагаемое над колонками текста или цифр. Содержит заголовок для каждого столбца и позволяет менять ширину столбцов.

Перегляд списку (LIST VIEW) — потужний засіб відображення елементів списку. Дозволяє відображати, окрім тексту, пов'язані з елементами піктограми. Підтримує чотири способи відображення. Три з них розташовують елементи з піктограмами по усьому полю відображення, четвертий спосіб дозволяє створити список, що складається з набору стовпців, що мають заголовки.*Просмотр списка* (LIST VIEW) — мощное средство отображения элементов списка. Позволяет отображать, помимо текста, связанные с элементами пиктограммы. Поддерживает четыре способа отображения. Три из них располагают элементы с пиктограммами по всему полю отображения, четвертый способ позволяет создать список, состоящий из набора столбцов, имеющих заголовки.

Рядок стану (STATUS LINE,) — горизонтальне вікно в нижній частині батьківського вікна, що служить для відображення різної інформації додатка, наприклад, підказки по вибраному пункту меню. Рядок стану може бути розбитий на декілька частин для відображення координат, часу, індикації натиснення спеціальних клавіш (<INS>, <SCROLL LOCK>, <NUM LOCK>) і так далі*Строка состояния* (STATUS LINE,) — горизонтальное окно в нижней части родительского окна, служащее для отображения различной информации приложения, например, подсказки по выбранному пункту меню. Строка состояния может быть разбита на несколько частей для отображения координат, времени, индикации нажатия специальных клавиш (<INS>, <SCROLL LOCK>, <NUM LOCK>) и т.д.

Панель інструментів (TOOLBAR) ~ дочірнє вікно, що складається з набору кнопок для швидкого виклику тих або інших команд.*Панель инструментов* (TOOLBAR) ~ дочернее окно, состоящее из набора кнопок для быстрого вызова тех или иных команд.

Підказка (TOOLTIP) — невелике спливаюче вікно однорядкового тексту з коротким описом деякого об'єкту додатка, наприклад, кнопки або іншого елементу управління*Подсказка* (TOOLTIP) — небольшое всплывающее окно однострочного текста с кратким описанием некоторого объекта приложения, например, кнопки или другого элемента управления,

Перегляд дерев (tree — view) — вікно перегляду даних, організованих у вигляді дерева, наприклад, утримуваного диска або каталогу (теки). Кожен елемент дерева може додатково до тексту мати і деяке растрове (бітове) зображення. Вузли дерева можна розкривати, тобто відображати елементи вузла, або навпаки — ховати їх.*Просмотр деревьев* (tree-view) — окно просмотра данных, организованных в виде дерева, например, содержимого диска или каталога (папки). Каждый элемент дерева может дополнительно к тексту иметь и некоторое растровое (битовое) изображение. Узлы дерева можно раскрывать, т.е. отображать элементы узла, либо наоборот — прятать их.

Вкладка (TAB) — засіб, що дозволяє створювати елементи, аналогічні закладкам книги, які в реальному житті використовуються при читанні. Вкладки використовуються для зміни сторінок діалогового вікна.*Вкладка* (TAB) — средство, позволяющее создавать элементы, аналогичные закладкам книги, которые в реальной жизни используются при чтении. Вкладки используются для смены страниц диалогового окна.

Вікно властивостей (PROPERTY SHEET) — вікно, що дозволяє переглядати і редагувати властивості деякого об'єкту. Гідність цього елементу управління в порівнянні із звичайним блоком діалогу полягає в тому, що вони дозволяють згрупувати велику кількість параметрів, а доступ до кожної такої групи здійснювати, використовуючи вкладки (TAB). В той же час на базі вікна властивостей можна будувати так звані майстри, де перехід від одного етапу запиту користувача до подальшого або попереднього здійснюється за допомогою відповідних кнопок. Створення кожною окремого вікна для групи властивостей або групи запитів (для майстра) полягає в підготовці ресурсу діалогу.*Окно свойств* (PROPERTY SHEET) —

окно, позволяющее просматривать и редактировать свойства некоторого объекта. Достоинство этого элемента управления по сравнению с обычным блоком диалога состоит в том, что они позволяют сгруппировать большое количество параметров, а доступ к каждой такой группе осуществлять, используя вкладки (TAB). В то же время на базе окна свойств можно строить так называемые мастера, где переход от одного этапа запроса пользователя к последующему или предыдущему осуществляется при помощи соответствующих кнопок. Создание каждою отдельного окна для группы свойств или группы запросов (для мастера) заключается в подготовке ресурса диалога.

Клавиш виклику (HOT — KEY) — комбінація клавіш, яка надалі може бути використана для швидкого виконання команди. Цей елемент відображає призначений для користувача вибір і перевіряє його допустимість.*Клавиш вызова* (HOT-KEY) — комбинация клавиш, которая в дальнейшем может быть использована для быстрого выполнения команды. Этот элемент отображает пользовательский выбор и проверяет его допустимость.

Елемент управління з форматуванням — розширений редактор (RICH EDIT) — вікно редагування. На відміну від зумовленого елементу редагування (EDIT) — редактор RTF -формата дозволяє управляти шрифтами, кольором, розміром, вирівнюванням абзаців. Крім того, цей елемент управління підтримує роботу з впровадженими об'єктами OLE.*Элемент управления с форматированием — расширенный редактор* (RICH EDIT) — окно редактирования. В отличие от предопределенного элемента редактирования (EDIT) — редактор RTF-формата позволяет управлять шрифтами, цветом, размером, выравниванием абзацев. Кроме того, этот элемент управления поддерживает работу с внедренными объектами OLE.

Лінійка з бігунком (TRACKBAR) — елемент, що складається з бігунка і, можливо, градуйованої лінійки. Зміна положення бігунка робиться за допомогою клавіатури, або мишею. При цьому бігунок посилає повідомляючі повідомлення про зміну свого стану.*Линейка с бегунком* (TRACKBAR) — элемент, состоящий из бегунка и, возможно, градуированной линейки. Изменение положения бегунка производится при помощи клавиатуры, либо мышью. При этом бегунок посылает уведомляющие сообщения об изменении своего состояния.

Перегляд відеокліпів (ANIMATION) — цей елемент дозволяє переглядати відеокліпи, що складаються з набору бітових зображень. Такі кліпи мають бути записані у форматі Audio Video Interleaved (AVI). Цей елемент придатний ддя відображення тільки кліпів, що не містять звукового оформлення.*Просмотр видеоклипов* (ANIMATION) — этот элемент позволяет просматривать видеоклипы, состоящие из набора битовых изображений. Такие клипы должны быть записаны в формате Audio Video Interleaved (AVI). Этот элемент пригоден ддя отображения только клипов, не содержащих звукового оформления.

Індикатор (PROGRESS BAR) — засіб індикації деякого тривалого процесу або операції, наприклад, копіювання файлів. Реалізується у вигляді прямокутника, що поступово заповнюється у міру завершення опеоацми.*Индикатор* (PROGRESS BAR) — средство индикации некоторого продолжающегося процесса или операции, например, копирования файлов. Реализуется в виде прямоугольника, постепенно заполняемого по мере завершения опеоацми.

Спін (UP — DOWN) — пара кнопок із стрілками для збільшення або зменшення значення пов'язаного з ним елементу управління, наприклад, елементу редагування, що містить чисельну інформацію*Спин* (UP-DOWN) — пара кнопок со стрелками для увеличения или уменьшения значения связанного с ним элемента управления, например, элемента редактирования, содержащего численную информацию,

Список малюнків (IMAGE LIST) — допоміжний елемент, використовуваний, передусім, іншими загальними елементами управління. Є списком співпадаючих за розміром піктограм і бітових масивів, доступ до яких здійснюється по індексу. Використовується з елементами перегляду списку і дерев, для зберігання піктограм вкладок і так далі*Список рисунков* (IMAGE LIST) — вспомогательный элемент, используемый, прежде всего, другими общими элементами управления. Представляет собой список совпадающих по размеру пиктограмм и битовых массивов, доступ к которым осуществляется по индексу. Используется с элементами просмотра списка и деревьев, для хранения пиктограмм вкладок и т.д.

Ресурси додатка

Кожне застосування зазвичай має цілий ряд стандартних або призначених для користувача ресурсів. Ресурсами додатка називаються масиви даних, які розробник може додати у виконуваний модуль. Нижче перераховані стандартні ресурси:

Акселератори (accelerators) — структури даних, списки гарячих клавіш і команд, що асоціюються з ними, що містять,,*Акселераторы* (accelerators) — структуры данных, содержащие списки горячих клавиш и команд, ассоциированных с ними,,

Китові масиви (bitmaps) — масиви точок для відображення на екрані, л якості, наприклад, пунктів меню або елементів управлінні.*Китовые массивы* (bitmaps)— массивы точек для отображения на экране, л качестве, например, пунктов меню или элементов управлении.

Курсори (cursors) — бітові масиви, що використовуються в якість растрових зображень курсорів миші.*Курсоры* (cursors) — битовые массивы, использующиеся в качество растровых изображений курсоров мыши.

Шаблони діалогів (dialogs) — структури, що описують блок лиачоги, включаючи елементи управління, віконні стилі, положення на екрані і інші параметри. Використовується для виводу на екран блоків діалогу.*Шаблоны диалогов* (dialogs) — структуры, описывающие блок лиачоги, включая элементы управления, оконные стили, положение на экране и другие параметры. Используется для вывода на экран блоков диалога.

Піктограми (icons) — бітові масиви, що використовуються для візуального представлення різних об'єктів в системі, — додатків, документів, елементів списків деяких типів і так далі*Пиктограммы* (icons) — битовые массивы, использующиеся для визуального представления различных объектов в системе — приложений, документов, элементов списков некоторых типов и т.д.

Шаблони меню (menus) — описують пункти, пов'язані з командами, що задаються користувачем.*Шаблоны меню* (menus) — описывают пункты, связанные с задаваемыми пользователем командами.

Строкові таблиці (string tables) — списки статичних символьних травні сивої, доступних додатку по ідентифікатору.*Строковые таблицы* (string tables) — списки статических символьных мае сивой, доступных приложению по идентификатору.

Шаблони *панелей інструментів* (toolbars) — описують елементи управління, що складаються з набору бітових масивів в якості кнопок, кожна з яких пов'язана з певною командою.Шаблоны *панелей инструментов* (toolbars) — описывают элементы управления, состоящие из набора битовых массивов в качестве кнопок, каждая из которых связана с определенной командой.

Описи версій (version) — статичні структури спеціального виду, що містять інформацію про програму, включаючи ім'я додатка, помер версії, інформацію про авторські права і так далі. Використовується операційною системою для висновку інформації про програму.*Описания версий* (version) — статические структуры специального вида, содержащие информацию о программе, включая имя приложения, помер версии, информацию об авторских правах и т.д. Используется операционной системой для вывода информации о программе.

Окрім стандартних, розробник може створювати і будь-які призначені для користувача ресурси, проте в цьому випадку він не зможе користуватися стандартними функціями доступу до них, оскільки системі невідома їх семантика.

Використання терміну *ресурси* може здатися дещо несподіваним, проте, додатки працюють з вищепереліченими об'єктами дійсно як з ресурсами, причому що розділяються, тому що декілька копій додатка можуть користуватися тільки однією копією деякого ресурсу.Использование термина *ресурсы* может показаться несколько неожиданным, тем не менее, приложения работают с вышеперечисленными объектами действительно как с ресурсами, причем разделяемыми, потому что несколько копий приложения могут пользоваться только одной копией некоторого ресурса.

