“МЕТОДИКА ПРЕПОДАВАНИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЕ”

методические указания

составитель: **Д.Хакимова**

Введение

Данные методические указания предназначены для дисциплины программирования на языке Си, читаемой студентам специальности 5350400 «Профессиональное обучение в сфере ИКТ». Язык программирования Си был создан в 1972 г. Денисом Ритчи. На сегодняшний день это самый распространенный и популярный язык программирования, позволяющий строить эффективные программы для различных областей программирования, в том числе и решения задач системного программирования и защиты информации. Не случайно самые популярные на сегодняшний день операционные системы Ms Windows и UNIX написаны именно на этом языке программирования.

Материал, рассматриваемый в данном пособии, является основой для изучения последующих специальных дисциплин специальности 5350400 «Профессиональное обучение в сфере ИКТ».

Практическая работа № 1.

Тема :Алгоритмы, линейные алгоритмы.

Теоретическая часть.

Каждый алгоритм сортировки состоит из трех основных частей:

- Функция сравнения пары элементов сортируемого массива
- Процедура перестановки, меняющая местами пару элементов
- Сортирующий алгоритм, который осуществляет сравнение и перестановку элементов до тех пор, пока все элементы множества не будут упорядочены.

```
#define less(x,y)  (x < y)            // функция сравнения элементов
#define swap(x,y)  {int t=x; x=y; y=t;} // процедура перестановки элементов
```

Пузырьковая сортировка

Операция сравнения/перестановки выполняется для каждых двух стоящих рядом элементов. После первого прохода по массиву "вверху" оказывается самый "легкий" элемент. Следующий цикл сортировки выполняется начиная со второго элемента, в результате чего вторым в массиве оказывается второй наименьший по величине элемент, и так далее.
Алгоритм прост и крайне неэффективен.

```
void sort_bubble(int a[], int max)      // пузырьковая сортировка
  {
    for (int i=0; i<max; i++)
    for (int j=max-1; j>i; j--)
      if (less(a[j], a[j-1]))
        swap(a[j], a[j-1]);
  }
```

Сортировка выбором

Во время i-го прохода по массиву выявляется наименьший элемент, который затем меняется местами с i-м.
Все остальное аналогично пузырьковой сортировке.

```
void sort_choose(int a[], int max)      // сортировка выбором
  {
```

```
   for (int i=0; i<max; i++)
     {
       int k=i;
       for (int j=i+1; j<max; j++)
         if (less(a[j], a[k]))
           k=j;
       if (i!=k)
         swap(a[i], a[k]);
     }
 }
```

Сортировка Шелла

Основная идея алгоритма заключается в том, чтобы вначале устранить массовый беспорядок в массиве, сравнивая далеко стоящие друг от друга элементы.
Интервал между сравниваемыми элементами постепенно уменьшается до единицы. Это означает, что на поздних стадиях сортировка сводится просто к перестановкам соседних элементов.

```
void sort_shell(int a[], int max)              // сортировка Шелла
  {
    for (int gap = max/2; gap>0; gap/=2)       // выбор интервала
      for (int i=gap; i<max; i++)              // проход массива
// сравнение пар, отстоящих на gap друг от друга
        for (int j=i-gap; j>=0 && less(a[j+gap],a[j]); j-=gap)
          swap(a[j], a[j+gap]);
  }
```

Сортировка Хоора

Суть метода заключается в том, чтобы найти такой элемент множества, подлежащего сортировке, который разобьет его на два подмножества: те элементы, что меньше делящего элемента, и те, что не меньше его.

Для массива из 10 чисел установим два индекса: на первый (i) и на последний (j) элементы.

```
10 7 28 49 31 25 17 3 14 43
i                    <--j
                  step==-1
```

На каждом шаге будем изменять значение индекса j на значение step (вначале оно равно -1, индекс уменьшается), т.е. двигать j влево. Мы будем делать так в том случае, если j-й элемент больше, чем i-й элемент. Если это условие не

выполнено - поменяем местами i-й и j-й элементы массива и сами индексы i и j, а также изменим знак у значения step -- оно станет равным +1:

```
3 7 28 49 31 25 17 10 14 43
j-->            i
step==+1
```

Продолжим процесс: теперь j движется вправо, и изменится условие - сейчас для продолжения процесса необходимо, чтобы j-й элемент был меньше i-го.

Будем продолжать вышеописанный процесс до тех пор, пока индексы i и j не станут одинаковыми.

В этот момент можно утверждать, что i-й (он же и j-й) элемент разделил исходное множество на два подмножества: все элементы, находящиеся слева от делящего элемента, меньше его, и все элементы, находящиеся справа, не меньше делящего (i-го) элемента. Получилось, что i-й элемент стоит на своем месте:

```
3 7 10 49 31 25 17 28 14 43
   ij
```

Таким образом, мы описали процедуру нахождения расположения одного элемента. Иными словами, мы "отсортировали" один элемент множества. Такая же процедура применима к элементам "левого" и "правого" подмножеств, которые на данный момент еще не отсортированы.

Условие завершения нашей рекурсивной процедуры - совпадение границ, которое эквивалентно тому, что в множестве остался один элемент.

```
void sort_hoor(int a[], int l, int r)   // сортировка Хоора
  {
    int i=l, j=r, step=-1, condition=1;
    if (l>=r) return;                  // сортировать нечего

    do {                        // сортируем с левой границы до правой
      if (condition == less(a[j],a[i]))
        {
          swap(a[i], a[j]);           // перестановка чисел
          swap(i, j);                 // обмен местами индексов
          step = -step;               // теперь - в другую сторону
          condition = !condition;     // обмен условия на противоположное
        }
      j += step;                      // передвинем индекс
    } while (j!=i);                   // пока индексы не сойдутся
```

```
  sort_hoor(a, l, i-1);              // левое подмножество
  sort_hoor(a, i+1, r);               // правое
 }
```

QuickSort

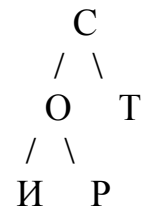
По существу является разновидностью сортировки Хоора, но программная реализация немного красивее и быстрее.

Основное отличие - за делящий элемент всегда выбирается середина массива.

```
void sort_quick(int a[], int l, int r) // QuickSort
 {
  int i=l, j=r, x=a[(l+r)/2];
  do {
    while (less(a[i],x)) i++;
    while (less(x,a[j])) j--;
    if (i<=j) {
      swap(a[i], a[j]);
      i++;
      j--;
    };
  } while (i<j);
  if (l<j) sort_quick(a,l,j);
  if (i<r) sort_quick(a,i,r);
 }
```

Сортировка с помощью двоичного дерева

Двоичным деревом назовем упорядоченную структуру данных, в которой каждому элементу -- предшественнику или корню (под)дерева -- поставлены в соответствие по крайней мере два других элемента (преемника). Причем для каждого предшественника выполнено следующее правило: левый преемник всегда меньше, а правый преемник всегда больше или равен предшественнику. Если мы составим такое дерево из букв слова "СОРТИР", сравнивая ASCII коды букв, то получим следующую структуру:



Как видно, узел "С" имеет два преемника: левый "О" и правый "Т". Если составить бинарное дерево из элементов неупорядоченного массива, то в

общем случае дерево получится достаточно хорошо заполненным (а если массив уже был рассортирован, то дерево выродится в линейный список). Если мы будем обходить дерево по правилу "ЛЕВЫЙ преемник - КОРЕНЬ - ПРАВЫЙ преемник", то, записывая все встречающиеся элементы в массив, мы получим упорядоченное в порядке возрастания множество. На этом и основана идея сортировки деревом.

Использование такой сортировки удобно тогда, когда удобно представлять данные в виде дерева.

Недостаток метода - требуется много памяти. В приведенном примере - дополнительный мегабайт данных на каждые 256k элементов.

```
/********** сортировка с помощью двоичного дерева **********/

typedef struct tree
  {
    int a;              // данные
    struct tree *left;  // левый  преемник
    struct tree *right; // правый преемник
  } TREE;

TREE *add_to_tree(TREE *root, int new_value)
{
   if (root==NULL)  // если дошли до корня - создаем новый элемент
     {
       root = (TREE*)malloc(sizeof(TREE));
       root->a = new_value;
       root->left = root->right = 0;
       return root;
     }
   if less(root->a, new_value)         // добавлем ветвь
     root->right = add_to_tree(root->right, new_value);
   else
     root->left  = add_to_tree(root->left,  new_value);
   return root;
}

void tree_to_array(TREE *root, int a[]) // процедура заполнения массива
  {
    static max2=0;                  // счетчик элементов нового массива
    if (root==NULL) return;             // условие окончания - найден корень
    tree_to_array(root->left, a);       // обход левого поддерева
    a[max2++] = root->a;
```

```
   tree_to_array(root->right, a);      // обход правого поддерева
   free(root);
  }

void sort_tree(int a[], int max)       // собственно сортировка
{
  TREE *root = 0;
  for (int i=0; i<max; i++)           // проход массива и заполнение дерева
    root = add_to_tree(root, a[i]);
  tree_to_array(root, a);             // заполнение массива
}
```

Сортировка с помощью массива индексов

Это не столько метод, сколько хинт для ускорения любого метода сортировки, когда в качестве данных используются структуры большого размера.

Идея заключается в том, что параллельно с основным массивом данных существует так называемый массив индексов, через который осуществляется перестановка индексов основного массива данных.

```
struct MASS a[10000]; // основной массив
int index[10000];     // массив индексов {0,1,2,...,9999}
```

Перед сортировкой массив индексов инициализируется соответствующими порядковыми номерами.

Доступ к массиву данных ведется не как a[i], а как a[index[i]], и функция сравнения теперь выглядит иначе:

```
#define less(i,j)  (a[index[i]].XPEH_member < a[index[j]].MASS_member)
```

В результате фактически сортируется массив с индексами, а не с данными.

```
void sort_quick_index(MASS a[], int index[], int l, int r)
 {
   int i=l, j=r, x=(l+r)/2;           // теперь x не элемент а его индекс
   do {
     while (less(i,x)) i++;
     while (less(x,j)) j--;
     if (i<=j)
       {
```

```
        if (x==i) x==j; else if (x==j) x==i;
        swap(index[i], index[j]);     // сортируем индексы
        i++;
        j--;
       };
  } while (i<j);
  if (l<j) sort_quick_index(a,index,l,j);
  if (i<r) sort_quick_index(a,index,i,r);
 }
```

А теперь сравните по скорости тысячу операций swap(x,y), где в первом случае x и y - это большие структуры данных, а во втором - числа по 2-4 байта.

Практическая работа №2

Тема : Организация ввода-вывода С ++.

Теоретическая часть.

Программа на языке С++ состоит из одного или нескольких модулей, называемых функциями. Выполнение программы начинается с функции main () (все символы — в нижнем регистре), поэтому ваша программа обязательно должна включать эту функцию. Функция состоит из заголовка и тела. В заголовке функции указывается, каким является возвращаемое значение (если таковое существует), генерируемое функцией, и какую информацию принимает функция в виде аргументов. Тело функции состоит из последовательности операторов языка С++, заключенных в фигурные скобки ({ }).

В языке программирования С++ выделяют следующие типы операторов:

- Оператор объявления. В операторе объявления указывается имя и тип переменной, которая используется в функции.
- Оператор присваивания. Этот оператор использует операцию присваивания (=) для присваивания значения переменной.
- Оператор сообщений. Оператор сообщений посылает сообщение объекту, инициируя некоторое действие.
- Вызов функции. Вызов функции активизирует ее работу. Когда вызываемая функция завершает свою работу, программа возвращается к оператору в вызывающей функции, следующим за вызовом функции.
- Прототип функции. В прототипе функции объявляется тип возвращаемого функцией значения, а также количество и тип аргументов, передаваемых функции.
- Оператор возврата. Оператор возврата посылает значение из вызываемой функции обратно вызывающей функции. Класс представляет собой определяемую пользователем спецификацию типа данных. В ней подробно описывается способ представления информации и действия, которые могут выполняться над этими данными. Объект — это сущность, созданная в соответствии с предписанием класса, а простая переменная является сущностью, созданной в соответствии с описанием типа данных.