Додаткова перевага використання ресурсів додатка в тому, що усі вони зосереджені у файлі, що є окремою одиницею компіляції, що надзвичайно корисно, наприклад, за наявності великої кількості строкових повідомлень, які у разі їх реалізації у вигляді ресурсів, не розкидані по різних файлах. Особливо корисне використання ресурсів для створення шаблонів діалогів, шаблонів меню і акселераторів.

Доступ до ресурсів здійснюється по числових або строкових ідентифікаторах.

Інтерфейс програмування додатків

Інтерфейс програмування додатку — (Application Program Interface, API) — це набір необхідних функцій, за допомогою яких будь-яке застосування може взаємодіяти з операційною системою.*Интерфейс программирования приложении* — (Application Program Interface, API) — это набор необходимых функций, при помощи которых любое приложение может взаимодействовать с операционной системой.

У рамках Windows з'явилося декілька таких API. Найперший був розроблений для 16-розрядних операційних систем Microsoft. Природно з появою 32-розрядних Windows NT і Windows 95 з'явився і 32-розрядний інтерфейс — Win32. API — ця сполучна ланка між додатками і операційною системою, воно містить близько 2000 функцій і декілька тисяч повідомлень, макросів і зумовлених констант.В рамках Windows появилось несколько таких API. Самый первый был разработан для 16-разрядных операционных систем Microsoft. Естественно с появлением 32-разрядных Windows NT и Windows 95 появился и 32-разрядный интерфейс — Win32. API — это связующее звено между приложениями и операционной системой, оно содержит около 2000 функций и несколько тысяч сообщений, макросов и предопределенных констант.

Win32 API складається з ядра: Kernel, User і GDI, які забезпечують інтерфейс з операційною системою, управлінням вікнами і додатками, а також підтримку графіки, і декількох додаткових компонентів.Win32 API состоит из ядра: Kernel, User и GDI, которые обеспечивают интерфейс с операционной системой, управлением окнами и приложениями, а также поддержку графики, и нескольких дополнительных компонентов.

Щоб можна було скористатися функціями API, кожне застосування, написане для Windows, містить заголовний файл **windows.h**, який включає допоміжні файли, що містить визначення констант, оголошень typedef і прототипи функцій. Там же знаходиться величезне число макросів.Чтобы можно было воспользоваться функциями API, каждое приложение, написанное для Windows, содержит заголовочный файл **windows.h**, который включает вспомогательные файлы, содержащий определения констант, объявлений typedef и прототипы функций. Там же находится огромное число макросов.

Вимоги до структури Windows -приложения

Будь-яке Windows -приложение, що має інтерфейс з користувачем (за винятком консольних застосувань), функціонально складається з двох основних частин: функції winMain і так званій функції вікна (віконної процедури).Любое Windows-приложение, имеющее интерфейс с пользователем (за исключением консольных приложений), функционально состоит из двух основных частей: функции winMain и так называемой функции окна (оконной процедуры).

Кожна Windows -програма починає своє виконання, природно, якщо вона написана на З або C++, з функції winMain. Оголошення цієї функції у рамках Win32 API і опис аргументів наступні:Каждая Windows-программа начинает свое выполнение, естественно, если она написана на C или C++, с функции winMain. Объявление этой функции в рамках Win32 API и описание аргументов следующие:

```
int WINAPI WinMain {
  HINSTANCE hInstance
  HINSTANCE hPrevInstance
  LPSTR lpCmdLine
  int nCmdShow);
```

hInstance — дескриптор (унікальне число), що асоціюється з поточним застосуванням; **hPrevInstance** — дескриптор попередньої запущеної копії додатка, якщо така є; **lpCmdLine** — покажчик на командний рядок; **nCmdShow** — режим початкового відображення головного вікна додатка, тобто в якому виді — нормальному, мінімізованому або такому, що максимізував буде відображено вікно додатка.**hInstance** — дескриптор (уникальное число), ассоциируемый с текущим приложением; **hPrevInstance** — дескриптор предыдущей запущенной копии приложения, если такая имеется; **lpCmdLine** — указатель на командную строку; **nCmdShow** — режим начального отображения главного окна приложения, т.е. в каком виде — нормальном, минимизированном или максимизированном будет отображено окно приложения.

Ось приклад простої програми для Windows.Вот пример простейшей программы для Windows.

```
#include <windows.h>
int WINAPI WinMain(
  HINSTANCE hInstance
  HINSTANCE hPrevInstance
  LPSTR lpCmdLine
  int nCmdShow){int nCmdShow) {
   return TRUE;return TRUE;
}
```

Природно, призначення функції winMain набагато ширше, ніж просто завершення програми. Тому розглянемо, що необхідно виконати у функції winMain для створення додатків, що підтримують однодокументний (SDI), багатодокументний (MDI) інтерфейс або інтерфейс на базі вікна діалогу — основних типів додатків для Windows. У лістингу 1 приведена стандартна, хоча і трохи усічена реалізація головної функції Windows -приложения.Естественно, назначение функции winMain гораздо шире, чем просто завершение программы. Поэтому рассмотрим, что необходимо проделать в функции winMain для создания приложений, поддерживающих однодокументный (SDI), многодокументный (MDI) интерфейс или интерфейс на базе окна диалога — основных типов приложений для Windows. В листинге 1 приведена стандартная, хотя и немного усеченная реализация главной функции Windows-приложения.

Лістинг 1Листинг 1

```
int WINAPI WinMain(
  HINSTANCE hInstance
  HINSTANCE hPrevInstance
  LPSTR lpCmdLine
  int nCmdShow){
  MSG msg;MSG msg;
   /* хоча параметр hPrevInstance в Win32 завжди рівний NULL, продовжуємо дляхотя
параметр hPrevInstance в Win32 всегда равен NULL, продолжаем для
      сумісності перевіряти його значення */
   if (!hPrevInstance){if (!hPrevInstance) {
      /* ініціалізували додаток - готуємо дані класу вікна
         і реєструємо його */ и регистрируем его */
      if ( !InitApplication (hInstance))
         return (FALSE);return (FALSE);
   }    /* завершуємо створення копії додатка -
           створюємо головне вікно додатка */создаем главное окно приложения */
   if ( !InitInstance (hInstance, nCmdShow))
      return (FALSE);return (FALSE) ;
   /*  Цикл обробки  повідомлень   */
   while (GetMes'sage (&msg,   NULL,   0,   0)){
     TranslateMessage (&msg);
     DispatchMessage (&msg);
   }
   return (msg.wParam);
}
```

Функція складається з трьох основних функціональних частин (малюнок 1) : реєстрації класу вікна, створення головного вікна додатка і циклу обробки повідомлень. Використовувані функції InitApplication і InitInstance — не системні, вони мають бути написані розробником програми.

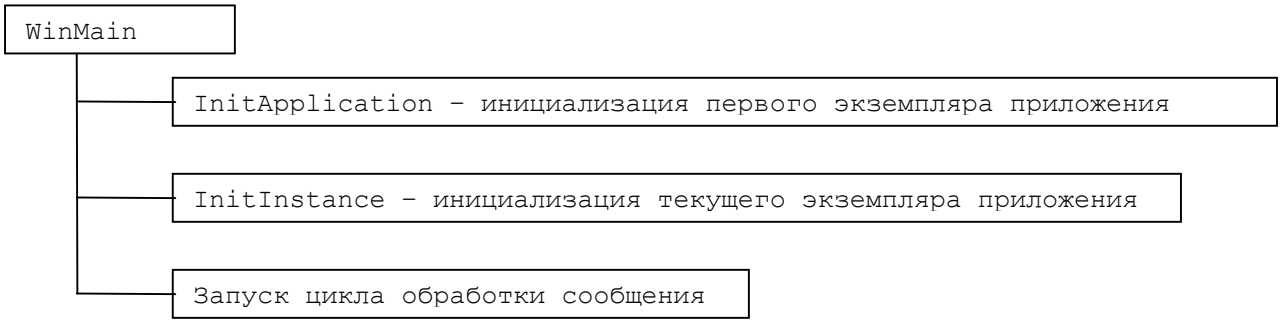


Рисунок 1 – Упрощенная структура функции WinMain

Регистрация класса окна

Будь-яке вікно Windows належить одному з існуючих в даний момент в системі класів, який має бути визначений до того, як вікно буде відображено на екрані. Клас вікна задає найбільш загальні властивості вікон, наприклад, форму покажчика миші при переміщенні його в зону вікна або ім'я меню, визначеного для вікон цього класу. Іншими словами, клас вікна — це шаблон, в якому визначаються вибрані стилі, шрифти, заголовки, піктограми, розмір, розташування вікна і так далі. Оскільки кожен клас має свою структуру параметрів вікна, доступну усім, не відбувається непотрібного дублювання даних. При цьому слід враховувати, що ім'я класу вікна має бути унікальним, щоб не виникало конфліктів з класами вікон інших застосувань. Крім того, два вікна одного і того ж класу використовують загальну функцію вікна (віконну процедуру) з усіма її допоміжними функціями.Любое окно Windows принадлежит одному из существующих в данный момент в системе классов, который должен быть определен до того, как окно будет отображено на экране. Класс окна задает наиболее общие свойства окон, например, форму указателя мыши при перемещении его в область окна или имя меню, определенного для окон этого класса. Другими словами, класс окна — это шаблон, в котором определяются выбранные стили, шрифты, заголовки, пиктограммы, размер, расположение окна и т.д. Поскольку каждый класс имеет свою структуру параметров окна, доступную всем, не происходит ненужного дублирования данных. При этом следует учитывать, что имя класса окна должно быть уникальным, чтобы не возникало конфликтов с классами окон других приложений. Кроме того, два окна одного и того же класса используют общую функцию окна (оконную процедуру) со всеми ее вспомогательными функциями.

```
BOOL InitApplication(HINSTANCE hlnstance){
  WNDCLASS  wc;
  // Будь-які необхідні дії з ініціалізації додатка
  // Заповнюємо структуру WNDCLASS класу вікнаЗаполняем структуру WNDCLASS класса
окна
  wc.style         = CS_HREDRAW | CS_VREDRAW;wc.style         = CS_HREDRAW |
CS_VREDRAW;
  wc.lpfnWndProc   = (WNDPROC) WndProc;wc.lpfnWndProc   = (WNDPROC)WndProc;
  wc.cbClsExtra    = 0;wc.cbClsExtra    = 0;
  wc.cbWndExtra    = 0;
  wc.hInstance     = hInstance;
  wc.hIcon         = LoadIcon (hlnstance, IDI_APLICATION);
  wc.hCursor       = LoadCursor (NULL, IDC ARROW);
  wc.hbrBackground = (HBRUSH)  (COLOR_WINDOW + 1);
  wc.lpszMenuName  = NULL;
  wc.lpszClassName = szAppName;
  // Реєструємо клас вікна
  return RegisterClass (&wc);
}
```

У Win32 для реєстрації класу вікна використовуються два типи структур : звична WNDCLASS і нова WNDCLASSEX, які відрізняються двома полями — cbSize (визначає розмір структури у байтах і обчислюється як sizeof (WNDCLASS)) і hIconSm {задає ідентифікатор маленької піктограми, яка відображатиметься в області системного меню; у Windows NT це поле не використовується і має бути встановлене в NULL).

Введення нової структури доки не привело до її широкого використання і частіше як і раніше використовується структура WNDCLASS, що вважається застарілою, оголошення якої розташовується у файлі winuser.**h**:В Win32 для регистрации класса окна используются два типа структур: привычная WNDCLASS и новая WNDCLASSEX, которые отличаются двумя полями — cbSize (определяет размер структуры в байтах и вычисляется как sizeof (WNDCLASS)) и hIconSm {задает идентификатор маленькой пиктограммы, которая будет отображаться в области системного меню; в Windows NT это поле не используется и должно быть установлено в NULL). Введение новой структуры пока не привело к ее широкому использованию и чаще по-прежнему используется считающаяся устаревшей структура WNDCLASS, объявление которой располагается в файле **winuser.h**:

```
typedef struct tagWNDCLASSA {
  UINT style;
  WMDPROC lpfnWndProc;
  int cbClsExtra;
  int cbWndExtra;
  HINSTANCE hInstance;
  HICON hIcon;
  HCURSOR hCursor;
  HBRUSH hbrDackgrcund;
  LPCSTR lpszMeriuName;
  LPCSTR lpszClassName;
} WNDCLASS;WNDCLASS;
```

Розглянемо поля цієї структури :

- lpszClassName — покажчик на рядок, що містить ім'я класу. Оскільки після реєстрації клас стає доступний усім іншим додаткам Windows, ім'я класу має бути унікальним;lpszClassName — указатель на строку, содержащую имя класса. Поскольку после регистрации класс становится доступен всем другим приложениям Windows, имя класса должно быть уникальным;
- lpfnWndProc — визначає адреса функції вікна, яка обробляє повідомлення для вікон цього класу.lpfnWndProc — определяет адрес функции окна, которая обрабатывает сообщения для окон данного класса.
- hInstance — повідомляє Windows про того, хто створює визначення класу. Це необхідно для службових дій усередині Windows. Коли завершується останній екземпляр програми, Windows видаляє усі пов'язані визначення класів.hInstance — сообщает Windows о том, кто создает определение класса. Это необходимо для служебных действий внутри Windows. Когда завершается последний экземпляр программы, Windows удаляет все связанные определения классов.
- cbClsExtra — задає число байт, яке необхідно додатково запросити у Windows під цю структуру для зберігання власних даних, приєднаних до класу.cbClsExtra — задает число байт, которое необходимо дополнительно запросить у Windows под эту структуру для хранения собственных данных, присоединенных к классу.
- cbWndExtra — задає число байт, яке необхідно додатково запросити у Windows для розміщення усіх структур, що створюються спільно з цим класом для зберігання власних даних, приєднаних до вікна.cbWndExtra — задает число байт, которое необходимо дополнительно запросить у Windows для размещения всех структур, создаваемых совместно с данным классом для хранения собственных данных, присоединенных к окну.
- hIcon — визначає піктограму, яка використовуватиметься для зображення додатка, наприклад, на панелі завдань. Ви можете створити піктограму самі або скористатися якою-небудь із зумовлених:hIcon — определяет пиктограмму, которая будет использоваться для изображения приложения, например, на панели задач. Вы можете создать пиктограмму сами или воспользоваться какой-либо из предопределенных:

IDI_APPLICATION	Стандартна піктограма для додатка
IDI_HAND	Піктограма "знак стоп"
IDI_QYESTION	Піктограма "знак питання"
IDI_EXCLAMATION	Піктограма "знак оклику"
IDI_ASTERISK	Піктограма "інформація"

- hCursor — ідентифікує курсор миші за умовчанням, використовуваний r цьому вікні. Якщо немає бажання створювати свій курсор, то можна скористатися одним з

тих, що надаються системою;hCursor — идентифицирует курсор мыши по умолчанию, используемый r данном окне. Если нет желания создавать свой курсор, то можно воспользоваться одним из предоставляемых системой;

IDC_ARROW	Стрілка
IDC_IBEAM	Вертикальна риса
IDC_WAIT	"пісочний годинник"
IDC_CRQSS	Перехрестя
IDC_UPARROW	Вертикальна стрілка
IDC_SIZE	Чотири стрілки, що вказують в різні боки
IDC_ICON	Курсор, використовуваний при перенесенні файлів
IDC_SIZENWSE	Двонаправлена стрілка північний захід/південний схід
IDC_5IZENESW	Двонаправлена стрілка північний схід/південний захід
IDC_SIZEW5	Двонаправлена стрілка схід-захід
IDC_SIZENS	Двонаправлена стрілка північ-південь

- hbrBackground — загавкає цьет фону для вікна. Для його визначення можна скористатися будь-якою з 20 системних констант кольори, які описані у файлі winuser.h таким чином:hbrBackground — залает цьет фона для окна. Для его определения можно воспользоваться любой из 20 системных констант цвета, которые описаны в файле winuser.h следующим образом:

```
#define COLOR_WINDOW         0
#define COLOR_DACKGROUN D    1
#define COLOR_APPWORKSPACE   12define COLOR_APPWORKSPACE   12
```

При використанні цих констант до їх значення обов'язково треба додавати одиницю, оскільки перше значення для них дорівнює нулю, а нуль є недійсним значенням для логічного номера кисті. Крім того, необхідно виконати приведення твань НЕШиЯН, інакше компілятор видасть помилку.При использовании этих констант к их значению обязательно нужно добавлять единицу, т.к. первое значение для них равно нулю, а ноль является недействительным значением для логического номера кисти. Кроме того, необходимо выполнить приведение к тину НЕШиЯН, иначе компилятор выдаст ошибку.

- lpszMenuNaiae — покажчик на ім'я меню вікна, визначене у файлі ресурсівlpszMenuNaiae — указатель на имя меню окна, определенное в файле ресурсов,
- style — визначає властивості вікна, які можуть комбінуватися за допомогою оператора | (ЧИ). При програмуванні для Windows цей термін відноситься до набору параметрів, кожен з яких управляється одним або двома бітами. Якщо цьому полю присвоїти значення NULL, то Windows автоматично встановить значення за умовчанням. Вікно є основою додатка і длл того, щоб більше не повертатися до цього питання, приведемо тут список допустимих стилів :style — определяет свойства окна, которые могут комбинироваться при помощи оператора | (ИЛИ). При программировании для Windows этот термин относится к набору параметров, каждый из которых управляется одним или двумя битами. Если этому полю присвоить значение NULL, то Windows автоматически установит значения по умолчанию. Окно является основой приложения и длл того, чтобы больше не возвращаться к этому вопросу, приведем здесь список допустимых стилей:

Таблиця

CS_BYTEALXGNCLIENT	Робить вплив на ширину вікна, задаючи вирівнювання по межі байт клієнтської зони вікна. Таке вирівнювання дозволяє збільшити продуктивність операцій малюванні у вікні
CS_BYTEALIGNWINDOW	Встановлює вирівнювання по межі байт вікна. Таке вирівнювання дозволяє збільшити продуктивність деяких типів операцій, таких як переміщення і зміну розмірів вікна, рисбвание пунктів меню
CS_CLAS3DC	Забезпечує контекст пристрою, використовуваний спільно усіма вікнами в класі
CS_DBLCLKS	Служить для установки таймера після отримання першого повідомлення про натиснення кнопки миші. Повідомлення про подвійне натиснення буде сформовано тільки у тому випадку, якщо друге натиснення станеться до витікання певного часу таймера. Забезпечує усім вікнам класу можливість сприймати повідомлення про подвійне натиснення на одну з кнопок миші
CS_CLOBALCLASS	Дозволяє додатку створювати вікно цього класу, не зважаючи на значення параметра hlnstance. Якщо ви не визначите цей стиль,

	то параметр hlnstance, що передається у функції створення вікна, має бути таким же, який передається у функцію RegisterClas :>. Іншими словами, установка цього стилю дозволяє створювати віконні класи, призначені для спільного використання декількома програмами
CS_HREDRAW	Визначає, що вікно перемальовуватиметься, як тільки зміниться його горизонтальний розмір
CS_VREDRAW	Визначає, що вікно перемальовуватиметься, як тільки зміниться його вертикальний розмір
CS_NOCLOSE	Забороняє пункт Close (Закрити) системного меню. Цей стиль слід використати з вікнами, що мають системне меню, які не можуть бути закриті користувачем
CS_OWNDC	Надає приватний контекст пристрою для кожного вікна в цьому класі. Цей тип контексту пристрою найбільш марнотратний в сенсі використання пам'яті, проте забезпечує найвищу швидкість реакції
CS_PARENTDC	Встановлює, що дочірні вікна не можуть малювати у своєму батьківському вікні. Цей стиль зазвичай застосовується для заздалегідь певних класів, необхідних для створення елементів управління блоків діалогу
CS_SAVEBITS	Пропонує системі зберігати копію частини екрану, пошкоджене вікном. Після видалення вікна пошкоджена частина може бути відновлена. При цьому Windows не посилає повідомлення m paint для перемальовування області, яку займало вікно. Цей стиль застосовується для невеликих вікон (наприклад, меню або блоків діалогу), які з'являються на екрані на короткі проміжки часу

Після того, як клас вікна зареєстрований, можна переходити безпосередньо до створення головного вікна додатка.После того как класс окна зарегистрирован, можно переходить непосредственно к созданию главного окна приложения.

Створення вікна

Наступна функція з лістингу 1, яку необхідно реалізувати самостійно — це initinstance. Її основне призначення — створення вікна :Следующая функция из листинга 1, которую необходимо реализовать самостоятельно — это initinstance. Ее основное назначение — создание окна:

```
   LPCSTR szClassName = "WinAPI";LPCSTR szClassName = "WinAPI";
   LPCSTR szTitle =    "Створення вікна Windows";LPCSTR szTitle =    "Создание окна
Windows";
   BOOL Initinstance(HINSTANCE hlnstance, int nCmdShow){
      HWND hWnd;HWND hWnd;
      hWnd = CreateWindow(hWnd = CreateWindow(
         szClassName,  // покажчик на рядок зареєстрованогоszClassName,  // указатель
на строку зарегистрированного
                       // імені класу
         szTitle,       // покажчик на рядок заголовка вікнаszTitle,       // указатель
на строку заголовка окна
         WS_OVERLAPPEDWINDOW, // стиль вікнаWS_OVERLAPPEDWINDOW, // стиль окна
         CW_USEDEFAULT, // горизонтальна координата вікнаCW_USEDEFAULT, //
горизонтальная координата окна
         CW_USEDEFAULT, // вертикальна координата вікнаCW_USEDEFAULT, // вертикальная
координата окна
         CW_USEDEFAULT, // ширина вікнаCW_USEDEFAULT, // ширина окна
         CW_USEDEFAULT, // висота вікнаCW_USEDEFAULT, // высота окна
         NULL,           // дескриптор батьківського вікнаNULL,           // дескриптор
родительского окна
         NULL,           // дескриптор меню вікнаNULL,           // дескриптор меню
окна
         hlnstance,     // дескриптор екземпляра додаткаhlnstance,     // дескриптор
экземпляра приложения
```

```
        NULL);          // покажчик на додаткові ці вікнаNULL);          // указатель
на дополнительные данные окна
       if (ihWnd)
          return (FALSE);
       ShowWindow{hWnd, nCmdShow);
       UpdateWindow(hWnd);UpdateWindow(hWnd);
        return (TRUE);return (TRUE);
  }
```

Перші два параметри — ім'я класу і заголовок — дозволяють системі і користувачеві ідентифікувати вікно. Параметры з четвертого по сьомій задають початкове положення і розміри вікна. Константа CWJJSEDEFAULT пропонує Windows самій вибрати ці параметри. Це значення використовується найчастіше, але якщо потрібні деякі конкретні величини, то можна вказати їх тут. Наступні три параметри визначають дескриптори відповідно батьківського вікна, меню і самого застосування. Останній (одинадцятий) параметр дозволяє асоціювати з вікном деякі додаткові дані. Функція возврпша-сг дескриптор створеного вікна Windows.Первые два параметра — имя класса и заголовок — позволяют системе и пользователю идентифицировать окно. Параметры с четвертого по седьмой задают начальное положение и размеры окна. Константа CWJJSEDEFAULT предписывает Windows самой выбрать эти параметры. Это значение используется наиболее часто, но если необходимы некоторые конкретные величины, то можно указать их здесь. Следующие три параметра определяют дескрипторы соответственно родительского окна, меню и самого приложения. Последний (одиннадцатый) параметр позволяет ассоциировать с окном некоторые дополнительные данные. Функция возврпша-сг дескриптор созданного окна Windows.