В языке С++ имеются два предварительно определенных объекта (cin и cout) для обработки ввода и вывода. Они являются примерами классов istream и ostream, которые определены в файле iostream. Эти классы рассматривают ввод и вывод в виде потоков символов. Операция вставки (<<), которая определена для класса ostream, позволяет помещать данные в поток вывода, а операция извлечения (>>), которая определена для класса istream, позволяет извлекать информацию из потока ввода. Как cin, так и cout являются интеллектуальными объектами, способными автоматически преобразовывать информацию из одной формы в другую в соответствии с контекстом

программы.
Язык программирования C++ может использовать обширный набор библиотечных функций языка C. Чтобы использовать библиотечную функцию, необходимо включить заголовочный файл, который предоставляет для функции ее прототип.
Теперь, когда вы уже знаете, что собой представляют простые программы на языке C++, можно приступить к изучению следующей главы, вникая во все более широкий круг вопросов.

Практическая часть

1. Напишите программу на C++, которая отобразит вашу фамилию и почтовый адрес на экране монитора.
2. Напишите программу на C++, которая выдает запрос на ввод расстояния в фар- лонгах и преобразовывает его в ярды. (Один фарлонг равен 220 ярдам.)
3. Напишите программу на C++, которая использует три определяемых пользователем функции (включая main ()), и результатом ее выполнения является следующий вывод:
Three blind mice Three blind
mice See how they run See how
they run
Одна функция, вызываемая два раза, должна генерировать первые две строки, а вторая функция, также вызываемая два раза, должна генерировать оставшиеся строки.
4. Напишите программу, в которой функция main () вызывает определяемую пользователем функцию, которая в качестве аргумента принимает значение температуры по Цельсию и возвращает эквивалентное значение температуры по Фаренгейту. Программа должна выдать запрос на ввод значения по Цельсию и отобразить следующий результат:
Please enter a Celsius value: 20
20 degrees Celsius is 68 degrees Fahrenheit.

Для справки, формула для выполнения этого преобразования:

Температура в градусах по Фаренгейту = 1,8 * Температура в градусах по Цельсию + 32
5. Напишите программу, в которой функция main () вызывает определяемую пользователем функцию, которая в качестве аргумента принимает расстояние в световых годах и возвращает расстояние в астрономических единицах. Программа должна выдать запрос на ввод значения светового года и отобразить следующий результат:
Enter the number of light years: 4.2
4.2 light years = 265608 astronomical units.

Астрономическая единица равна среднему расстоянию Земли от Солнца (около 150 000 000 км, или 93 000 000 миль), а световой год соответствует

расстоянию, пройденному лучом света за один земной год (примерно 10 триллионов километров, или 6 триллионов миль). (Ближайшая звезда после Солнца находится на расстоянии 4.2 световых года.) Используйте тип double (как в листинге 2.4) и следующий коэффициент преобразования:
1 световой год = 63 240 астрономических единиц

6. Напишите программу, которая выдает запрос на ввод значения часов и значения минут. Функция main () должна передать эти два значения функции, имеющей тип void, которая отображает эти два значения в следующем виде:

Enter the number of hours: 9 Enter the number of minutes: 28 Time: 9:28

Практическая работа № 3

Тема: Условия в C++. Создание ветвлений в программе.

Теоретическая часть.

Программы и процесс программирования становятся более интересными, когда вводятся операторы, которые позволяют программе выбрать альтернативные действия. В C++ имеются операторы if, if else, а также switch, представляющие собой средства управления выбором пути выполнения, if позволяет программе выполнить оператор или блок операторов в случае удовлетворения некоторого условия. То есть программа выполняет этот оператор или блок, только если конкретное условие истинно, if else позволяет программе выбрать для выполнения одно из двух операторов или блоков. Вы можете добавлять дополнительные операторы if else для представления серии вариантов выбора. Оператор C++ switch направляет поток управления программы в определенное место из списка возможных.

В C++ также доступны операции, помогающие принимать решения. В главе 5 обсуждаются выражения отношений, которые сравнивают два значения, if и if else обычно в качестве проверочных условий используют выражения сравнения. Используя логические операции C++ (&&, | | и !), вы можете комбинировать или модифицировать выражения сравнения для конструирования более сложных тестов. Условная операция (? :) предлагает компактный способ выбора одного из двух значений по условию.

Библиотека символьных функций cctype предлагает удобный и мощный набор инструментов анализа символьного ввода.

Циклы и операторы выбора — это полезные инструменты для организации файлового ввода-вывода, который во многом повторяет консольный. После того как вы объявляете объекты ifstream и ofstream и ассоциируете их с файлами, их можно использовать в той же манере, что и стандартные cin и cout.

Используя циклы и условные операторы C++, вы можете писать интересные, интеллектуальные и мощные программы. Но мы только начали исследовать реальную мощь языка C++. Далее мы обратимся к функциям.

Практическая часть.

1. Напишите программу, которая читает клавиатурный ввод до символа @ и повторяет его, за исключением десятичных цифр,

преобразуя каждую букву верхнего регистра в букву нижнего регистра и наоборот. (Не забудьте о семействе cctype.)

2. Напишите программу, читающую в массив double до 10 значений пожертвований. Программа должна прекращать ввод при получении нечисловой величины. Она должна выдавать среднее значение полученных чисел, а также количество значений в массиве, превышающих среднее.
3. Напишите предшественник программы, управляемой меню. Она должна отображать меню из четырех пунктов, каждый из них помечен буквой. Если пользователь вводит букву, отличающуюся от четырех допустимых, программа должна повторно приглашать его ввести правильное значение до тех пор, пока он этого не сделает. Затем она должна выполнить некоторое простое действие на основе пользовательского выбора. Работа программы должна выглядеть примерно так:
   Пожалуйста, введите одно из следующих значений: c)
   хищник p) пианист t) дерево д) игра **f**
   Пожалуйста, введите ac, p, t, или g: q
   Пожалуйста, введите a c, p, t, или g: **t** Клен —
   это дерево.
4. Когда вы вступите в Благотворительный Орден Программистов (БОП), к вам могут обращаться на заседаниях БОП по вашему настоящему имени, по должности либо секретному имени БОП. Напишите программу, которая может выводить списки членов по настоящим именам, должностям, секретным именам либо по предпочтению самого члена. В основу положите следующую структуру:

```
// Структура имен Благотворительного Ордена Программистов (БОП)
struct bop {
    char fullname[strsize]; // настоящее имя
    char title[strsize];        // должность
    char bopname[strsize]; // секретное имя БОП
    int preference;             // 0 = полное имя, 1 = титул, 2 = имя БОП
};
```

В этой программе создайте небольшой массив таких структур и инициализируйте его соответствующими значениями. Пусть программа запустит цикл, который даст возможность пользователю выбирать разные альтернативы:

a. отображать по именам Ь. Отображать по должностям
c. отображать по именам БОП d. отображать по предпочтениям q.
выйти

Обратите внимание, что “отображать по предпочтениям” — не значит, что нужно отобразить предпочтение члена; это значит, что

нужно отобразить значение того поля структуры, которое соответствует предпочтению. Например, если preference равно 1, то выбор d должен вызвать отображение должности для данного программиста. Пример запуска этой программы может выглядеть примерно так:

```
Отчет о Благотворительно Ордене Программистов
a. отображать по именам     Ь. Отображать по должностям
c. отображать по именам БОП d. отображать по предпочтениям
q. выйти
Ваш выбор: a
Wimp Macho
Raki Rhodes
Celia Laiter
Hoppy Hipman
Pat Hand
Следующий выбор: d Wimp
Macho Junior Programmer
MIPS
Analyst Trainee LOOPY
Следующий выбор: q Пока!
```

5. Королевство Нейтрония, где денежной единицей служит тварп, использует следующую шкалу налогообложения:
Первые 5 ООО тварпов — налог 0%
Следующие 10 ООО тварпов — налог 10%
Следующие 20 000 тварпов — налог 15%
Свыше 35 000 тварпов — налог 20%
Например, если некто зарабатывает 38 ООО тварпов, то он должен заплатить налогов 5000 х 0.00 + 10000 х 0.10 + 20000 х 0.15 + 3000 х 0.20, или 4 600 тварпов. Напишите программу, которая использует цикл для запроса доходов и выдачи подлежащего к выплате налога. Цикл должен прерываться, когда пользователь вводит отрицательное или нечисловое значение.
6. Скомпонуйте программу, которая отслеживает пожертвования в Общество Защиты Влиятельных Лиц. Она должна запрашивать у пользователя количество меценатов, а затем приглашать вводить их имена и суммы пожертвований от каждого. Информация должна сохраняться в динамически выделенном массиве структур. Каждая структура должна иметь два члена: символьный массив (или объект string) для хранения имени и переменную-член типа double — для хранения суммы пожертвования. После чтения всех данных программа должна отображать имена и суммы пожертвований тех, кто не пожалел $10 000 и более. Этот список должен быть озаглавлен меткой Grand Patrons. После этого

программа должна выдать список остальных жертвователей. Он должен быть озаглавлен Patrons. Если в какой-то из двух категорий не окажется никого, программа должна напечатать “попе”. Помимо отображения двух категорий, никакой другой сортировки делать не нужно.

Практическая работа № 4

Тема : Простые действия над массивами.

Теоретическая часть.

Массивы, структуры и указатели — три составных типа в C++. Массив может содержать множество значений одного и того же типа в одном объекте данных. Используя индекс, вы получаете доступ к индивидуальным элементам массива.

Структура может содержать несколько значений разных типов в одном объекте данных, и для доступа к ним вы можете использовать операцию членства (.). Первым шагом в применении структуры является создание шаблона структуры, который определяет члены, входящие в нее. Имя, или дескриптор такого шаблона затем становится идентификатором нового типа данных. Далее вы можете объявлять структурные переменные этого типа.

Объединение может содержать единственное значение, которое может трактоваться разными типами, причем имя члена при этом указывает, какой режим (тип) используется в данный момент.

Указатели — это переменные, которые предназначены для хранения адресов памяти. Говорят, что указатель указывает на адрес, который он хранит. Объявление указателя всегда включает тип объекта, на который он указывает. Применяя операцию разыменования (*), можно получить значение, находящееся в памяти по адресу, хранящемуся в указателе.

Строка — это серия символов, ограниченная нулевым символом. Строка может’ быть представлена в виде строковой константы, заключенной в кавычки, при этом нулевой символ подразумевается неявно. Вы можете сохранить строку в массиве char и представить строку как указатель на char, представляющий ее начальный символ. Функция strlen () возвращает длину строки, исключая нулевой символ-ограничитель. Функция strcpy () копирует строку из одного места в другое. Для применения этих функций необходимо включить в программу заголовочный файл cstring или string.h.

Класс C++ string, поддерживаемый заголовочным файлом string, предлагает альтернативный, более дружественный к пользователю способ обращения со строками. В частности, объекты string автоматически изменяют свои размеры, приспосабливаясь к хранимым строкам, а для копирования их можно применять операцию присваивания.

Операция new позволяет запрашивать память для объектов данных во время выполнения программы. Эта операция возвращает адрес выделенного участка памяти, и вы можете присвоить его указателю. Единственный способ доступа к такой памяти— через указатель. Если объект данных — это простая переменная, вы можете применить операцию разыменования (*) для получения значения. Если объект данных — массив, вы можете использовать указатель на него как обычное имя массива и обращаться к его элементам по индексу. Если же объект данных — структура, вы можете использовать

операцию -> для доступа к ее членам.