Після повернення з функції Createwindow система Windows записує і свою внутрішню базу даних інформацію, необхідну для супроводу цього конкретного вікна. Але при цьому вікно на екрані не з'являється— потрібно виклик двох функцій, що залишилися, завдяки яким повністю оформлене вікно з'являється на екрані :После возврата из функции Createwindow система Windows записывает и свою внутреннюю базу данных информацию, необходимую для сопровождения данного конкретного окна. Но при этом окно на экране не появляется— требуется вызов оставшихся двух функций, благодаря которым полностью оформленное окно появляется на экране:

```
ShowWindow(hWnd, nCmdShow);
UpdateWindow(hWnd);
```

Функція ShowWindow передає в Windows інформацію про те, в якому виді необхідно відобразити вікно: згорнутому, звичайному або розгорнутому, що визначається параметром nCmdShow. Функція UpdateWindow пропонує Windows послати вікну повідомлення WM PAINT, щоб мати можливість перемальовувати вікно.Функция ShowWindow передает в Windows информацию о том, в каком виде необходимо отобразить окно: свернутом, обычном или развернутом, что определяется параметром nCmdShow. Функция UpdateWindow предписывает Windows послать окну сообщение WM PAINT, чтобы иметь возможность перерисовать окно.

Розглянута послідовність дій при створенні вікна типова для додатків Windows.Рассмотренная последовательность действий при создании окна типична для приложений Windows.

Проте залишився ще один (третій) параметр — стиль вікна, який потрібно розглянути детальніше.Однако остался еще один (третий) параметр — стиль окна, который надо рассмотреть более подробно.

Вікна, визначені в Windows

Існує всього три основні типи вікон — *що перекриваються, спливаючі \ дочірні,* з яких програміст може створювати безліч найрізноманітніших об'єктів, комбінуючи зумовлені біти стилю. Розглянемо ці біти, їх вплив на зовнішній вигляд і деякі властивості вікон (таблиця 2) :Существует всего три основных типа окон — *перекрывающиеся, всплывающие \\ дочерние,* из которых программист может создавать множество самых разнообразных объектов, комбинируя предопределенные биты стиля. Рассмотрим эти биты, их влияние на внешний вид и некоторые свойства окон (таблица 2):

Таблиця 2 - Біти стилю

Константа	Значення
WS_BORDER	Вікно має рамку без заголовка
WS_CAPTION	Вікно має заголовок і рамку. Як правило, цей стиль використовується для вікон, що перекриваються, і не може застосовуватися спільно із стилем WS_DLGFRAMF
WS_CHILD	Створюване вікно є дочірнім. Не може використовуватися спільно із стилем WS_POPUP
WS_CLIPCHILDREN	Виключає область, зайняту дочірнім вікном з області малювання батьківського вікна; використовується тільки для батьківських вікон
WS_CLIPSIBLINGS	Виключає усі інші дочірні вікна зі своєї області малювання. Іншими словами, якщо дочірні вікна перекриваються, а цей стиль не вказаний, то при зміні робочої області одного з вікон можуть бути зіпсовані робочі області інших дочірніх вікон. Цей стиль використовується тільки разом із стилем WS_CHILD
WS_DISABLED	Створюється неактивне вікно, тобто відразу після створення воно недоступне для введення з клавіатури або за допомогою миші
WS_DLGFRAME	Вікно має подвійну рамку і не має заголовка
WS_GROUP	Визначає перший елемент управління групи вікон, до яких користувач може переходити за допомогою клавіш із стрілками. Усі елементи управління, визначені з цим стилем, закінчують поточну і починають нову групу (одна група закінчується там, де починається інша)
WS_HSCROLL	Вікно має горизонтальну смугу прокрутки
WS_VSCROLL	Вікно має вертикальну смугу прокрутки
WS_MAXIMINE	Створюване вікно при відображенні матиме максимально можливий для нього розмір
WS_MAXIMIZEB0X	Створюване вікно матиме кнопку максимізації. Якщо вікно являється дочірньому вікном елементу управління, то цей прапор стилю використовується під іншим ім'ям — WS_TABSTOP
WS_MINIMIZE	Створюване вікно буде відображено у вигляді піктограми. Використовується тільки із стилем WS_OVERLAPPED
WS_MINIMIZEBOX	Створюване вікно має кнопку мінімізації. Якщо вікно є дочірнім вікном елементу управління, то цей прапор стилю використовується під іншим ім'ям — WS_GROUP
WS_OVERLAPPED	Створюване вікно є таким, що перекривається. Зазвичай має заголовок і рамку
WS_POPUP	Створюємося спливаюче вікно. Не може використовуватися спільно із стилем WS_CHILD
WS_SYSMENU	Створюване вікно имеот системне меню в смузі заголовка
WS_TABSTOP	Визначається для одного або декількох елементів управління для того, щоб між ними можна було переміщатися використовуючи клавішу табуляції (<TAB>)
WS_THICKFRAME	Створюване вікно має потовщену рамку, за допомогою якої можна змінювати розмір вікна
WS_VISIBLE	Вікно стає видимим відразу після створення. Цей стиль в основному використовується для вікон діалогу

Окрім перерахованих стилів вікон Windows, у файлі **winuscr.h** визначені їх комбінації (таблиця 3) :Помимо перечисленных стилей окон Windows, в файле **winuscr.h** определены их комбинации (таблица 3):

Таблиця 3

Константа	Значення
WS_OVERLAPPEDWINDOW	Комбінація стилів WS_OVKRLAPPFD, WS_CAPTION, WS_SYSMKNU, WS_THICKFKAME, WS_MAXIMIZFHGX і WS_MINIMSПЮХ
WS_POPUPWINDOW	Комбінація стилів WS_POPUP, WS_BORDER і WS_SYSMENU

При роботі з вікнами часто також використовуються додаткові або розширені біти стилю :При работе с окнами часто также используются дополнительные или расширенные биты стиля:

Таблиця 4 - Розширені біти стилю

Константа	Значення
WS_EX_ABSPOSITION	Визначає, що позиція вікна задається в абсолютних одиницях, відносно лівого верхнього утла екрану
WS_EX_ACCEPTFILES	Визначає, що вікно допускає перетягання (drag - and - drop) файлів
WS_EX_CLIENTEDGE	Для надання "тривимірного ефекту" межі вікна, клієнтська область якого як би втиснута у вікно
WS_EX_CONTEXTHELP	Включає знак питання в заголовок вікна; коли користувач натискає на цей знак, то курсор змінюється на покажчик у вигляді знаку питання; якщо тепер вибрати яке-небудь дочірнє вікно — воно отримає повідомлення WM_HELP
WS_EX_CONTROLPAKENT	Дозволяє користувачеві переміщатися по дочірніх вікнах батьківського вікна, використовуючи для цього клавішу табуляції (<TAB>)
WS_EX_DLGMODALFRAME	Створюване вікно матиме подвоєну межу, що спільно із стилем ws caption дозволяє створювати вікно із смугою заголовка
WS_EX_LEFT	Встановлює для вікна властивість лівого вирівнювання (за умовчанням)
WS_EX_LEFTSCROLLBAR	Якщо є смуга прокрутки, то вона розташовується з лівого боку робочої (клієнтською) зони вікна
WS_EX_LTRREADING	Цей стиль використовується за умовчанням і визначає природний порядок (зліва направо) для більшості мов введення і читання тексту
WS_EX_MDICHILD	Створюється дочірнє вікно MDI (вікно багатодокументного інтерфейсу)
WS_EX_NOPARENTNOTIFY	Визначає, що дочірнє вікно, створене з цим стилем, не посилає повідомлення WM_PARENTNOTIFY своєму батьківському вікну при створенні або руйнуванні
WS_EX_RIGHT	Залежно від класу вікна встановлює для нього групову властивість правого вирівнювання
WS_EX_RIGHTSCROLLRAR	Якщо є смуга прокрутки, то вона розташовується з правого боку робочої (клієнтською) зони вікна. Цей стиль використовується за умовчанням
WS_EX_RTLREADING	Визначає порядок введення і читання тексту справа наліво
WS_EX_SMCAPTION	Створюване вікно має зменшену висоту смуги заголовка; використовується при створенні "плаваючих" панелей інструментів
WS_EX_STATICEDGE	Створюване вікно має тривимірний стиль межі, заданий для використання в елементах, до яких користувач не дістає доступу (статичні елементи управління)
WS_EX_TOOLWINDOW	Створюється вікно інструментів, яке використовується для створення "плаваючої" панелі інструментів; таке вікно має укорочену смугу заголовка і не відображається на панелі завдань
WS_EX_TOPMOST	Визначає, що вікно розташовуватиметься поверх усіх вікон (створених без цього стилю). Для установки або видалення цього атрибуту можна використати функцію SetWindowPos
WS_EX_TRANSPARENT	Вказує, що вікно є прозорим, тобто не закриває інші вікна, розташовані під ним. Повідомлення WM_PAINT таке вікно отримує до того, як це повідомлення отримують усі вікна того ж рівня, але розташованих під ним
WS_EX_WINDOWEDGE	Визначає, що для надання вікну "тривимірного ефекту" воно має підведену межу

Як і у разі основних стилів, для розширених визначені два комбіновані додаткові стилі **(winuser.h)** :Как и в случае основных стилей, для расширенных определены два комбинированных дополнительных стиля **(winuser.h):**

Таблиця 4 - Додаткові комбіновані стилі вікон

Константа	Значення
WS_EX_OVERIAPPEDWINDOW	Комбінація стилів WS_EX_CLIENTEDGE і WS_EX_WTNDOWEDGE
WS_EX_PALETTEWINDOW	Комбінація стилів WS_EX_WlNDOWEDGE, W3_EX_SMCAPTION і WS_EX_TOPMOST

Розглянемо основні типи вікон Windows. До них відносяться наступні.Рассмотрим основные типы окон Windows. К ним относятся следующие.

1. Вікна (overlapped window), **що перекриваються.** Це основний, найбільш універсальний тип вікон Windows, Для їх створення найчастіше використовується стиль WS_OVERLAPPEDWINDOW. Головне вікно додатка, як правило, має саме цей тип.**Перекрывающиеся окна** (overlapped window). Это основной, наиболее универсальный тип окон Windows, Для их создания чаще всего используется стиль WS_OVERLAPPEDWINDOW. Главное окно приложения, как правило, имеет именно этот тип.

2. Допоміжні або спливаючі вікна (popup window). Частіше увесь цей тип вікон створюється з використанням стилю WS_POPUP. Зазвичай вони використовуються для відображення якої-небудь інформації на короткий проміжок часу. Найчастіше вікна цього типу використовуються для отображеиия діалогових вікон або вікон повідомлень. Основна їх відмінність від інших вікон полягає в тому, що навіть якщо вони мають батьківське, то все одно завжди відображаються поверх усіх вікон на екрані, вискакуючи, як поплавці, вгору навіть тоді, коли користувач робить активним інше вікно. Для вікон цього типу, як правило, організовується своя віконна процедура. Вони можуть і не мати батька. Якщо таке вікно не має батьківського, то воно є абсолютно незалежним від вікна, що створило його, і за своїми властивостями практично невідмітно від вікон, що перекриваються. Поведінка спливаючого вікна, що має батька, залежить від того, що відбувається з останнім. Коли головне вікно мінімізується, спливаюче вікно "без батька" ховається, а "з батьком" залишається на екрані згори. Підкреслимо ще один важливий момент: коли ми говоримо, що допоміжне вікно має батька, це зовсім не означає, що воно є дочірнім.**Вспомогательные или всплывающие окна** (popup window). Чаще всего этот тип окон создается с использованием стиля WS_POPUP. Обычно они используются для отображения какой-либо информации на короткий промежуток времени. Наиболее часто окна этого типа используются для отображеиия диалоговых окон или окон сообщений. Основное их отличие от других окон заключается в том, что даже если они имеют родительское, то все равно всегда отображаются поверх всех окон на экране, выскакивая, как поплавки, наверх даже тогда, когда пользователь делает активным другое окно. Для окон этого типа, как правило, организуется своя оконная процедура. Они могут и не иметь родителя. Если такое окно не имеет родительского, то оно является совершенно независимым от создавшего его окна и по своим свойствам практически неотличимо от перекрывающихся окон. Поведение всплывающего окна, имеющего родителя, зависит от того, что происходит с последним. Когда главное окно минимизируется, всплывающее окно "без родителя" скрывается, а "с родителем" остается на экране сверху. Подчеркнем еще один важный момент: когда мы говорим, что вспомогательное окно имеет родителя, это совсем не означает, что оно является дочерним.

3. Дочірні вікна (child window). Вікна цього типу створюються тоді, коли у додатка вже є головне вікно. Такі вікна пов'язані деякими характеристиками (як би підлеглі) з тим вікном, з якого вони були створені, звідси і назва. Призначення цих вікон може бути найрізноманітнішим, починаючи від простого ділення батьківського вікна на області до організації багатодокументного інтерфейсу. Усі елементи управління також є дочірніми вікнами. З основних властивостей цього типу вікон відмітимо наступні. Дочірні вікна ніколи не відображаються поза своїм батьківським вікном ні в розкритому виді, ні у вигляді піктограми — вони як би цілком належать батьку. Розташовуються вони у батьківському вікні відносно верхнього лівого кута його робочої (клієнтською) області. Більше того, при переміщенні батьківського вікна по екрану його дочірні вікна переміщаються разом з ним. І, нарешті, дочірнє вікно ніколи не може стати активним.**Дочерние окна** (child window). Окна этого типа создаются тогда, когда у приложения уже есть главное окно. Такие окна связаны некоторыми характеристиками (как бы подчинены) с

тем окном, из которого они были созданы, отсюда и название. Назначение этих окон может быть самым разнообразным, начиная от простого деления родительского окна на области до организации многодокументного интерфейса. Все элементы управления также являются дочерними окнами. Из основных свойств этого типа окон отметим следующие. Дочерние окна никогда не отображаются вне своего родительского окна ни в раскрытом виде, ни в виде пиктограммы — они как бы целиком принадлежат родителю. Располагаются они в родительском окне относительно верхнего левого угла его рабочей (клиентской) области. Более того, при перемещении родительского окна по экрану его дочерние окна перемещаются вместе с ним. И, наконец, дочернее окно никогда не может стать активным.

Стандартний цикл обробки повідомлень

Частина функції winMain, що залишилася, — стандартний цикл обробки повідомлень — має наступний вигляд:Оставшаяся часть функции winMain — стандартный цикл обработки сообщений — имеет следующий вид:

```
while (GetMessage(&msg, NULL, 0, 0)){
  TranslateMessage (&msg);
  DispatchMessage (&msg);DispatchMessage (&msg);
}
```

Цей цикл працює впродовж усього часу виконання програми. Кожна ітерація представляє прийом одного повідомлення з буфера повідомлень поточного потоку. За це відповідає функція GetMessage, другий аргумент якої (NULL) говорить про те, що слід обробляти усі повідомлення. Коли повідомлення вибирається з черги, воно поміщається в спеціальну структуру MSG, яка виглядає таким чином:Данный цикл работает в течение всего времени выполнения программы. Каждая итерация представляет прием одного сообщения из буфера сообщений текущего потока. За это отвечает функция GetMessage, второй аргумент которой (NULL) говорит о том, что следует обрабатывать все сообщения. Когда сообщение выбирается из очереди, оно помещается в специальную структуру MSG, которая выглядит следующим образом:

```
typedef struct tagMSG {
   HWND gwnd;
  UINT  message;
  WPARAM wParam;
  LPARAM lParam;
  DWORD time;
  POINT pt;
} MSG;
```

Розглянемо кожного з шести елементів структури :

hwnd — містить дескриптор (логічний номер) вікна, чия віконна процедура отримує повідомлення; message — визначає номер повідомлення; wParam і **lParam** — визначають додаткову інформацію про повідомлення. Конкретний зміст цих параметрів залежить від типу повідомлення; **time** і **pt** — описують стан системи у момент постановки повідомлення в чергу: **time** зберігає час, а **pt** — позицію покажчика миші у цей момент.**hwnd** — содержит дескриптор (логический номер) окна, чья оконная процедура получает сообщение; message — определяет номер сообщения; wParam и **lParam** — определяют дополнительную информацию о сообщении. Конкретное содержание этих параметров зависит от типа сообщения; **time** и **pt** — описывают состояние системы в момент постановки сообщения в очередь: **time** хранит время, а **pt** — позицию указателя мыши в этот момент.

Після того, як повідомлення узяте з черги повідомлень, воно передається у функцію **TranslateMessage**, яка викликає драйвер клавіатури Windows для перетворення віртуальних кодів клавіш в ASCII -значения, які ставляться в чергу програмних подій у вигляді повідомлення **WM_CHAR**. Це дозволяє програмі відрізнити, наприклад, "A" від "a" без аналізу стану клавіші регістра.После того как сообщение взято из очереди сообщений, оно передается в функцию **TranslateMessage**, которая вызывает драйвер клавиатуры Windows для преобразования виртуальных кодов клавиш в ASCII-значения, которые ставятся в очередь программных событий в виде сообщения **WM_CHAR**. Это позволяет программе отличить, например, "A" от "a" без анализа состояния клавиши регистра.

Остання функція циклу — **DipatchMessage** бере дані про повідомлення із структури **msg** і передасть їх у відповідну віконну процедуру для обробки. Після того, як повідомлення передане, знову викликається функція **GetMessage**, щоб узяти з черги наступне повідомлення, якщо таке є. За винятком **WM_QUIT**, кожне повідомлення примушує **GetMessage** повернути значення **TRUE**. Приймаючи повідомлення **WM_QUIT**, програма виходить з циклу обробки повідомлень і завершує роботу.Последняя функция цикла — **DipatchMessage** берет данные о сообщении из структуры **msg** и передаст их в соответствующую оконную процедуру для обработки. После того как сообщение передано, снова вызывается функция **GetMessage**, чтобы взять из очереди следующее сообщение, если таковое имеется. За исключением **WM_QUIT**, каждое сообщение заставляет **GetMessage** возвратить значение **TRUE**. Принимая сообщение **WM_QUIT**, программа выходит из цикла обработки сообщений и завершает работу.

Важливо розуміти, що цей, або подібний до нього, цикл без кінця повторюється впродовж усього життя додатка. Він складає суть будь-якої програми, написаної для Windows.Важно понимать, что этот, или подобный ему, цикл без конца повторяется в течение всей жизни приложения. Он составляет суть любой программы, написанной для Windows.

Віконна процедура

Щоб програма могла взаємодіяти з "зовнішнім світом", необхідно це яким-небудь чином забезпечити. У будь-якій Windows -программе роль одержувача повідомлень виконує віконна процедура, якій передаються повідомлення, що поступають з очсреци застосування :Чтобы программа могла взаимодействовать с "внешним миром", необходимо это каким-либо образом обеспечить. В любой Windows-программе роль получателя сообщений выполняет оконная процедура, которой передаются сообщения, поступающие из очсреци приложения:

```
LRESULT CALLBACK  WindowProc (
 HWND  hwnd
 UINT  message
 WPARAM  wParam
 LPARAM  lParam){
   switch (message} {
     case WM_DESTROY :
       PostQuitMessage(0);
       break;
     Default:
       return DefWindowProc (hwnd, message, wParan, lParam);
   }
   return 0;return 0;
}
```

Зазвичай це означає, що повідомлення переходять н віконну процедуру, де і обробляються індивідуально. У простому випадку додаток явно відповідає тільки на повідомлення **WM_DESTROY**, а усі інші передаються в **DefWindowProc** — функцію, що управляє поведінкою вікна за умовчанням. Вона уміє обробляти за умовчанням майже усі види поведінка, що асоціюється з конкретним типом вікна, Наприклад, вона знає як обробляти дії миші, що змінюють форму і місце розташування вікна. Проте функція **DefWindowProc** не займається повідомленням **WM_DESTROY**, тому потрібна його обробка. РядкиОбычно это означает, что сообщения переходят н оконную процедуру, где и обрабатываются индивидуально. В простейшем случае приложение явно отвечает только на сообщение **WM_DESTROY**, а все остальные передаются в **DefWindowProc** — функцию, управляющую поведением окна по умолчанию. Она умеет обрабатывать по умолчанию почти все виды поведения, ассоциированные с конкретным типом окна, Например, она знает как обрабатывать действия мыши, изменяющие форму и местоположение окна. Однако функция **DefWindowProc** не занимается сообщением **WM_DESTROY**, поэтому необходима его обработка. Строки

```
case WM_DESTROY :case WM_DESTROY:
   PostQuitMessage(O);PostQuitMessage(O);
   break;break;
```

є не що інше, як простий обробник повідомлень, які широко використовуються як при написанні традиційних С-программ, так і програм, що базуються на якій-небудь бібліотеці класів, будь то OWL (бібліотека класів компілятора Borland C++) або MFC.представляют собой не что иное, как простейший обработчик сообщений, которые широко используются как при написании традиционных С-программ, так и программ, базирующихся на какой-либо библиотеке классов, будь то OWL (библиотека классов компилятора Borland C++) или MFC.

Опис лабораторної роботи

1. Як і у разі консольного застосування, щоб отримати виконуваний файл, необхідно спочатку створити проект додатка. Visual C++ надає для роботи з Win32 цілей три типи проектів : Win32 Application, Win32 Dinamic - Link Library і Win32 Static Library. Найчастіше використовується Win32 Application.Как и в случае консольного приложения, чтобы получить исполняемый файл, необходимо сначала создать проект приложения. Visual C++ предоставляет для работы с Win32 целей три типа проектов: Win32 Application, Win32 Dinamic-Link Library и Win32 Static Library. Наиболее часто используется Win32 Application.