Указатели и массивы тесно связаны. Если ar — имя массива, то выражение ar [i] интерпретируется как * (ar + i), то есть имя массива интерпретируется как адрес его первого элемента. Таким образом, имя массива играет ту же роль, что и указатель. В свою очередь, вы можете использовать имя указателя в нотации массива, чтобы обращаться к элементам массива, выделенным операцией new.

Операции new и delete позволяют явно контролировать, когда объекты данных размещаются и когда покидают пул свободной памяти. Автоматические переменные, которые объявляются внутри функций, и статические переменные, определяемые вне функций или же с помощью ключевого слова static, являются менее гибкими. Автоматические переменные создаются, когда управление приходит в содержащий их блок (обычно в теле функции), и исчезают, когда оно его покидает. Статические переменные сохраняются в течение всего времени исполнения программы.

Практическая часть.

1. Напишите программу C++, которая запрашивает и отображает информацию, как показано в следующем примере вывода:
   What is your first name? Betty Sue
   What is your last name? Yew
   What letter grade do you deserve? B
   What is your age? 22
   Name: Yew, Betty Sue
   Grade: C
   Age: 22
   Обратите внимание, что программа должна принимать имена, состоящие из более чем одного слова. Также отметьте, что программа должна уменьшать значение **grade** на один шаг — то есть, на одну букву выше. Предполагается, что пользователь может ввести A, B или C, поэтому вам не нужно беспокоиться о пропуске между D и E
2. Перепишите листинг 4.4, применив класс C++ **string** вместо массивов **char.**
3. Напишите программу, которая просит пользователя ввести ее или его имя, затем фамилию, а затем построит, сохранит и отобразит третью строку, состоящую из фамилии пользователя, за которой следует запятая, пробел и его имя. Используйте массивы **char** и функции из заголовочного файла **cstring.** Пример запуска должен выглядеть так:
   Enter your first name: Flip Enter your last name: Fleming
   Here's the information in a single string: Fleming, Flip
4. Напишите программу, которая приглашает пользователя ввести его имя и фамилию, а затем построит, сохранит и отобразит третью строку, состоящую из фамилии, за которой следует запятая, пробел и имя.

Используйте объекты **string** и методы из заголовочного файла **string.** Пример запуска должен выглядеть так:
Enter your first name: Flip Enter your last name: Fleming
Here's the information in a single string: Fleming, Flip

5. Структура **CandyBar** включает три члена. Первый из них содержит наименование коробки конфет. Второй — ее вес (который может иметь дробную часть), а третий — число калорий (целое значение). Напишите программу, объявляющую эту структуру и создающую переменную типа **CandyBar** по имени **snack,** инициализируя ее члены значениями **"Mocha Munch",** 2.3 и 350, соответственно. Инициализация должна быть частью объявления **snack.** И, наконец, программа должна отобразить содержимое этой переменной.

Практическая работа № 5

Тема : Двумерные и многомерные массивы и их использование

Теоретическая часть.

Массивы, структуры и указатели — три составных типа в C++. Массив может содержать множество значений одного и того же типа в одном объекте данных. Используя индекс, вы получаете доступ к индивидуальным элементам массива.

Структура может содержать несколько значений разных типов в одном объекте данных, и для доступа к ним вы можете использовать операцию членства (.). Первым шагом в применении структуры является создание шаблона структуры, который определяет члены, входящие в нее. Имя, или дескриптор такого шаблона затем становится идентификатором нового типа данных. Далее вы можете объявлять структурные переменные этого типа.

Объединение может содержать единственное значение, которое может трактоваться разными типами, причем имя члена при этом указывает, какой режим (тип) используется в данный момент.

Указатели — это переменные, которые предназначены для хранения адресов памяти. Говорят, что указатель указывает на адрес, который он хранит. Объявление указателя всегда включает тип объекта, на который он указывает. Применяя операцию разыменования (*), можно получить значение, находящееся в памяти по адресу, хранящемуся в указателе.

Строка — это серия символов, ограниченная нулевым символом. Строка может’ быть представлена в виде строковой константы, заключенной в кавычки, при этом нулевой символ подразумевается неявно. Вы можете сохранить строку в массиве char и представить строку как указатель на char, представляющий ее начальный символ. Функция strlen () возвращает длину строки, исключая нулевой символ-ограничитель. Функция strcpy () копирует строку из одного места в другое. Для применения этих функций необходимо включить в программу заголовочный файл cstring или string.h.

Класс C++ string, поддерживаемый заголовочным файлом string, предлагает альтернативный, более дружественный к пользователю способ обращения со строками. В частности, объекты string автоматически изменяют свои размеры, приспосабливаясь к хранимым строкам, а для копирования их можно применять операцию присваивания.

Операция new позволяет запрашивать память для объектов данных во время выполнения программы. Эта операция возвращает адрес выделенного участка памяти, и вы можете присвоить его указателю. Единственный способ доступа к такой памяти— через указатель. Если объект данных — это простая переменная, вы можете применить операцию разыменования (*) для получения значения. Если объект данных — массив, вы можете использовать указатель на него как обычное имя массива и обращаться к его элементам по индексу. Если же объект данных — структура, вы можете использовать

операцию -> для доступа к ее членам.

Указатели и массивы тесно связаны. Если ar — имя массива, то выражение ar [i] интерпретируется как * (ar + i), то есть имя массива интерпретируется как адрес его первого элемента. Таким образом, имя массива играет ту же роль, что и указатель. В свою очередь, вы можете использовать имя указателя в нотации массива, чтобы обращаться к элементам массива, выделенным операцией new.

Операции new и delete позволяют явно контролировать, когда объекты данных размещаются и когда покидают пул свободной памяти. Автоматические переменные, которые объявляются внутри функций, и статические переменные, определяемые вне функций или же с помощью ключевого слова static, являются менее гибкими. Автоматические переменные создаются, когда управление приходит в содержащий их блок (обычно в теле функции), и исчезают, когда оно его покидает. Статические переменные сохраняются в течение всего времени исполнения программы.

Практическая часть.

6. Структура **CandyBar** включает три члена, как описано в предыдущем упражнении. Напишите программу, которая создает массив из трех структур **CandyBar,** инициализирует их значениями на ваше усмотрение и затем отображает содержимое каждой структуры.
7. Вильям Вингейт (William Wingate) заведует службой анализа пиццы. О каждой пицце он записывает следующую информацию:
   - Наименование компании — производителя пиццы, которое может состоять из более чем одного слова.
   - Диаметр пиццы.
   - Вес пиццы.

   Разработайте структуру, которая может содержать всю эту информацию, и напишите программу, использующую структурную переменную этого типа. Программа должна запрашивать у пользователя ввод каждого из перечисленных показателей и затем отображать введенную информацию. Применяйте **cin** (или его методы) и **cout.**
8. Выполните упражнение **7,** но с применением **new** для размещения структуры в свободной памяти вместо объявления структурной переменной. Кроме того, сделайте так, чтобы программа сначала запрашивала диаметр пиццы, а потом — наименование компании.
9. Выполните упражнение 6, но вместо объявления массива из трех структур **CandyBar** используйте операцию **new** для динамического размещения массива.

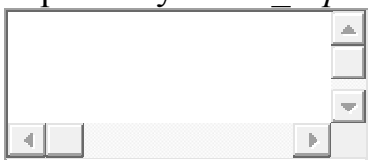
Практическая работа №6.

Тема : Указатели и их использование. Работа с указателями

Теоретическая часть.

Указатели широко используются в C++. В чем то, именно их наличие сделало этот язык более удобным для системного программирования. Но стоит отметить, что это одна из наиболее сложных для освоения возможностей C++. Идея работы с указателями состоит в том, что пользователь работает с адресом ячейки памяти и имеет возможность динамически создавать и уничтожать переменные.

Как правило, при обработке оператора объявления переменной *тип имя_переменной;* компилятор автоматически выделяет память под переменную *имя_переменной* в соответствии с указанным типом:



```
1 char C='$'; //Будет выделена под символьную переменную C,
2 //и ей присвоено стартовое значение
```

Доступ к объявленной переменной осуществляется по ее имени. При этом все обращения к переменной меняются на адрес ячейки памяти, в которой хранится ее значение:

```
1 cout<<C; //из ячейки памяти с именем C будет извлечено значение
2 //и выведено на экран
```

При завершении программы или функции, в которой была описана переменная, память автоматически освобождается.

Доступ к значению переменной можно получить иным способом — определить собственные переменные для хранения адресов памяти. Такие переменные называют *указателями.* С помощью указателей можно обрабатывать массивы, строки и структуры, создавать новые переменные в процессе выполнения программы, передавать адреса фактических параметров.

Указатель — это переменная, значением которой является адрес памяти, по которому хранится объект определенного типа (другая переменная). Например, если **C** — это переменная типа *char,* а **P —** указатель на **C**, значит в **P** находится адрес, по которому в памяти компьютера хранится значение переменной **C**.

Как и любая переменная, указатель должен быть *объявлен*. При объявлении указателей всегда указывается тип объекта, который будет хранится по данному адресу:
тип имя_переменной;
Например:

```
1 int *p //по адресу, записанному в переменной p,
2 //будет хранится переменная типа int
3 //или, другими словами, p указывает на тип данных int
```

Звездочка в описании указателя относится непосредственно к имени, поэтому, чтобы объявить несколько указателей, ее ставят перед именем каждого из них:

C++

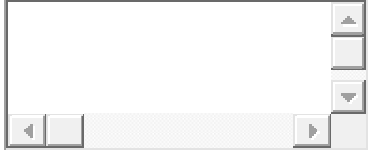
```
1 float *x, y, *z; //описаны указатели на вещественное число - x и z
2 //а также вещественная переменная y
```

Операции * и & при работе с указателями

Операция получения адреса обозначается знаком &, а операция разадресации *. Первая возвращает адрес своего операнда. Например:

```
1 float a; //объявлена вещественная переменная a
2 float *adr_a; //объявлен указатель на тип float
3 adr_a = &a; //оператор записывает в переменную adr_a
4 //адрес переменной a
```

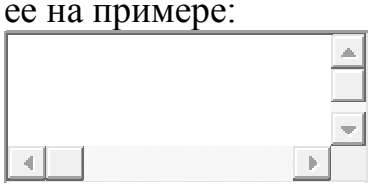
Операция *разадресации* * возвращает значение переменной, хранящееся в по заданному адресу, то есть выполняет действие, обратное операции &:



```
1 float a; //объявлена вещественная переменная a
2 float *adr_a; //объявлен указатель на тип float
3 a = *adr_a; //оператор записывает в переменную a
4 //вещественное значение, хранящиеся по адресу adr_a
```

Присваивание указателей

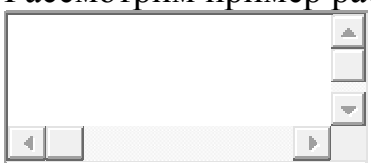
Значение одного указателя можно присвоить другому. Если указатели одного типа, то для этого применяют обычную операцию *присваивания* . Рассмотрим ее на примере:



```
1  #include "stdafx.h"
2  #include <iostream>
3  using namespace std;
4  int main()
5  {setlocale (LC_ALL, "Rus");
6  float PI=3.14159, *p1, *p2;
7  p1=p2=&PI;
8  cout<<"По адресу p1="<<p1<<" хранится *p1="<<*p1<<"\n";
9  cout<<"По адресу p2="<<p2<<" хранится *p2="<<*p2<<"\n";
10 system ("pause");
11 return 0;
12 }
```

В данной программе определены: вещественная переменная **PI**=3.14159 и два указателя на тип **float** — **p1** и **p2**. В указатели записывается адрес переменной **PI.** В результате работы программы в переменных **p1** и **p2** будет хранится значение одного и того же адреса, по которому хранится вещественная переменная **PI=3.14159**.
Результат работы программы:

Если указатели ссылаются на различные типы, то при присваивании значения одного указателя другому, необходимо использовать *преобразование типов*
Без преобразования можно присваивать любому указателю указатель **void*.**
Рассмотрим пример работы с указателями различных типов:



```
1  #include "stdafx.h"
2  #include <iostream>
3  using namespace std;
4  int main()
```

```
5  {
6  setlocale (LC_ALL, "Rus");
7  float PI=3.14159; //объявлена вещественная переменная PI
8  float *p1; //объявлен указатель на float - p1
9  double *p2; //объявлен указатель на double - p2
10 p1=&PI; //переменной p1 присваивается значние адреса PI
11 p2=(double *)p1; //указателю на double присваивается значение,
12 //которое ссылается на тип float
13 cout<<"По адресу p1="<<p1<<" хранится *p1="<<*p1<<"\n";
14 cout<<"По адресу p2="<<p2<<" хранится *p2="<<*p2<<"\n";
15 system ("pause");
16 return 0;
17 }
```

В указателях **p1** и **p2** хранится один и тот же адрес, но значения, на которые они ссылаются, оказываются разными. Это связано с тем, что указатель типа *float адресует 4 байта, а указатель ***double** — 8 байт. После присваивания p2=(double *)p1; при обращении к ***p2** происходит следующее: к переменной, хранящийся по адресу **p1**, дописывается еще 4 следующих байт из памяти. В результате значение ***p2** не совпадает со значением ***p1**. Результат работы программы будет примерно такой:

А вот что произойдет в результате работы следующего кода:

```
1  #include "stdafx.h"
2  #include <iostream>
3  using namespace std;
4  int main()
5  {
6  setlocale (LC_ALL, "Rus");
7  double PI=3.14159, *p1; float *p2;
8  p1=&PI;
9  p2=(float *)p1;
10 cout<<"По адресу p1="<<p1<<" хранится *p1="<<*p1<<"\n";
11 cout<<"По адресу p2="<<p2<<" хранится *p2="<<*p2<<"\n";
12 system ("pause");
13 return 0;
14 }
```

После присваивания **p2=(double*)p1;** при обращении к *p2 происходит следующее: из переменной, хранящийся по адресу **p1**, выделяется только 4 байта. В результате значение ***p2** не совпадает со значением ***p1.**

Результат работы программы:

Таким образом, при преобразовании указателей разного типа приведение типов разрешает не только синтаксическую проблему присваивания. Следует помнить, что операция * над указателями различного типа, ссылающимися на один и тот же адрес, возвращает различные значения.

Практическая работа № 7

Тема : Файлы в С++. Работа с файлами.

Теоретическая часть.

Файловый ввод и вывод

Большинство компьютерных программ работают с файлами. Текстовые процессоры создают файлы документов. Программы баз данных создают и выполняют поиск в информационных файлах. Компиляторы читают файлы исходного кода и генерируют исполняемые файлы. Сам по себе файл — это группа байтов, сохраненных на некотором устройстве, возможно, магнитной ленте, возможно, оптическом диске, дискете или жестком диске. Как правило, операционная система управляет файлами, отслеживая их местоположение, размеры, дату их создания и тому подобное. Если только вы не программируете на уровне операционной системы, обычно вам не нужно заботиться об упомянутых вещах. Все, что вам нужно знать — это способ подключения программы к файлу, способ прочесть в программе его содержимое, и способ создавать и читать файлы внутри программы.

Перенаправление (описанное ранее в настоящей главе) может предоставить некоторую поддержку файлов, но значительно более ограниченную, чем явный ввод- вывод, осуществляемый из программы. К тому же перенаправление обеспечивается операционной системой, а не С++, поэтому оно доступно не во всех системах. В этой книге мы уже касались темы файлового ввода-вывода, но эта глава освещает данную тему более тщательно.

Пакет классов ввода-вывода С++ управляет файловым вводом и выводом в основном так же, как он делает это со стандартным вводом и выводом. Чтобы записывать в файл, вы создаете объект ofstream и используете такие его методы, как операция вставки << или write (). Чтобы читать из файла, вы создаете объект if stream и используете методы istream вроде операции извлечения >> и get (). Однако файлы требуют больше внимания, нежели стандартный ввод и вывод. Например, вы должны ассоциировать вновь открытый файл с потоком. Вы можете открыть файла в режиме только для чтения, только для записи либо для чтения-записи. Если вы записываете в файл, то можете пожелать создать новый, заменить старый либо добавить информацию в существующий файл. Или же вы можете вернуться и пройти по файлу заново. Чтобы помочь в выполнении этих задач, С++ определяет несколько новых классов в заголовочном файле fstream (бывший fstream.h), включая класс ifstream для файлового ввода и класс ofstream для файлового вывода. С++ также определяет класс fstream для совместного файлового ввода-вывода. Эти классы унаследованы от классов из заголовочного файла iostream, поэтому объекты этих новых классов могут использовать методы, которые вы уже изучили ранее.

Простой файловый ввод-вывод

Предположим, вы хотите, чтобы ваша программа записывала в файл. Вы должны сделать следующее:

1. Создать объект ofstream для управления выходным потоком.
2. Ассоциировать этот объект с конкретным файлом.
3. Использовать объект тем же способом, как вы используете cout. Единственным отличием будет то, что вывод направляется в файл вместо экрана.

Чтобы достичь этого, вы должны начать с включения заголовка fstream. Его включение в большинстве, хотя и не во всех реализациях, автоматически включает файл iostream, поэтому вам не обязательно включать явно iostream. Затем вы должны объявить объект типа ofstream:

```
ofstream fout;          // создает объект ofstream по имени fout
```

Именем объекта может быть любое допустимое в C++ имя вроде fout, outFile, cgate или didi.

Затем вы должны ассоциировать этот объект с конкретным файлом. Вы можете сделать это с помощью метода open (). Предположим, например, что вы хотите открыть файл jar для вывода. Вы можете сделать это следующим образом:

```
fout.open("jar.txt"); // ассоциировать fout с jar.txt
```

Вы можете совместить эти два шага (создание объекта и ассоциация файла с ним) в одном операторе, используя другой конструктор:

```
ofstream fout("jar.txt"); //создать объект fout, ассоциировать его с jar.txt
```

После того, как вы все это сделаете, вы сможете использовать fout (или любое другое выбранное вами имя) в той же манере, что и cout. Например, если вы захотите поместить слова Dull Data в этот файл, то можете сделать это следующим образом:

```
fout << "Dull Data";
```

В самом деле, поскольку ostream — это базовый класс для класса ofstream, вы вправе использовать все методы ostream, включая разнообразные операции вставки, а также форматирующие методы и манипуляторы. Класс ofstream использует буферизованный вывод, поэтому программа выделяет пространство для выходного буфера, когда создает объект типа ofstream, подобный fout. Если вы создадите два объекта ofstream, то программа создаст два буфера — по одному для каждого объекта. Объект ofstream, такой как fout, принимает от программы вывод — байт за байтом, а затем, когда буфер наполняется, передает его содержимое в файл назначения. Поскольку дисковые приводы спроектированы так, что передают данные крупными порциями, а не байт за байтом, буферизованный подход значительно повышает скорость передачи данных из программы в файл.

Открытие файла для вывода, таким образом, создает новый файл, если файла с указанным именем не существовало. Если же файл с этим именем существовал ранее, то действие по его открытию усекает его до нулевого

размера, так что вывод начинается в пустой файл. Позднее в этой главе вы увидите, как открыть существующий файл и сохранить его содержимое.

Внимание!

тЯ ^{0тк}Р^{ытие} Ф^{айла} Для вывода в режиме по умолчанию автоматически усекает его до нулевого fi-JF размера, что уничтожает его предыдущее содержимое.

Требования к чтению файла очень похожи на требования, которые нужно выполнить для записи в файл:

1. Создать объект ifstream для управления входным потоком.
2. Ассоциировать этот объект с конкретным файлом.
3. Использовать объект тем же способом, что используется cin.

Шаги для чтения файла похожи на шаги, которые нужно выполнить для его записи. Во-первых, конечно, вы включаете заголовочный файл fstream. Затем вы объявляете объект ifstream и ассоциируете его с именем файла. Это можно сделать в двух операторах или же в одном:

```
// два оператора
ifstream fin; // создать объект ifstream по имени fin
fin.open("jellyjar.dat"); // открыть jellyjar.dat для чтения
/ / один оператор
ifstream fis("jamjar.dat");//создать fis и ассоциировать его с jamjar.dat
```

Затем вы можете использовать fin или fis почти так же, как используете cin. Например, можно поступить следующим образом:

```
char ch;
fin >> ch;              // прочесть символ из файла jellyjar.dat
char buf [80];
fin » buf;              // прочесть слово из файла
fin.getline(buf, 80); // прочесть строку из файла
string line;
getline(fin, line); // прочесть из файла в строковый объект
```

Ввод, как и вывод, также буферизуется, поэтому создание объекта ofstream, такого как fin, создает входной буфер, которым управляет объект fin. Как и в случае вывода, буферизация перемещает данные гораздо быстрее, чем передача байт за байтом.

Соединение с файлом закрывается автоматически, когда объекты ввода и вывода уничтожаются, например, по завершении программы. Кроме того, вы можете закрыть соединение с файлом явно, используя для этого метод close ():

```
fout.close();          // закрыть вывод, подключенный к файлу
fin.close();           // закрыть ввод, подключенный к файлу
```

Закрытие подключения не уничтожает поток; он просто отключается от файла. Однако средства управления потоком остаются на месте. Например, объект fin продолжает существовать вместе с входным буфером, которым он управляет. Как вы вскоре увидите, этот поток можно подключить заново к тому же файлу либо к другому.

Рассмотрим краткий пример. Программа в листинге fileio.cpp запрашивает

имя файла. Она создает файл с этим именем, пишет некоторую информацию в него и закрывает файл. Закрытие файла сбрасывает буфер, тем самым гарантируя обновление файла. Затем программа открывает тот же файл для чтения и отображает его содержимое. Отметим, что программа использует имена fin и fout в той же манере, что и если бы вы применяли cin и cout. Также программа читает имя файла в объект string и использует метод c str () для обеспечения аргумента в виде строки стиля C для конструкторов ofstream и ifstream.

Листинг fileio.cpp

```
// fileio.cpp — сохранение в файле
tinclude <iostream> // для многих систем не требуется
#include <fstream> tinclude <string> int main()
{
   using namespace std; string filename;
   cout << "Введите имя нового файла: "; cin » filename;
   // создать объект выходного потока для нового файла и назвать его
   fout ofstream fout(filename.c_str () ) ;
   fout « "Только для ваших глаз!\n"; // писать в файл cout <<
   "Введите секретное число: "; // писать на экран float secret; cin >>
   secret;
   fout « "Ваше секретное число " << secret << endl; fout.close (); //
   закрыть файл
   // создать объект входного потока для нового файла и назвать его fin
   ifstream fin(filename.c_str());
   cout « "Вот содержимое " « filename << ":\n";
   char ch;
   while (fin.get(ch)) // читать символы из файла cout << ch; //и писать
   их на экран cout << "Готово.\n"; fin.close (); return 0;
}
```

Пример запуска программы из листинга fileio.cpp:

```
Введите имя файла: pythag Введите секретное число: 3.14159 Вот
содержимое pythag:
Только для ваших глаз!
Ваше секретное число 3.14159 Готово.
```

Если вы просмотрите каталог, содержащий программу, то найдете там файл по имени pythag и, загрузив его в любой текстовый редактор, увидите то же содержимое, что показал вывод программы.

Практическая часть

1. Напишите программу, которая подсчитывает количество символов вплоть до первого $ в строке, оставляя $ во входном потоке.