2. Створіть теку для файлів лабораторної роботи і помістіть в неї файл Win32API.c.Создайте папку для файлов лабораторной работы и поместите в нее файл Win32API.c.

3. Запустіть на виконання середовище розробки Visual C++ 6.0. Виберіть тип додатка Win32 Application, задайте його ними і місце розташування. Для створення файлу проекту натисніть кнопку ОКИ.Запустите на выполнение среду разработки Visual C++ 6.0. Выберите тип приложения Win32 Application, задайте его ими и местоположение. Для создания файла проекта нажмите кнопку ОК.

4. Додайте в проект файл Win32API.c з кодом, представленим в лістингу 2.Добавьте в проект файл Win32API.c с кодом, представленным в листинге 2.

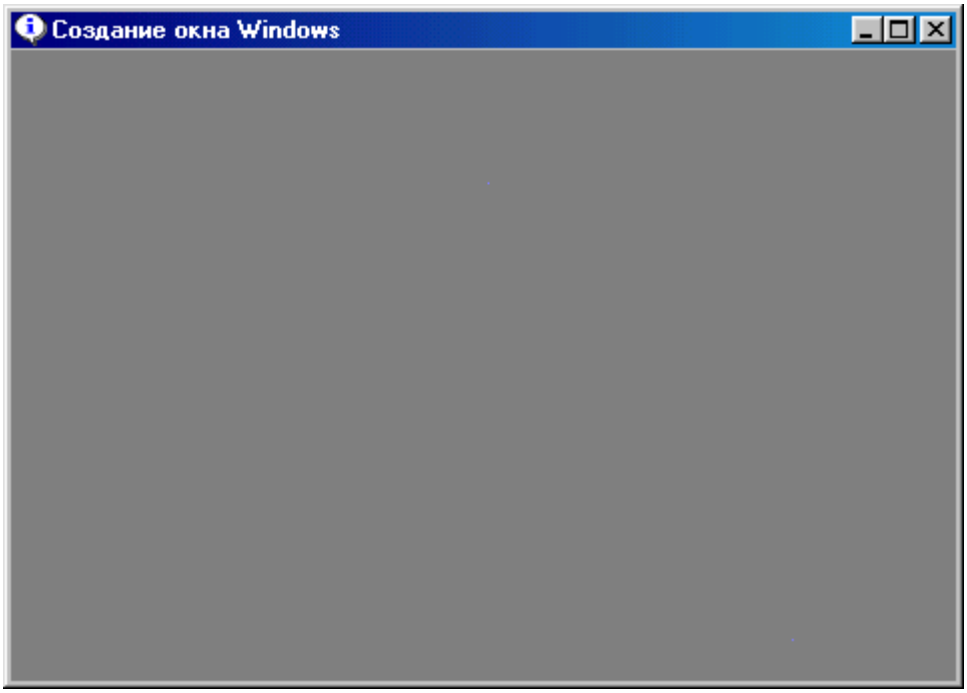
Лістинг 2Листинг 2

```
/*************************************************************
Модуль WinAPIМодуль WinAPI
*************************************************************/
#include <windows.h>

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL InitApplication(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow);

LPCSTR szClassName = "WinAPI";
LPCSTR szTitle =     "Створення вікна Windows";

int WINAPI WinMain(
  HINSTANCE hInstance,
  HINSTANCE hPrevInstance,
  LPSTR lpCmdLine,
  int nCmdShow)
{
  MSG msg;

   /* хоча параметр hPrevInstance в Win32 завжди рівне NULLхотя параметр
hPrevInstance в Win32 всегда равно NULL
      продовжуємо для сумісності перевіряти його значення */
   if (!hPrevInstance)  {if (!hPrevInstance)   {
     /* ініціалізували додаток -
        готуємо дані класу вікна і реєструємо його */
     if (!InitApplication(hInstance))
       return (FALSE);
   }

   /* завершуємо створення копії додатка -
      створюємо головне вікно додатка */
   if (!InitInstance(hInstance, nCmdShow))
```

```
    return (FALSE);

  /* Цикл обробки повідомлень */Цикл обработки сообщений */
  while (GetMessage(&msg, NULL, 0, 0)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
  return (msg.wParam);
}

BOOL InitApplication(HINSTANCE hInstance){
  WNDCLASS  wc;

  // Заповнюємо структуру класу вікна WNDCLASSЗаполняем структуру класса окна
WNDCLASS
  wc.style         = CS_HREDRAW | CS_VREDRAW;
  wc.lpfnWndProc   = (WNDPROC) WndProc;
  wc.cbClsExtra    = 0;
  wc.cbWndExtra    = 0;
  wc.hInstance     = hInstance;
  wc.hIcon         = LoadIcon(NULL, IDI_ASTERISK);
  wc.hCursor       = LoadCursor(NULL, IDC_CROSS);
  wc.hbrBackground = (HBRUSH) (COLOR_APPWORKSPACE+1);
  wc.lpszMenuName  = NULL;
  wc.lpszClassName = szClassName;
  // Реєструємо клас вікна
  return RegisterClass(&wc);
}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow){
  HWND hWnd;HWND hWnd;
  hWnd = CreateWindow(hWnd = CreateWindow(
    szClassName,         // покажчик на рядок зареєстрованогоszClassName,
// указатель на строку зарегистрированного
                         // імені класу
    szTitle,             // покажчик на рядок заголовка вікнаszTitle,
// указатель на строку заголовка окна
    WS_OVERLAPPEDWINDOW, // стиль вікнаWS_OVERLAPPEDWINDOW, // стиль окна
    CW_USEDEFAULT,       // горизонтальна координата вікнаCW_USEDEFAULT,       //
горизонтальная координата окна
    CW_USEDEFAULT,       // вертикальна координата вікнаCW_USEDEFAULT,       //
вертикальная координата окна
    CW_USEDEFAULT,       // ширина вікнаCW_USEDEFAULT,       // ширина окна
    CW_USEDEFAULT,       // висота вікнаCW_USEDEFAULT,       // высота окна
    NULL,                // дескриптор батьківського вікнаNULL,                //
дескриптор родительского окна
    NULL,                // дескриптор меню вікнаNULL,                //
дескриптор меню окна
    hInstance,           // дескриптор екземпляра додаткаhInstance,           //
дескриптор экземпляра приложения
    NULL);               // покажчик на додаткові ці вікнаNULL);               //
указатель на дополнительные данные окна

  if (!hWnd)
    return (FALSE);

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);
  return (TRUE);
}

LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
```

```
    switch(message)  {
      case WM_DESTROY :
        PostQuitMessage(0);
        break;
      default:
        return DefWindowProc(hwnd, message, wParam, lParam);
    }
    return 0;return 0;
  }
```

5 Відкомпілюйте проект і запустіть його на виконання. Зовнішній вигляд головного вікна додатка приведений на малюнку 1.



Малюнок 1 - Головне вікно додатка WinAPIРисунок 1 – Главное окно приложения WinAPI

6. Внесіть зміни в додаток, які гойдаються віконної процедури і представлені в лістингу 3.

Лістинг 3Листинг 3

```
  LRESULT CALLBACK WndProc(HWND hwnd, UINT message
                           WPARAM wParam, LPARAM lParam)
  {
    HDC hDC;
    PAINTSTRUCT ps;
    switch(message)
    {
      case WM_PAINT :
        hDC = BeginPaint(hwnd, &ps);
        TextOut(hDC, 250, 200, "Обробка повідомлення WM_PAINT", 28);TextOut(hDC, 250,
200, "Обработка сообщения WM_PAINT", 28);
        EndPaint(hwnd, &ps);
        break;
      case WM_DESTROY :
        PostQuitMessage(0);
        break;
      default:
        return DefWindowProc(hwnd, message, wParam, lParam);
    }
    return 0;return 0;
  }
```

7. Відкомпілюйте змінене застосування і знову запустіть на виконання. Зробимо декілька зауважень з приводу внесених змін.

Для будь-якого виводу у вікно Window необхідно використати функції графічного виводу, які в якості параметра використовують контекст облаштування виводу. У додатку ідентифікатор контексту отримують за допомогою функції BeginPaint :Для любого вывода в окно Window необходимо использовать функции графического вывода, которые в качестве параметра используют контекст устройства вывода. В приложении идентификатор контекста получают с помощью функции BeginPaint:

```
PBC = BeginPaint (hwnd, &ps);PBC = BeginPaint (hwnd, &ps);
```

де **hwnd** - дескриптор поточного вікна, а **ps** - покажчик на структуру PAINTSTRUCT, описану в заголовному файлі **wingdi.h.**где **hwnd** – дескриптор текущего окна, а **ps** – указатель на структуру PAINTSTRUCT, описанную в заголовочном файле **wingdi.h.**

Виведення інформації у вікно завершується викликом функції EndPaint :Вывод информации в окно завершается вызовом функции EndPaint:

```
EndPaint = (hwnd, &ps);EndPaint = (hwnd, &ps);
```

Для кожного виклику функції **BeginPaint** повинен існувати відповідний виклик функції **EndPaint**.Для каждого вызова функции **BeginPaint** должен существовать соответствующий вызов функции **EndPaint**.

Графічна функція, розташована між цими «дужками», - **TextOut**, відповідає за виведення заданого тексту в певне місце вікна.Графическая функция, расположенная между этими «скобками» - **TextOut**, отвечает за вывод заданного текста в определенное место окна.

Детальніше питання графічного виводу розглядаються в наступній лабораторній роботі.

Порядок виконання роботи

1. Прочитайте опис лабораторної роботи. Відповідно до умов вирішуваної задачі складіть граф-схему алгоритму. Він має бути виконаний акуратно.
2. Підготуйте письмовий звіт і покажіть його викладачеві. Після отримання дозволу приступайте до виконання лабораторної роботи.
3. Скопіюйте у свою теку заготівлі файлів, запустіть Visual C++ і створіть проект.Скопируйте в свою папку заготовки файлов, запустите Visual C++ и создайте проект.
4. Виконайте усі кроки, пов'язані з відладкою і тестуванням програми.
5 Поекспериментуйте із стилями вікон, змінивши відповідні структури в додатку відповідно до даних таблиць
5. Захистіть звіт про виконану лабораторну роботу.Защитите отчет о выполненной лабораторной работе.
6. Закінчіть сеанс роботи з операційною системою.

Звіт повинен містити:

1) Найменування і мета роботи.
2) Тексти програм з необхідними коментарями.
3) Висновки по лабораторній роботі.

Контрольні питання

1. Перерахуйте основні кроки, необхідні для створення Windows - приложения.Перечислите основные шаги, необходимые для создания Windows-приложения.
2. У чому полягають відмінності між дочорнимо і допоміжними вікнами Windows -приложения?В чем заключаются различия между дочерним и вспомогательными окнами Windows-приложения?
3. Поясніть призначення віконної процедури.
4. Перерахуйте усі елементи структури, в яку поміщається повідомлення в Windows -программах.Перечислите все элементы структуры, в которую помещается сообщение в Windows-программах.
5. Для чого використовуються блоки діалогу?
6. Чим визначається реакція Windows -приложения?Чем определяется реакция Windows-приложения?