2. Напишите программу, которая копирует ваш клавиатурный ввод (вплоть до эмулируемого конца файла) в файл, чье имя передано в командной строке.
3. Напишите программу, копирующую один файл в другой. Имена файлов программа должна получать из командной строки. Если не удается открыть файл, должно выдаваться соответствующее сообщение.
4. Напишите программу, которая открывает два текстовых файла для ввода и один — для вывода. Программа должна соединять соответствующие строки входных файлов, используя в качестве разделителя пробел, и писать результаты в выходной файл. Например, предположим, что первый входной файл имеет следующее содержимое:

eggs kites donuts balloons
hammers stones

А второй файл — следующее:
zero lassitude finance drama

Результирующий файл должен выглядет так:
eggs kites donuts zero lassitude balloons hammers
finance drama stones

Практическая работа №8.

Тема: Классы. Виды классов.

Теоретическая часть.

Механизм классов в C++ позволяет пользователям определять собственные типы данных. По этой причине их часто называют пользовательскими типами. Класс может наделять дополнительной функциональностью уже существующий тип. Так, например, IntArray, введенный в главе 2, предоставляет больше возможностей, чем тип "массив int". С помощью классов можно создавать абсолютно новые типы, например Screen (экран) или Account (расчетный счет). Как правило, классы используются для абстракций, не отражаемых встроенными типами адекватно.

В этой главе мы узнаем, как определять типы и использовать объекты классов; увидим, что определение класса вводит как данные-члены, описывающие его, так и функции-члены, составляющие набор операций, применимых к объектам класса. Мы покажем, как можно обеспечить сокрытие информации, объявив внутреннее представление и реализацию закрытыми, но открыв операции над объектами. Говорят, что закрытое внутреннее представление инкапсулировано, а открытую часть класса называют его интерфейсом.

Далее в этой главе мы познакомимся с особым видом членов класса – статическими членами. Мы расскажем также, как можно использовать указатели на члены и функции-члены класса, и рассмотрим объединения, представляющие собой специализированный вид класса для хранения объектов разных типов в одной области памяти. Завершается глава обсуждением области видимости класса и описанием правил разрешения имен в этой области; затрагиваются такие понятия, как вложенные классы, классы-члены пространства имен и локальные классы.

Определение класса

Определение класса состоит из двух частей: заголовка, включающего ключевое слово class, за которым следует имя класса, и тела, заключенного в фигурные скобки. После такого определения должны стоять точка с запятой или список объявлений:

```
class Screen { /* ... */ };
class Screen { /* ... */ } myScreen, yourScreen;
```

Внутри тела объявляются данные-члены и функции-члены и указываются уровни доступа к ним. Таким образом, тело класса определяет список его членов.

Каждое определение вводит новый тип данных. Даже если два класса имеют одинаковые списки членов, они все равно считаются разными типами:

```
class First {
  int memi;
  double memd;
};

class Second {
  int memi;
  double memd;
};

class First obj1;
Second obj2 = obj1;   // ошибка: obj1 и obj2 имеют разные типы
```

Тело класса определяет отдельную область видимости. Объявление членов внутри тела помещает их имена в область видимости класса. Наличие в двух разных классах членов с одинаковыми именами – не ошибка, эти имена относятся к разным объектам. (Подробнее об областях видимости классов мы поговорим в разделе 13.9.)

После того как тип класса определен, на него можно ссылаться двумя способами:

- написать ключевое слово class, а после него – имя класса. В предыдущем примере объект obj1 класса First объявлен именно таким образом;
- указать только имя класса. Так объявлен объект obj2 класса Second из приведенного примера.

Оба способа сослаться на тип класса эквивалентны. Первый заимствован из языка C и остается корректным методом задания типа класса; второй способ введен в C++ для упрощения объявлений.

Данные-члены

Данные-члены класса объявляются так же, как переменные. Например, у класса Screen могут быть следующие данные-члены:

```
#include
class Screen {
  string           _screen;   // string( _height * _width )
  string::size_type  _cursor;   // текущее положение на экране
  short            _height;   // число строк
  short            _width;    // число колонок
};
```

Поскольку мы решили использовать строки для внутреннего представления объекта класса Screen, то член _screen имеет тип string. Член _cursor – это смещение в строке, он применяется для указания текущей позиции на экране. Для него использован переносимый тип string::size_type. (Тип size_type рассматривался в разделе 6.8.)

Необязательно объявлять два члена типа short по отдельности. Вот объявление класса Screen, эквивалентное приведенному выше:

```
class Screen {
/*
 * _ screen адресует строку размером _height * _width
 * _cursor указывает текущую позицию на экране
 * _height и _width - соответственно число строк и колонок
 */
  string           _screen;
  string::size_type  _cursor;
  short            _height, _width;
};
```

Член класса может иметь любой тип:

```
class StackScreen {
  int topStack;
  void (*handler)();     // указатель на функцию
  vector stack;  // вектор классов
};
```

Описанные данные-члены называются нестатическими. Класс может иметь также и статические данные-члены. (У них есть особые свойства, которые мы рассмотрим в разделе 13.5.)

Объявления данных-членов очень похожи на объявления переменных в области видимости блока или пространства имен. Однако их, за исключением статических членов, нельзя явно инициализировать в теле класса:

```
class First {
  int    memi = 0;    // ошибка
  double memd = 0.0;  // ошибка
};
```

Данные-члены класса инициализируются с помощью конструктора класса. (Мы рассказывали о конструкторах в разделе 2.3; более подробно они рассматриваются в главе 14.)

Функции-члены

Пользователям, по-видимому, понадобится широкий набор операций над объектами типа Screen: возможность перемещать курсор, проверять и устанавливать области экрана и рассчитывать его реальные размеры во время выполнения, а также копировать один объект в другой. Все эти операции можно реализовать с помощью функций-членов.

Функции-члены класса объявляются в его теле. Это объявление выглядит точно так же, как объявление функции в области видимости пространства имен. (Напомним, что глобальная область видимости – это тоже область видимости пространства имен. Глобальные функции рассматривались в разделе 8.2, а пространства имен – в разделе 8.5.) Например:

```
class Screen {
public:
  void home();
  void move( int, int );
  char get();
  char get( int, int );
  void checkRange( int, int );
  // ...
};
```

Определение функции-члена также можно поместить внутрь тела класса:

```
class Screen {
public:
  // определения функций home() и get()
  void home() { _cursor = 0; }
  char get() { return _screen[_cursor]; }
  // ...
};
```

home() перемещает курсор в левый верхний угол экрана; get() возвращает символ, находящийся в текущей позиции курсора.

Функции-члены отличаются от обычных функций следующим:

- функция-член объявлена в области видимости своего класса, следовательно, ее имя не видно за пределами этой области. К функции-члену можно обратиться с помощью одного из операторов доступа к членам – точки (.) или стрелки (->):

  ```
  ptrScreen->home();
  myScreen.home();
  ```

(в разделе 13.9 область видимости класса обсуждается более детально);

- функции-члены имеют право доступа как к открытым, так и к закрытым членам класса, тогда как обычным функциям доступны лишь открытые. Конечно, функции-члены одного класса, как правило, не имеют доступа к данным-членам другого класса.

Функция-член может быть перегруженной (перегруженные функции рассматриваются в главе 9). Однако она способна перегружать лишь другую функцию-член своего класса. По отношению к функциям, объявленным в других классах или пространствах имен, функция-член находится в отдельной области видимости и, следовательно, не может перегружать их. Например, объявление get(int, int) перегружает лишь get() из того же класса Screen:

```
class Screen {
public:
  // объявления перегруженных функций-членов get()
  char get() { return _screen[_cursor]; }
  char get( int, int );
  // ...
};
```

(Подробнее мы остановимся на функциях-членах класса в разделе 13.3.)

Доступ к членам

Часто бывает так, что внутреннее представление типа класса изменяется в последующих версиях программы. Допустим, опрос пользователей нашего класса Screen показал, что для его объектов всегда задается размер экрана 80 ? 24. В таком случае было бы желательно заменить внутреннее представление экрана менее гибким, но более эффективным:

```
class Screen {
public:
  // функции-члены
private:
  // инициализация статических членов (см. 13.5)
  static const int   _height = 24;
  static const int   _width = 80;
  string             _screen;
  string::size_type  _cursor;
};
```

Прежняя реализация функций-членов (то, как они манипулируют данными-членами класса) больше не годится, ее нужно переписать. Но это не означает, что должен измениться и интерфейс функций-членов (список формальных параметров и тип возвращаемого значения).

Если бы данные-члены класса Screen были открыты и доступны любой функции внутри программы, как отразилось бы на пользователях изменение внутреннего представления этого класса?

- все функции, которые напрямую обращались к данным-членам старого представления, перестали бы работать. Следовательно, пришлось бы отыскивать и изменять соответствующие части кода;
- так как интерфейс не изменился, то коды, манипулировавшие объектами класса Screen только через функции-члены, не пришлось бы модифицировать. Но поскольку сами функции-члены все же изменились, программу пришлось бы откомпилировать заново.

Сокрытие информации – это формальный механизм, предотвращающий прямой доступ к внутреннему представлению типа класса из функций программы. Ограничение доступа к членам задается с помощью секций тела класса, помеченных ключевыми словами public, private и protected – спецификаторами доступа. Члены, объявленные в секции public, называются открытыми, а объявленные в секциях private и protected соответственно закрытыми или защищенными.

- открытый член доступен из любого места программы. Класс, скрывающий информацию, оставляет открытыми только функции-члены, определяющие операции, с помощью которых внешняя программа может манипулировать его объектами;
- закрытый член доступен только функциям-членам и друзьям класса. Класс, который хочет скрыть информацию, объявляет свои данные-члены закрытыми;
- защищенный член ведет себя как открытый по отношению к производному классу и как закрытый по отношению к остальной части программы. (В главе 2 мы видели пример использования защищенных членов в классе IntArray. Детально они рассматриваются в главе 17, где вводится понятие наследования.)

В следующем определении класса Screen указаны секции public и private:

```
class Screen {
public:
  void home() { _cursor = 0; }
  char get() { return _screen[_cursor]; }
  char get( int, int );
  void move( int, int );
  // ...
private:
  string           _screen;
  string::size_type  _cursor;
  short            _height, _width;
```

```
};
```

Согласно принятому соглашению, сначала объявляются открытые члены класса. (Обсуждение того, почему в старых программах C++ сначала шли закрытые члены и почему этот стиль еще кое-где сохранился, см. в книге [LIPPMAN96a].) В теле класса может быть несколько секций public, protected и private. Каждая секция продолжается либо до метки следующей секции, либо до закрывающей фигурной скобки. Если спецификатор доступа не указан, то секция, непосредственно следующая за открывающей скобкой, по умолчанию считается private.

Лабораторная работа №1

Тема : Создание линейных программ на С ++.

1.1. Цель лабораторной работы

1.Освоить создание программ с простейшим линейным алгоритмом.
2.Освоить работу с интегрированной средой Borland C++.
3.Освоить работу с отладчиком.
4.Ознакомится с организацией контролирующих программ.

1.2. Используемое программное обеспечение

Для выполнения лабораторной работы используется Borland C версия 3.1 и программа генерации задания lab1.exe.

1.3. Порядок выполнения лабораторной работы

1.Запустить программу генерации задания lab1 и Borland C++.
2.Зарегистрироваться в контролирующей программе.
3.Получить задание (генерируется контролирующей программой).
4.Составить программу.
5.Проверить правильность работы программы с помощью контролирующей программы и ручного расчета.
6.Проверить работу программы в отладчике.

Теоретическая часть.

Основные типы данных в языке C++ делятся на две группы. Одна группа представляет значения, которые хранятся в виде целых чисел. Другая группа представляет значения, которые хранятся в формате с плавающей точкой. Целочисленные типы отличаются друг от друга по количеству памяти, которое отводится для хранения значений, а также по тому, имеют они знак или нет. Целочисленными типами являются следующие (в порядке возрастания диапазона представляемых значений): bool, char, signed char, unsigned char, short, unsigned short, int, unsigned int, long и unsigned long. Существует также тип wchar t, чье положение среди них зависит от реализации. C++ гарантирует, что тип char имеет достаточно большой размер, чтобы хранить любой член расширенного набора символов в системе, тип short имеет как минимум 16 битов, int как минимум такой же, как short, а long имеет минимум 32 бита и как минимум такой же, как и int. Точный размер зависит от реализации.