7. Перерахуйте стандартні ресурси додатка.

8. Назвіть функцію, яка першою отримує управління при запуску Windows - приложения.Назовите функцию, которая первой получает управление при запуске Windows-приложения.

9. Що називається ресурсом додатка?

10. Які властивості задає клас вікна?

11. Назвіть основні процедури функції WinMain.Назовите основные процедуры функции WinMain.

12. Перерахуйте основні елементи структури WNDCLASS.Перечислите основные элементы структуры WNDCLASS.

13. Назвіть декілька стандартних елементів управління, вживаних в Windows - приложениях.Назовите несколько стандартных элементов управления, применяемых в Windows-приложениях.

14. До якого типу вікон належать елементи управління?

Лабораторна робота 4

Збір і відображення інформації про систему

Мета роботи — Вивчити методи управління графічним виводом в операційній системі Windows, основні типи графічних об'єктів, типи контекстів пристроїв, основні елементи графічного інтерфейсу - меню, вікна діалогу, візуальні елементи управління. Навчитися програмувати і відлагоджувати додатки для Windows, що використовують графічний інтерфейс.

Літ.: [1], с. 159-182, 221-274.

Короткий огляд системи Windows

Система Windows створювалася, передусім, як графічна оболонка. Відповідно, головна відмінність від багатьох інших операційних систем — це графічне представлення інформації. Тому природно, що практично усі застосовні програми використовують екран, щоб відобразити дані, якими вони управляють. Навіть якщо це не так, то все одно візуальні елементи додатків відображаються на екрані самою операційною системою. Windows забезпечує універсальність представлення інформації як на екрані, так і на інших облаштуваннях виводу, наприклад, на принтері. Причому для цього використовуються одні і ті ж примітиви відображення. Система сама розпізнає цільовий пристрій і активізує модулі, що відповідають йому. У зв'язку з тим, що Windows — багатозадачна система, до застосовних програм пред'являється ряд вимог, які дозволяють виключити конфлікти при використанні функцій виводу. Проте це не означає, що Windows забезпечує додатки тільки набором функцій виводу на екран або друк — система управляє усім виводом.Система Windows создавалась, прежде всего, как графическая оболочка. Соответственно, главное отличие от многих других операционных систем — это графическое представление информации. Поэтому естественно, что практически все прикладные программы используют экран, чтобы отобразить данные, которыми они управляют. Даже если это не так, то все равно визуальные элементы приложений отображаются на экране самой операционной системой. Windows обеспечивает универсальность представления информации как на экране, так и на других устройствах вывода, например, на принтере. Причем для этого используются одни и те же примитивы отображения. Система сама распознает целевое устройство и активизирует соответствующие ему модули. В связи с тем, что Windows — многозадачная система, к прикладным программам предъявляется ряд требований, которые позволяют исключить конфликты при использовании функций вывода. Однако это не значит, что Windows обеспечивает приложения только набором функций вывода на экран или печать — система управляет всем выводом.

Взагалі, правильніше буде сказати, що додатки в якості первинного облаштування виводу використовують швидше вікна, чим безпосередньо екран. Кожен такий пристрій в Windows має ндбор поточних параметрів, з використанням яких і відбувається власне вивід.

Графічні об'єкти

Система Windows надає додатку набір графічних об'єктів, які дозволяють управляти виводом. Для цього застосовуються:Система Windows предоставляет приложению набор графических объектов, которые позволяют управлять выводом. Для этого применяются:

- ***Бітові образи*** (bitmaps) — прямокутні масиви точок, що формують растрові зображення.***Битовые образы*** (bitmaps) — прямоугольные массивы точек, формирующие растровые изображения.
- ***Олівці*** (pens) — використовуються для завдання параметрів малювання ліній, таких як товщина, колір і стиль (суцільна, переривчаста і тому подібне).***Карандаши*** (pens) — используются для задания параметров рисования линий, таких как толщина, цвет и стиль (сплошная, прерывистая и т.п.).
- ***Кисті*** (brushes) — використовуються для завдання параметрів заливки замкнутих контурів, таких як колір і стиль.***Кисти*** (brushes) — используются для задания параметров заливки замкнутых контуров, таких как цвет и стиль.
- ***Шрифти*** (fonts) — використовуються для завдання параметрів виведення тексту, включаючи ім'я шрифту, розмір символів і тому подібне***Шрифты*** (fonts) — используются для задания параметров вывода текста, включая имя шрифта, размер символов и т.п.
- ***Регіони*** (regions) — області вікна, які можуть бути обмежені прямокутником, багатокутником, еліпсом або їх комбінацією для виконання операцій заповнення, заливки, інверсії і тому подібне. Крім того, вони служать для визначення місця розташування курсора.***Регионы*** (regions) — области окна, которые могут быть ограничены прямоугольником, многоугольником, эллипсом или их комбинацией для выполнения операций заполнения, заливки, инверсии и т.п. Кроме того, они служат для определения местоположения курсора.
- ***Логічні палітри*** (logical palettes) — забезпечують інтерфейс між додатком і кольоровим облаштуванням виводу, таким як дисплей, містять список кольорів необхідних додатку.***Логические палитры*** (logical palettes) — обеспечивают интерфейс между приложением и цветным устройством вывода, таким как дисплей, содержат список цветов необходимых приложению.
- ***Контури*** (paths) — використовуються для цілей заповненні або виділення контура різних фігур.***Контуры*** (paths) — используются для целей заполнении или выделения контура различных фигур.

Апаратно-незалежний графічний вивід

Одна з головних особливостей Win32 API — незалежність пристроїв. Програмне забезпечення, яке підтримує цю незалежність, міститься в двох бібліотеках динамічного компонування (DLL). Перша, **GDI.DLL**, забезпечує графічний інтерфейс пристрою (Graphics Device Interface, GDI), друга є драйвером пристрою. Використання того або іншого драйвера залежить, природно, від облаштування виводу. Наприклад, якщо додаток здійснює вивід в зоні призначеного для користувача вікна на дисплеї VGA, ця бібліотека— **VGA.DLL**, якщо вивід здійснюється на принтер Epson FX — 80, ця бібліотека — **EPSON9.DLL.**Одна из главных особенностей Win32 API — независимость устройств. Программное обеспечение, которое поддерживает эту независимость, содержится в двух библиотеках динамической компоновки (DLL). Первая, **GDI.DLL**, обеспечивает графический интерфейс устройства (Graphics Device Interface, GDI), вторая является драйвером устройства. Использование того или иного драйвера зависит, естественно, от устройства вывода. Например, если приложение осуществляет вывод в области пользовательского окна на дисплее VGA, эта библиотека— **VGA.DLL**, если вывод осуществляется на принтер Epson FX-80, эта библиотека - **EPSON9.DLL.**

Перед операцією виводу на деякий пристрій додаток повинен запросити GDI про завантаження відповідного драйвера (зазвичай це відбувається автоматично і не вимагає від програміста додаткових зусиль із програмування). Як тільки драйвер завантажений, додаток може настроїти ряд параметрів виводу, таких як колір лінії і її ширина, тип кисті і її колір, шрифт, область відсікання і т. д. Windows забезпечує зберігання цих і інших параметрів в спеціальній структурі, що називається контекстом пристрою.Перед операцией вывода на некоторое устройство приложение должно запросить GDI о загрузке соответствующего драйвера (обычно это

происходит автоматически и не требует от программиста дополнительных усилий по программированию). Как только драйвер загружен, приложение может настроить ряд параметров вывода, таких как цвет линии и ее ширина, тип кисти и ее цвет, шрифт, область отсечения и т. д. Windows обеспечивает хранение этих и других параметров в специальной структуре, называемой контекстом устройства.

Контекст пристрою

Контекст пристрою — структура, що визначає набір графічних об'єктів і пов'язаних з ними атрибутів і графічних режимів, які власне і впливають на вивід. У таблиці 1 приведені атрибути графічного виведення контексту пристрою.*Контекст устройства* — структура, определяющая набор графических объектов и связанных с ними атрибутов и графических режимов, которые собственно и воздействуют на вывод. В таблице 1 приведены атрибуты графического вывода контекста устройства.

Таблиця 1 - Атрибути графічного виводу, пристрої, що входять в контекст**Таблица 1 -** Атрибуты графического вывода, входящие в контекст устройства

Атрибути графічного виводу	Значення за умовчанням	Лінії	Заливка	Текст	Растр	Коментарі
Колір фону	Білий	X	X	X		Олівець з призначеним стилем, кисть з штрихуванням, текст
Режим фону	OPAQUE	X	X	X		Двопозиційний перемикач
Логічний номер кисті	Біла кисть		X		X	Зафарбовувані області
Початок координат кисті	(0,0)		X		X	
Логічний номер області промальовування	Уся поверхня	X	X	X	X	
Логічний номер колірної палітри	Палітра за умовчанням	X	X	X		
Поточна позиція олівця	(0, 0)	X				
Режим графічного виводу	R2_COPYPEN	X	X			Логіка відображення зображень, що перекриваються
Логічний номер шрифту	Системний шрифт			X		
Міжсимвольний інтервал	0			X		
Режим відображення	MM_TEXT	X	X	X	X	Одна одиниця дорівнює одному пікселу
Логічний номер олівця	Чорний олівець	X	X			
Режим зафарбовування багатокутників	Альтернатив-ный		X			
Режим розтягування	Чорний по білому				X	
Вирівнювання меж тексту	По верхньому і лівому краям			X		
Колір тексту	Чорний для тексту і кистей з монохромним зафарбовуванням		X	X		Колір переднього плану
Вирівнювання рядків тексту	(0,0)			X		Відкидання зайвих символів

Атрибути графічного виводу	Значення за умовчанням	Лінії	Заливка	Текст	Растр	Коментарі
Протяжність вікна даних екрану	(1, 1)	X	X	X	X	Відображення координат вікна даних і вікна екрану
Початок координат вікна даних екрану	(0,0)	X	X	X	X	Те ж
Протяжність вікна екрану	(1, 1)	X	X	X	X	Те ж
Початок координат вікна екрану	(0,0)	X	X	X	X	Те ж

На відміну від більшості інших структур Win32 API, застосовна програма ма ніколи не має прямого доступу до контексту пристрою — налаштування параметрів здійснюється *тільки* за допомогою виклику відповідних функцій.В отличие от большинства других структур Win32 API, прикладная программа ма никогда не имеет прямого доступа к контексту устройства — настройка параметров осуществляется *только* посредством вызова соответствующих функций.

Графічні режими

Система Windows підтримує п'ять різних графічних режимів, які дозволяють додаткам визначати тип змішування кольорів, місце і параметри виводу і так далі :Система Windows поддерживает пять различных графических режимов, которые позволяют приложениям определять тип смешивания цветов, место и параметры вывода и т.д.:

- **Режим налаштування фону** — визначає, як відбувається змішування кольору фону текстових об'єктів і растрових зображень з полем виводу (вікно або екран).**Режим настройки фона** — определяет, как происходит смешивание цвета фона текстовых объектов и растровых изображений с полем вывода (окно или экран).
- **Режим відображення** (малювання) — визначає, як відбувається змішування кольору олівців (pens), кистей (brushes), текстових об'єктів і растрових зображень з полем виводу (вікно або екран).**Режим отображения** (рисования) — определяет, как происходит смешивание цвета карандашей (pens), кистей (brushes), текстовых объектов и растровых изображений с полем вывода (окно или экран).
- **Режим масштабування** — визначає перетворення логічних координат при графічному виводі у вікна, *на* екран або принтер.**Режим масштабирования** — определяет преобразования логических координат при графическом выводе в окна, *на* экран или принтер.
- **Режим заливки контурів** — визначає використання шаблонів кистей при заливці складних контурів.**Режим заливки контуров** — определяет использование шаблонов кистей при заливке сложных контуров.
- **Режим стискування** — визначає, яким чином відбувається перетворення кольорів растрових зображень при їх збільшенні (зменшенні).**Режим сжатия** — определяет, каким образом происходит преобразование цветов растровых изображений при их увеличении (уменьшении).

Як і у разі графічних об'єктів, контекст пристрою ініціалізував ця значеннями за умовчанням і для графічних режимів.

Режими відображення

Досі, говорячи про завдання координат, ми мали на увазі, що йдеться і зміщенні відносно лівого верхнього кута екрану — *екранні координапп* (screen coordinates) або лівого верхнього кута клієнтської зони вікна — *клієнтські координати* (client coordinates). В принципі, для усього, що стосується вікон в цілому і віконних повідомлень, наприклад, WM_MOVE, WM_MOUSEMOVE це правильно. Проте чи завжди зручно малювати лінію "по точках", коли по-перше, усе перевернуто з ніг на голову — координати по вертикалі ростуть вниз, по-друге, доводиться постійно піклуватися про масштаб, і по-третє, необхідно пам'ятати формули обчислення

координат точок при повороті і зміщенні, враховуючи недоліки такої системи координат?До сих пор, говоря о задании координат, мы подразумевали, что речь идет і смещении относительно левого верхнего угла экрана — *экранные координапп* (screen coordinates) или левого верхнего угла клиентской области окна - *клиентские координаты* (client coordinates). В принципе, для всего, что касается окон в целом и оконных сообщений, например, WM_MOVE, WM_MOUSEMOVE это правильно. Однако всегда ли удобно рисовать линию "по точкам", когда во-первых, все перевернуто с ног на голову — координаты по вертикали растут вниз, во-вторых, приходится постоянно заботиться о масштабе, и в-третьих, необходимо помнить формулы вычисления координат точек при повороте и смещении, учитывая недостатки такой системы координат?

Система Windows забезпечує програміста можливістю створення соб ственной системи координат. Тобто, маніпулюючи різними параметрами, задавати напрям і масштаб осей, а також початок координат.

Налаштування режимів відображення

Функції Win32 API цієї групи встановлюють і настроюють систему до ординат, яка використовується усіма функціями виводу — координати виводу задаються саме в логічних одиницях. Окрім одиниць по осях "x" і "y", функції цієї групи задають напрям осей і початок координат. Практично усі функції групи реалізовані парами. Функції з префіксом Get дозволяють отримати поточне значення параметра, а з префіксом Set встановлюють нове значення і повертають попереднє. За умовчанням початок координат знаходиться в лівому верхньому кутку області виводу (раніше усього вікна), вісь "x" спрямована зліва направо, а вісь "y" зверху вниз. Одиниця виміру — один піксел (режим MM_TEXT). До речі сказати, маєток тому на різних пристроях, внаслідок відмінності розмірів піксела фактичний (візуальний) розмір по осі "x" може не відповідати розміру по осі "y".Функции Win32 API этой группы устанавливают и настраивают систему ко ординат, которая используется всеми функциями вывода — координаты вывода задаются именно в логических единицах. Кроме единиц по осям "x" и "y", функции этой группы задают направление осей и начало координат. Практически все функции группы реализованы парами. Функции с префиксом Get позволяют получить текущее значение параметра, а с префиксом Set устанавливают новое значение и возвращают предыдущее. По умолчанию начало координат находится в левом верхнем углу области вывода (прежде всего окна), ось "x" направлена слева направо, а ось "y" сверху вниз. Единица измерения — один пиксел (режим MM_TEXT). Кстати сказать, имение поэтому на различных устройствах, вследствие различия размеров пиксела фактический (визуальный) размер по оси "x" может не соответствовать размеру по оси "y".

Примітка

Режим MM_TEXT названий так не тому, що він найбільш зручний для виведення тек ста, а тому, що напрям осей — суть його система координат — соответ ствует способу читання, прийнятому для європейських мов, — зліва направо і зверху вниз.

Тепер познайомимося з режимами відображення, які підтримує Windows. Всього їх вісім, і основна функція, яка дозволяє вибраний режим, а значить і систему координат, наступна:

```
int SetMapMode{int SetMapMode{
  HOC hdc,          // Дескриптор контексту пристроюHOC hdc,          // Дескриптор
контекста устройства
   int nMapMode);   // Новий режим відображенняint nMapMode);   // Новый режим
отображения
```

Ця функція встановлює напрями осей і визначає логічні одиниці, тобто одиниці виміру для системи координат, що задається.

Можливі режими задаються параметром nMapMode, який може набувати наступних значень:Возможные режимы задаются параметром nMapMode, который может принимать следующие значения:

MM_TEXT	Одна логічна одиниця дорівнює одному пікселу, вісь x спрямована направо, вісь y спрямована вниз. Цей режим заданий за умовчанням.
MM_HIENGLISH	Одна логічна одиниця дорівнює 0.001 дюйма, вісь x спрямована направо, вісь y спрямована вгору.

MM_HIMETRIC	Одна логічна одиниця дорівнює 0.01 міліметра, вісь x спрямована направо, вісь y спрямована вгору.
MM_LOENGLISH	Одна логічна одиниця дорівнює 0.01 дюйма, вісь x спрямована направо, вісь y спрямована вгору.
MM_LOMETRIC	Одна логічна одиниця дорівнює 0.1 міліметра, вісь x спрямована направо, вісь y спрямована вгору.
MM_TWIPS	Одна логічна одиниця — твип (twip) — рівна 1/20 пункту (point) або 1/1440 дюйма, вісь x спрямована направо, вісь y спрямована вгору.
MM_ANISOTROPIC	Режим дозволяє настроювати, за допомогою функцій SetWindowExt і SetViewportExt, розмірність (окремо для кожної з осей), їх напрями і початок відліку.
MM_CSOTROPIC	Режим дозволяє настроювати, за допомогою функцій SetWindowExt і SetViewportExt, розмірність осей, їх напрями і початок відліку, проте одна одиниця по осі x дорівнює одній одиниці по осі y, таким чином, при необхідності GDI настроює розмірність по осях, забезпечуючи співвідношення 1:1.

Примітка

Історично в поліграфії використовуються такі одиниці виміру, як пункти (points) і твипсы (twips), розміри яких відповідно дорівнюють (приблизно) 1/72 і 1/1440 дюйма, ми ж вимушені використати поняття "піксел", говорячи про точку екрану.

Відмітимо, що тільки режими MM_HIENGLISH, MM_HIMETRIC, MM_LOENGLISH, MM_LOMETRIC і MM_TWIPS забезпечують додатки фізично зрозумілими одиницями виміру, з одного боку, а з іншої — певну незалежність від облаштувань виводу.

Для перших шести режимів можна міняти тільки початок координат, який за умовчанням відповідає лівому верхньому куту області виводу. Для режимів MM_ANISOTROPIC і MM_ICOTROPIC додатково можна настроювати напрям осей і масштаб. Для цього в Win32 API має у своєму складі функції, що описуються нижче.Для первых шести режимов можно менять только начало координат, которое по умолчанию соответствует левому верхнему углу области вывода. Для режимов MM_ANISOTROPIC и MM_ICOTROPIC дополнительно можно настраивать направление осей и масштаб. Для этого в Win32 API имеет в своем составе функции, описываемые ниже.

Для налаштування системи координат Windows використовує два поняття — *фізична область виводу* (viewport), координати і розміри якої задаються у фізичних одиницях — пікселах, і *логічна область виводу* (window), координати і розміри якої задаються в логічних одиницях, які визначаються режимом малювання.Для настройки системы координат Windows использует два понятия — *физическая область вывода* (viewport), координаты и размеры которой задаются в физических единицах — пикселах, и *логическая область вывода* (window), координаты и размеры которой задаются в логических единицах, которые определяются режимом рисования.

І фізична, і логічна області виводу характеризуються точкою, що визначає початок координат. Для налаштування початку координат фізичної області виводу використовується функція

```
BOOL SetViewportOrgEx{BOOL SetViewportOrgEx{
  HDC hdc
  int x
  int y
  LPPOINT lpPoint)LPPOINT lpPoint)
```

x, y — нові координати початку координат фізичної області виводу; lpPoint — покажчик на структуру, куди записується попереднє значення початку.

Для налаштування початку координат логічної області виводу використовується функція.

```
BOOL SotWindowOrgExf
  HDC hdc
  int x
  int yint y,
  LPP01NT lpPoint)LPP01NT lpPoint)
```

Обидві функції в якості параметрів набувають нових значень для установки початку координат, а повертають попередні значення. Причому, якщо для фізичної області виводу вказуються координати, які надалі будуть початком координат, то для логічної області виводу вказуються логічні координати, які надалі відповідатимуть лівому верхньому куту логічної області виводу.

З вищесказаного виходить, що настроювати початок координатної сегки є сенс тільки для однієї з областей, оскільки система координат логічної області пов'язана з системою координат фізичної області. Пояснимо сказане на прикладах.

Після відповідного налаштування обробники повідомлення WM_PAINT працюватимуть з координатною сіткою, початок координат якої розташовується в середині клієнтської області (малюнок 1).

```
    HDC hdc;
  RECT rect;
  PAINTSTRUCT ps;
  ...
  case WM_PAINT : {
    hdc = BeginPaint(hwnd, &ps);
    CRect rect;
    GetClientRect(hwnd, rect};
    SetViewportOrg(hdc, rect.cx / 2, rect.cy / 2, &rect);
    EndPaint(hwnd, &ps);
  }
  HDC hdc;
  PAINTSTRUCT ps;
  case WM_PAINT: {
    hdc = BeginPaint(hwnd, &ps);
    RECT rect[l];
    GetClientRect(hwnd, rect);
    DPtoLP(hdc, rect, 1};
    SetWindowOrgEx(hdc, -rect.ex / 2, -rect.cy / 2, srect);
    ...
    EndPaint(hwnd, &ps);
  }
```

Звіт повинен містити:

1) Найменування і мета роботи.
2) Тексти програм з необхідними коментарями.
3) Опис методики створення шаблонів ресурсів (меню і вікон діалогів).
4) Висновки по лабораторній роботі.

Контрольні питання

1. Перерахуйте набір графічних об'єктів Windows, які управляють виведенням інформації.Перечислите набор графических объектов Windows, которые управляют выводом информации.
2. Що таке контекст пристрою?. Які типи контекстів застосовуються в Windows?Что такое контекст устройства?. Какие типы контекстов применяются в Windows?
3. Перерахуйте атрибути графичекого виводу.
4. Які графічні режими підтримуються в Windows?Какие графические режимы поддерживаются в Windows?
5. У чому полягає налаштування режимів відображення?
6. Для чого використовується налаштування системи координат?
7. З якою метою використовуються зломщики сообшений?
8. Дайте визначення основним типам меню.
9. Перерахуйте модифікатори елементів меню.
10. Поясніть, як створюється шаблон меню в лабораторній роботі.

Компьютерные технологии и программирование

Методические указания для лабораторных работ
для студентов специальности
123