Представление символов осуществляется посредством их числовых кодов. Система ввода-вывода определяет, соответствует ли код цифре или символу.

Типы с плавающей точкой могут представлять дробные значения и

значения намного больше тех, которые могут быть представлены целыми типами. Существует три таких типа: float, double и long double. C++ гарантирует, что тип float не меньше, чем тип double, и что тип double не больше типа long double. Обычно тип float использует 32 бита памяти, double использует 64 бита, а long double использует от 80 до 128 битов.

Для решения конкретной задачи вы можете выбрать наиболее подходящий тип данных, поскольку язык C++ предлагает различные типы данных, характеризующиеся различными диапазонами представления чисел, и могут быть как знаковыми, так и беззнаковыми.

Над числовыми типами в языке C++ можно выполнять следующие арифметические операции: сложение, вычитание, умножение, деление и нахождение остатка целочисленного деления. Если две операции соперничают за одно значение, то первоочередность выполнения операции определяется в соответствии с правилами приоритета и ассоциативности.

C++ преобразовывает значение одного типа в другой, когда вы присваиваете значение переменной, комбинируете разные типы в арифметических операциях и используете приведение типов для принудительного преобразования типов. Многие преобразования типов являются “безопасными”, то есть они могут быть выполнены без потери или изменения данных. Например, вы можете преобразовать значение int в значение long без каких-либо проблем. Другие преобразования, например, преобразование типов с плавающей точкой в целые типы, требуют осторожности.

На первый взгляд вам может показаться, что в языке C++ имеется чересчур много базовых типов значений, особенно если принять во внимание различные правила преобразования. И все же, из всего разнообразия типов вы сможете выбирать именно такой, который будет наиболее всего соответствовать вашим требованиям.

Практическая часть.

1. Напишите короткую программу, которая выдавала бы запрос на ввод роста в целых дюймах и преобразовывала бы их в футы и дюймы. Программа должна использовать символ подчеркивания для обозначения позиции, с которой начинается ввод значений. Используйте также символьную константу const для представления коэффициента преобразования.
2. Напишите короткую программу, которая выдавала бы запрос на ввод значения роста в футах и дюймах и веса в фунтах. (Для хранения этой информации используйте три переменных.) Программа должна выдать индекс массы тела (BMI, body mass index). Чтобы рассчитать индекс, сначала преобразуйте высоту в футах и дюймах в высоту в дюймах (1 фут =12 дюймов). Затем преобразуйте вес в фунтах в массу в килограммах, разделив на 2.2. После этого рассчитайте свой индекс,

разделив массу в килограммах на квадрат вашего роста в метрах. Используйте символьные константы для представления различных коэффициентов преобразования.

3. Напишите программу, которая выдавала бы запрос на ввод широты в градусах, минутах и секундах, после чего отображала бы широту в десятичном формате. В одной минуте 60 секунд, а в одном градусе 60 минут; представьте эти значения посредством символьных констант. Для каждого вводимого значения следует использовать отдельную переменную. Пример результата выполнения программы выглядит следующим образом:

   Enter a latitude in degrees, minutes, and seconds:
   First, enter the degrees: 37
   Next, enter the minutes of arc: 51
   Finally, enter the seconds of arc: 19
   37 degrees, 51 minutes, 19 seconds = 37.8553 degrees

4. Напишите программу, которая выдавала бы запрос на ввод количества секунд в виде целого значения (используйте тип **long),** и затем отображала бы эквивалентное значение в сутках, часах, минутах и секундах. Для представления количества часов в сутках, количества минут в часе и количества секунд в минуте используйте символьные константы. Пример результата выполнения программы выглядит следующим образом:

   Enter the number of seconds: 31600000
   31600000 seconds = 365 days, 46 minutes, 40 seconds

5. Напишите программу, которая выдавала бы запрос на ввод миль, которые вы преодолели на автомобиле, и количества галлонов израсходованного бензина, а затем выдавала бы отчет о том, сколько миль вы преодолели, израсходовав один галлон бензина. Или, если хотите, программа может запросить расстояние в километрах и количество литров бензина, и затем представить отчет в европейском стиле — в количестве литров, израсходованных на один километр.

Напишите программу, которая выдавала бы запрос на ввод расхода бензина в европейском стиле (количество литров на 100 км) и преобразовывала бы его в стиль, принятый в США, — в милях на один галлон. Имейте в виду, что кроме использования других единиц измерения в США, в отличие от европейских стран, принято и другое соотношение: расстояние/топливо, а не топливо/расстояние. Учтите, что 100 километров соответствуют 62.14 милям, а 1 галлон равен 3.875 литрам. Таким образом, 19 миль/галлон примерно равно 12.4 литров на 100 км, а 27 миль/галлон примерно составляет 8.7 литров на 100 км.

Лабораторная работа №2

Тема :Разработка простой программы на С ++. Стандартные математические функции языка С ++

1. Цель лабораторной работы

1.Освоить механизм рекурсии.

2.Освоить возвращение из функции числовых значений.

1.2.Используемое программное обеспечение

Для выполнения лабораторной работы используется Borland C: версия 3.1 и контролирующая программа lab 2.exe.

1.3. Порядок выполнения лабораторной работы

Запустить программу генерации задания lab 2 и Borland C. 1.
Зарегистрироваться в контролирующей программе.
2. Получить задание (генерируется контролирующей программой).
3. Составить программу.
4. Проверить правильность работы программы с помощью контролирующей программы и ручного расчета.
5. Проверить работу программы в отладчике.

Теоретическая часть.

Программа на языке С++ состоит из одного или нескольких модулей, называемых функциями. Выполнение программы начинается с функции main () (все символы — в нижнем регистре), поэтому ваша программа обязательно должна включать эту функцию. Функция состоит из заголовка и тела. В заголовке функции указывается, каким является возвращаемое значение (если таковое существует), генерируемое функцией, и какую информацию принимает функция в виде аргументов. Тело функции состоит из последовательности операторов языка С++, заключенных в фигурные скобки ({ }).

В языке программирования С++ выделяют следующие типы операторов:

- Оператор объявления. В операторе объявления указывается имя и тип переменной, которая используется в функции.
- Оператор присваивания. Этот оператор использует операцию присваивания (=) для присваивания значения переменной.
- Оператор сообщений. Оператор сообщений посылает сообщение объекту, инициируя некоторое действие.
- Вызов функции. Вызов функции активизирует ее работу. Когда вызываемая функция завершает свою работу, программа возвращается к оператору в вызывающей функции, следующим за вызовом функции.

- Прототип функции. В прототипе функции объявляется тип возвращаемого функцией значения, а также количество и тип аргументов, передаваемых функции.
- Оператор возврата. Оператор возврата посылает значение из вызываемой функции обратно вызывающей функции. Класс представляет собой определяемую пользователем спецификацию типа данных. В ней подробно описывается способ представления информации и действия, которые могут выполняться над этими данными. Объект — это сущность, созданная в соответствии с предписанием класса, а простая переменная является сущностью, созданной в соответствии с описанием типа данных.

В языке C++ имеются два предварительно определенных объекта (cin и cout) для обработки ввода и вывода. Они являются примерами классов istream и ostream, которые определены в файле iostream. Эти классы рассматривают ввод и вывод в виде потоков символов. Операция вставки (<<), которая определена для класса ostream, позволяет помещать данные в поток вывода, а операция извлечения (>>), которая определена для класса istream, позволяет извлекать информацию из потока ввода. Как cin, так и cout являются интеллектуальными объектами, способными автоматически преобразовывать информацию из одной формы в другую в соответствии с контекстом программы.

Язык программирования C++ может использовать обширный набор библиотечных функций языка C. Чтобы использовать библиотечную функцию, необходимо включить заголовочный файл, который предоставляет для функции ее прототип.

Теперь, когда вы уже знаете, что собой представляют простые программы на языке C++, можно приступить к изучению следующей главы, вникая во все более широкий круг вопросов.

Практическая часть

7. Напишите программу на C++, которая отобразит вашу фамилию и почтовый адрес на экране монитора.
8. Напишите программу на C++, которая выдает запрос на ввод расстояния в фар- лонгах и преобразовывает его в ярды. (Один фарлонг равен 220 ярдам.)
9. Напишите программу на C++, которая использует три определяемых пользователем функции (включая main ()), и результатом ее выполнения является следующий вывод:

Three blind mice Three blind
mice See how they run See how
they run

Одна функция, вызываемая два раза, должна генерировать первые две строки, а вторая функция, также вызываемая два раза, должна генерировать оставшиеся строки.

10. Напишите программу, в которой функция main () вызывает определяемую пользователем функцию, которая в качестве аргумента принимает значение температуры по Цельсию и возвращает эквивалентное значение температуры по Фаренгейту. Программа должна выдать запрос на ввод значения по Цельсию и отобразить следующий результат:
Please enter a Celsius value: 20
20 degrees Celsius is 68 degrees Fahrenheit.

Для справки, формула для выполнения этого преобразования:
Температура в градусах по Фаренгейту = 1,8 * Температура в градусах по Цельсию + 32

11. Напишите программу, в которой функция main () вызывает определяемую пользователем функцию, которая в качестве аргумента принимает расстояние в световых годах и возвращает расстояние в астрономических единицах. Программа должна выдать запрос на ввод значения светового года и отобразить следующий результат:
Enter the number of light years: 4.2
4.2 light years = 265608 astronomical units.

Астрономическая единица равна среднему расстоянию Земли от Солнца (около 150 000 000 км, или 93 000 000 миль), а световой год соответствует расстоянию, пройденному лучом света за один земной год (примерно 10 триллионов километров, или 6 триллионов миль). (Ближайшая звезда после Солнца находится на расстоянии 4.2 световых года.) Используйте тип double (как в листинге 2.4) и следующий коэффициент преобразования:
1 световой год = 63 240 астрономических единиц

12. Напишите программу, которая выдает запрос на ввод значения часов и значения минут. Функция main () должна передать эти два значения функции, имеющей тип void, которая отображает эти два значения в следующем виде:
Enter the number of hours: 9 Enter the number of minutes: 28 Time: 9:28

Лабораторная работа № 3

Тема: Условия в С++. Создание ветвлений в программе.

1.Цель лабораторной работы

1.Освоить создание программ с условия в С++.
2.Освоить работу с интегрированной средой Borland C++.
3.Освоить работу с отладчиком.
4.Ознакомится с организацией контролирующих программ.

1.2.Используемое программное обеспечение

Для выполнения лабораторной работы используется Borland C версия 3.1 и программа генерации задания lab3.exe.

1.3.Порядок выполнения лабораторной работы

1.Запустить программу генерации задания lab3 и Borland C++.
2.Зарегистрироваться в контролирующей программе.
3.Получить задание (генерируется контролирующей программой).
4.Составить программу.
5.Проверить правильность работы программы с помощью контролирующей программы и ручного расчета.
6.Проверить работу программы в отладчике.

Теоретическая часть.

Программы и процесс программирования становятся более интересными, когда вводятся операторы, которые позволяют программе выбрать альтернативные действия. В С++ имеются операторы if, if else, а также switch, представляющие собой средства управления выбором пути выполнения, i f позволяет программе выполнить оператор или блок операторов в случае удовлетворения некоторого условия. То есть программа выполняет этот оператор или блок, только если конкретное условие истинно, if else позволяет программе выбрать для выполнения одно из двух операторов или блоков. Вы можете добавлять дополнительные операторы if else для представления серии вариантов выбора. Оператор С++ switch направляет поток управления программы в определенное место из списка возможных.

В С++ также доступны операции, помогающие принимать решения. В главе 5 обсуждаются выражения отношений, которые сравнивают два значения, if и if else обычно в качестве проверочных условий используют выражения сравнения. Используя логические операции С++ (&&, | | и !), вы можете комбинировать или модифицировать выражения сравнения для конструирования более сложных тестов. Условная операция (? :) предлагает компактный способ выбора одного из двух значений по условию.

Библиотека символьных функций cctype предлагает удобный и мощный

набор инструментов анализа символьного ввода.

Циклы и операторы выбора — это полезные инструменты для организации файлового ввода-вывода, который во многом повторяет консольный. После того как вы объявляете объекты ifstream и ofstream и ассоциируете их с файлами, их можно использовать в той же манере, что и стандартные cin и cout.

Используя циклы и условные операторы C++, вы можете писать интересные, интеллектуальные и мощные программы. Но мы только начали исследовать реальную мощь языка C++. Далее мы обратимся к функциям.

Практическая часть.

7. Напишите программу, которая читает клавиатурный ввод до символа @ и повторяет его, за исключением десятичных цифр, преобразуя каждую букву верхнего регистра в букву нижнего регистра и наоборот. (Не забудьте о семействе cc type.)
8. Напишите программу, читающую в массив double до 10 значений пожертвований. Программа должна прекращать ввод при получении нечисловой величины. Она должна выдавать среднее значение полученных чисел, а также количество значений в массиве, превышающих среднее.
9. Напишите предшественник программы, управляемой меню. Она должна отображать меню из четырех пунктов, каждый из них помечен буквой. Если пользователь вводит букву, отличающуюся от четырех допустимых, программа должна повторно приглашать его ввести правильное значение до тех пор, пока он этого не сделает. Затем она должна выполнить некоторое простое действие на основе пользовательского выбора. Работа программы должна выглядеть примерно так:
   Пожалуйста, введите одно из следующих значений: c)
   хищник p) пианист t) дерево д) игра **f**
   Пожалуйста, введите ac, p, t, или g: q
   Пожалуйста, введите a c, p, t, или g: **t** Клен —
   это дерево.
10. Когда вы вступите в Благотворительный Орден Программистов (БОП), к вам могут обращаться на заседаниях БОП по вашему настоящему имени, по должности либо секретному имени БОП. Напишите программу, которая может выводить списки членов по настоящим именам, должностям, секретным именам либо по предпочтению самого члена. В основу положите следующую структуру:

```
// Структура имен Благотворительного Ордена Программистов (БОП)
struct bop {
    char fullname[strsize]; // настоящее имя
    char title[strsize];        // должность
    char bopname[strsize]; // секретное имя БОП
    int preference;             // 0 = полное имя, 1 = титул, 2 = имя БОП
```

};

В этой программе создайте небольшой массив таких структур и инициализируйте его соответствующими значениями. Пусть программа запустит цикл, который даст возможность пользователю выбирать разные альтернативы:

a. отображать по именам Ь. Отображать по должностям
c. отображать по именам БОП d. отображать по предпочтениям q.
выйти

Обратите внимание, что “отображать по предпочтениям” — не значит, что нужно отобразить предпочтение члена; это значит, что нужно отобразить значение того поля структуры, которое соответствует предпочтению. Например, если preference равно 1, то выбор d должен вызвать отображение должности для данного программиста. Пример запуска этой программы может выглядеть примерно так:

Отчет о Благотворительно Ордене Программистов
a. отображать по именам Ь. Отображать по должностям
c. отображать по именам БОП d. отображать по предпочтениям
q. выйти
Ваш выбор: a
Wimp Macho
Raki Rhodes
Celia Laiter
Hoppy Hipman
Pat Hand
Следующий выбор: d Wimp
Macho Junior Programmer
MIPS
Analyst Trainee LOOPY
Следующий выбор: q Пока!

11. Королевство Нейтрония, где денежной единицей служит тварп, использует следующую шкалу налогообложения:

Первые 5 ООО тварпов — налог 0%
Следующие 10 ООО тварпов — налог 10%
Следующие 20 000 тварпов — налог 15%
Свыше 35 000 тварпов — налог 20%

Например, если некто зарабатывает 38 ООО тварпов, то он должен заплатить налогов 5000 х 0.00 + 10000 х 0.10 + 20000 х 0.15 + 3000 х 0.20, или 4 600 тварпов. Напишите программу, которая использует цикл для запроса доходов и выдачи подлежащего к выплате налога. Цикл должен прерываться, когда пользователь вводит отрицательное или нечисловое значение.

12. Скомпонуйте программу, которая отслеживает пожертвования в Общество Защиты Влиятельных Лиц. Она должна запрашивать у пользователя количество меценатов, а затем приглашать вводить их имена и суммы пожертвований от каждого. Информация должна сохраняться в

динамически выделенном массиве структур. Каждая структура должна иметь два члена: символьный массив (или объект string) для хранения имени и переменную-член типа double — для хранения суммы пожертвования. После чтения всех данных программа должна отображать имена и суммы пожертвований тех, кто не пожалел $10 000 и более. Этот список должен быть озаглавлен меткой Grand Patrons. После этого программа должна выдать список остальных жертвователей. Он должен быть озаглавлен Patrons. Если в какой-то из двух категорий не окажется никого, программа должна напечатать “попе”. Помимо отображения двух категорий, никакой другой сортировки делать не нужно.

Лабораторная работа № 4

Тема : Циклы. Разработка программ с использованием циклов. Виды циклов.

1.Цель лабораторной работы

1.Освоить создание циклических программ.
2.Освоить работу с двух мерными массивами и текстовыми строками.

2.Используемое программное обеспечение

Для выполнения лабораторной работы используется Borland C версия 3.1 и программа генерации задания lab4.exe.

3. Порядок выполнения лабораторной работы

Запустить программу генерации задания lab4 и Borland C.
1.Зарегистрироваться в контролирующей программе.
2.Получить задание (генерируется контролирующей программой).
3.Составить программу.
4.Проверить правильность работы программы с помощью контролирующей программы и ручного расчета.
5.Проверить работу программы в отладчике.

Теоретическая часть.

В C++ представлены три варианта циклов: for, while и do while. Цикл позволяет повторно выполнять один и тот же набор инструкций до тех пор, пока проверочное условие цикла оценивается как true или не ноль, и цикл прекращает их выполнение, когда это проверочное условие возвращает false или 0. Циклы for и while — циклы с проверкой на входе, это означает, что они оценивают проверочное условие перед выполнением операторов, находящихся в теле цикла. Цикл do while проверяет условие на выходе, то есть после выполнения операторов, содержащихся в его теле.
Синтаксис каждого цикла позволяет размещать в теле только один оператор. Однако этот оператор может быть составным, или блоком в виде группы операторов, заключенных в фигурные скобки.
Сравнивающие выражения (выражения отношений), которые сравнивают два значения, часто применяются в качестве проверочных условий цикла. Эти выражения формируются с использованием одной из шести операций отношений: <, <=, ==, >=, > или !=. Сравнивающие выражения возвращают значения типа bool: true или false.

Многие программы читают текстовый ввод или текстовые файлы символ за символом. Класс istream представляет несколько способов сделать. Если ch — переменная типа char, то оператор cin >> ch; читает очередной символ ввода в ch. Однако при этом пропускаются пробелы, переносы строки и символы табуляции. Вызов функции-члена
cin.get (ch);
читает очередной входной символ, независимо от его значения, и помещает его в ch. Вызов функции-члена cin.get () возвращает следующий символ ввода, включая пробелы, переносы строк и символы табуляции, поэтому он может быть использован следующим образом:
ch = cin.get ();
Функция-член cin.get (char) сообщает о встреченном состоянии EOF, возвращая значение, преобразуемое в тип bool, как false, в то время как функция-член cin.get () сообщает о EOF, возвращая значение EOF, определенное в файле заголовка iostream.
Вложенный цикл — это цикл, находящийся в теле другого цикла for. Вложенные циклы обеспечивают естественный способ обработки двумерных массивов.

Практическая часть.

Напишите программу, запрашивающую у пользователя ввести два целых числа. Затем программа должна вычислить и выдать сумму всех целых чисел, лежащих между этими двумя целыми. Предполагается, что меньшее значение введено первым. Например, если пользователь ввел 2 и 9, программа должна сообщить, что сумма целых от 2 до 9 равна 44.
Напишите программу, которая приглашает пользователя вводить числа. После каждого введенного значения программа должна выдавать накопленную сумму введенных значений. Программа должна завершаться при вводе 0.
Дафна инвестировала $100 под простых 10%. То есть, каждый год инвестиция должна приносить 10% инвестированной суммы, то есть $10 каждый год:
прибыль = 0.1 x исходный баланс
В то же время Клео инвестировала $100 под сложных 5%. То есть прибыль составит 5% от текущего баланса, включая предыдущую накопленную прибыль: прибыль - 0.05 x текущий баланс
Клео зарабатывает 5% от $100 в первый год, что дает ей $105. На следующий год она зарабатывает 5% от $105, что составляет $5.25, и так далее. Напишите программу, которая вычислит, сколько лет

понадобится для того, чтобы сумма баланса Клео превысила сумму баланса Дафны, с отображением значений обоих балансов за каждый год.

Вы продаете книгу “Язык C++ для чайников”. Напишите программу, которая позволит ввести помесячные объемы продаж в течение года (в количестве книг, а не в деньгах). Программа должна использовать цикл, в котором выводится приглашение с названием месяца, применяя массив указателей на char (или массив объектов string, если вы предпочитаете его), инициализированный строками — названиями месяцев, и сохраняя введенные значения в массиве int. Затем программа должна найти сумму содержимого массива и выдать общий объем продаж за год.

Выполните упражнение 4, но используя двумерный массив для сохранения данных о месячных продажах за 3 года. Выдайте общую сумму продаж за каждый год и за все годы вместе.

Лабораторная работа № 5

Тема: Оператор выбора switch. Switch как сложный оператор условного перехода.

1.Цель лабораторной работы

1.Освоить создание программ с оператор выбора switch.
2.Освоить работу с интегрированной средой Borland C++.
3.Освоить работу с отладчиком.
4.Ознакомится с организацией контролирующих программ.

2.Используемое программное обеспечение

Для выполнения лабораторной работы используется Borland C версия 3.1 и программа генерации задания lab5.exe.

3.Порядок выполнения лабораторной работы

1.Запустить программу генерации задания lab5 и Borland C++.
2.Зарегистрироваться в контролирующей программе.
3.Получить задание (генерируется контролирующей программой).
4.Составить программу.
5.Проверить правильность работы программы с помощью контролирующей программы и ручного расчета.
6.Проверить работу программы в отладчике.

Теоретическая часть.

Программы и процесс программирования становятся более интересными, когда вводятся операторы, которые позволяют программе выбрать альтернативные действия. В C++ имеются операторы if, if else, а также switch, представляющие собой средства управления выбором пути выполнения, i f позволяет программе выполнить оператор или блок операторов в случае удовлетворения некоторого условия. То есть программа выполняет этот оператор или блок, только если конкретное условие истинно, if else позволяет программе выбрать для выполнения одно из двух операторов или блоков. Вы можете добавлять дополнительные операторы if else для представления серии вариантов выбора. Оператор C++ switch направляет поток управления программы в определенное место из списка возможных.
В C++ также доступны операции, помогающие принимать решения. В главе 5 обсуждаются выражения отношений, которые сравнивают два значения, if и if else обычно в качестве проверочных условий используют выражения сравнения. Используя логические операции C++ (&&, | | и !), вы можете комбинировать или модифицировать

выражения сравнения для конструирования более сложных тестов. Условная операция (? :) предлагает компактный способ выбора одного из двух значений по условию.
Библиотека символьных функций cctype предлагает удобный и мощный набор инструментов анализа символьного ввода.
Циклы и операторы выбора — это полезные инструменты для организации файлового ввода-вывода, который во многом повторяет консольный. После того как вы объявляете объекты ifstream и ofstream и ассоциируете их с файлами, их можно использовать в той же манере, что и стандартные cin и cout.
Используя циклы и условные операторы C++, вы можете писать интересные, интеллектуальные и мощные программы. Но мы только начали исследовать реальную мощь языка C++. Далее мы обратимся к функциям.

Практическая часть.

Напишите программу, которая читает слова по одному за раз, пока не будет введена отдельная буква **q.** После этого программа должна сообщить количество слов, начинающихся с гласных, количество слов, начинающихся с согласных, а также количество слов, не попадающих ни в одну из этих категорий. Одним из возможных подходов может быть применение isalpha () для различия между словами, начинающимися с букв, и остальными, с последующим применением if или switch для идентификации тех слов, прошедших проверку isalpha (), которые начинаются с гласных.
Пример запуска может выглядеть так:
Вводите слова (q — для выхода):
The 12 awesome oxen ambled quietly across 15 meters of lawn, q
слов начинаются с гласных
слов начинаются с согласных
остальных
Напишите программу, которая открывает текстовый файл, читает его символ за символом до самого конца и сообщает количество символов в файле.
Выполните упражнение 6 из лаб. Работы №3, но измените его так, чтобы данные можно было получать из файла. Первым элементом файла должно быть количество жертвователей, а остальная часть состоять из пар строк, в которых первая строка содержит имя, а вторая — сумму пожертвования. То есть файл должен выглядеть примерно так:
Sam Stone **2000**
Freida Flass 100500

Tammy Tubbs 5000
Rich Raptor 5500

Лабораторная работа № 6

Тема: Указатели и их использование.

1. Цель лабораторной работы

1.Освоить работу с указателями.

2.Освоить возвращение из функции нескольких значений числовых значений через указатели.

3.Освоить работу с проектом и подключением внешних функций.

4.Разобраться с моделями памяти в Си.

2.Используемое программное обеспечение

Для выполнения лабораторной работы используется Borland C версия 3.1 и контролирующая программа lab6.exe.

3. Порядок выполнения лабораторной работы

Запустить программу генерации задания lab6 и Borland C. 1.
Зарегистрироваться в контролирующей программе.
2. Получить задание (генерируется контролирующей программой).
3. Составить программу.
4. Проверить правильность работы программы с помощью контролирующей программы и ручного расчета.
5. Проверить работу программы в отладчике.

Теоретическая часть.

При выполнении любой программы, все необходимые для ее работы данные должныбыть загружены в оперативную память компьютера. Для обращения к переменным, находящимся в памяти, используются специальные адреса, которые записываются в шестнадцатеричном виде, например 0x100 или 0x200.

Если переменных в памяти потребуется слишком большое количество, которое не сможет вместить в себя сама аппаратная часть, произойдет перегрузка системы или её зависание.

Если мы объявляем переменные статично, так как мы делали в предыдущих уроках, они остаются в памяти до того момента, как программа завершит свою работу, а после чего уничтожаются.

Такой подход может быть приемлем в простых примерах и несложных программах, которые не требуют большого количества ресурсов. Если же наш проект является огромным программным комплексом с высоким функционалом, объявлять таким образом переменные, естественно, было бы довольно не умно.

Можете себе представить, если бы небезызвестная **Battlefield 3** использовала такой метод работы с данными? В таком случае, самым заядлым геймерам пришлось бы перезагружать свои высоконагруженные системы кнопкой reset после нескольких секунд работы игры.

Дело в том, что играя в тот же Battlefield, геймер в каждый новый момент времени видит различные объекты на экране монитора, например сейчас я стреляю во врага, а через долю секунды он уже падает убитым, создавая вокруг себя множество спецэффектов, таких как пыль, тени, и т.п.

Естественно, все это занимает какое-то место в оперативной памяти компьютера. Если не уничтожать неиспользуемые объекты, очень скоро они заполнят весь объем ресурсов ПК.

По этим причинам, в большинстве языков, в том числе и C/C++, имеется понятие указателя. Указатель — это переменная, хранящая в себе адрес ячейки оперативной памяти, например **0x100.**

Мы можем обращаться, например к массиву данных через указатель, который будет содержать адрес начала диапазона ячеек памяти, хранящих этот массив.

После того, как этот массив станет не нужен для выполнения остальной части программы, мы просто освободим память по адресу этого указателя, и она вновь станет доступно для других переменных.

Ниже приведен конкретный пример обращения к переменным через указатель и напрямую.

Пример использования статических переменных

```
#include <iostream>
using namespace std;

int main()
{
    int a; // Объявление статической переменной
    int b = 5; // Инициализация статической переменной b

    a = 10;
    b = a + b;
    cout << "b is " << b << endl;
    return 0;
}
```

Пример использования динамических переменных

```
#include <iostream>
```

```
using namespace std;

int main()
{
   int *a = new int; // Объявление указателя для переменной типа int
   int *b = new int(5); // Инициализация указателя

   *a = 10;
   *b = *a + *b;

   cout << "b is " << *b << endl;

   delete b;
   delete a;

   return 0;
}
```

Синтаксис первого примера вам уже должен быть знаком. Мы объявляем/инициализируем статичные переменные **a** и **b**, после чего выполняем различные операции напрямую с ними.

Во втором примере мы оперируем динамическими переменными посредством указателей. Рассмотрим общий синтаксис указателей в C++.

Выделение памяти осуществляется с помощью оператора **new** и имеет вид: **тип_данных *имя_указателя = new тип_данных;**, например **int *a = new int;**. После удачного выполнения такой операции, в оперативной памяти компьютера происходит выделение диапазона ячеек, необходимого для хранения переменной типа **int.**

Логично предположить, что для разных типов данных выделяется разное количество памяти. Следует быть особенно осторожным при работе с памятью, потому что именно ошибки программы, вызванные утечкой памяти, являются одними из самых трудно находимых. На отладку программы в поисках одной ничтожной ошибки, может уйти час, день, неделя, в зависимости от упорности разработчика и объема кода.

Инициализация значения, находящегося по адресу указателя выполняется схожим образом, только в конце ставятся круглые скобки с нужным значением: **тип данных *имя_указателя = new тип_данных(значение)**. В нашем примере это **int *b = new int(5)**.

Для того, чтобы получить **адрес** в памяти, на который ссылается указатель, используется имя переменной-указателя с префиксом **&**. перед ним *(не путать со знаком ссылки в C++).*

Например, чтобы вывести на экран адрес ячейки памяти, на который ссылается указатель **b** во втором примере, мы пишем cout << "Address of b is " << &b << endl;. В моей системе, я получил значение **0x1aba030**. У вас оно может быть другим, потому что адреса в оперативной памяти распределяются таким образом, чтобы максимально уменьшить фрагментацию. Поскольку, в любой системе список запущенных процессов, а также объем и разрядность памяти могут отличаться, система сама распределяет данные для обеспечения минимальной фрагментации.

Для того, чтобы получить **значение**, которое находится **по адресу**, на который ссылается указатель, **используется префикс *** . Данная операция называется **разыменованием указателя**.

Во втором примере мы выводим на экран значение, *которое находится в ячейке памяти* (у меня это **0x1aba030**): cout << "b is " << *b << endl; . В этом случае необходимо использовать знак ***.**

Чтобы изменить значение, находящееся по адресу, на который ссылается указатель, нужно также использовать звездочку, например, как во втором примере — *b = *a + *b;.

- Когда мы оперируем **данными**, то используем знак ***
- Когда мы оперируем **адресами**, то используем знак **&**

В этих вещах очень часто возникают недопонимания, и кстати, не только у новичков. Многие из тех, кто начинал программировать с того же php, также часто испытывают подобную путаницу при работе с памятью.

Для того, чтобы освободить память, выделенную оператором **new,** используется оператор delete.

Пример освобождения памяти

```
#include <iostream>
using namespace std;

int main()
{
    // Выделение памяти
    int *a = new int;
    int *b = new int;
    float *c = new float;

    // ... Любые действия программы

    // Освобождение выделенной памяти
    delete c;
```

```
    delete b;
    delete a;

    return 0;
}
```

При использовании оператора delete для указателя, знак *** не используется**.

Лабораторная работа №7

Тема: Ссылки и их использование.

1. Цель лабораторной работы

1.Освоить работу с ссылками и их использование.
2.Освоить возвращение из функции нескольких значений числовых значений через указатели.
3.Освоить работу с проектом и подключением внешних функций.
4.Разобраться с моделями памяти в Си.

2.Используемое программное обеспечение

Для выполнения лабораторной работы используется Borland C версия 3.1 и контролирующая программа lab7.exe.

3. Порядок выполнения лабораторной работы

Запустить программу генерации задания lab7 и Borland C.
1. Зарегистрироваться в контролирующей программе.
2. Получить задание (генерируется контролирующей программой).
3. Составить программу.
4. Проверить правильность работы программы с помощью контролирующей программы и ручного расчета.
5. Проверить работу программы в отладчике.

Теоретический часть.

Ссылки — это третий базовый тип переменных. **Ссылка** — это тип переменной в C++, который работает как псевдоним другого объекта или значения.

C++ поддерживает три типа ссылок:

ссылки на неконстантные значения (обычно их называют просто «ссылки» или «неконстантные ссылки»), которые мы обсудим в этом уроке.

ссылки на константные значения (обычно их называют «константные ссылки»), которые мы обсудим в следующем уроке.

в C++11 добавлены ссылки на r-value, о которых мы поговорим уже в следующих главах.

Ссылки на неконстантные значения

Ссылка (на неконстантное значение) объявляется с использованием амперсанда (**&**) между типом и именем ссылки:

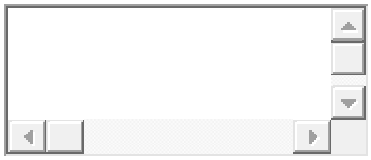
```
1 int value = 7; // обычная переменная
2 int &ref = value; // ссылка на переменную value
```

В этом контексте амперсанд не означает «оператор адреса», он означает «ссылка на».

Ссылки как псевдонимы

Ссылки обычно ведут себя идентично значениям, на которые они ссылаются. В этом смысле ссылка работает как **псевдоним** объекта, на который она ссылается.

Например:



```
1   #include <iostream>
2
3   int main()
4   {
5       int value = 7; // обычная переменная
6       int &ref = value; // ссылка на переменную value
7
8       value = 8; // value теперь 8
9       ref = 9; // value теперь 9
10
11      std::cout << value; // выведется 9
12      ++ref;
13      std::cout << value; // выведется 10
14
15      return 0;
16  }
```

Результат:

```
9
10
```

В примере выше ref и value обрабатываются как одно целое.

Использование оператора адреса с ссылкой приведет к возврату адреса значения, на которое ссылается ссылка:

```
1 cout << &value; // выведется 0035FE58
2 cout << &ref; // выведется 0035FE58
```

Краткий обзор l-value и r-value

В **10 уроке** мы уже рассматривали, что такое l-value и r-value, и говорили вам тогда не слишком беспокоиться о них. Теперь же, наконец, мы дошли до того момента, где эти термины уже будет полезно не только вспомнить, но и использовать.

l-value — это объект, который имеет определенный адрес памяти (например, переменная x) и сохраняется за пределами одного выражения. r-value — это временное значение без определенного адреса памяти и с областью видимости выражения (т.е. сохраняется в пределах одного выражения). В качестве r-values могут быть как результаты выражения (например, 2 + 3), так и **литералы.**

Ссылки должны быть инициализированы

Ссылки должны быть инициализированы при создании:

```
1 int value = 7;
2 int &ref = value; // корректная ссылка. Инициализирована переменной value
3
4 int &invalidRef; // некорректная ссылка. Ссылка должна ссылаться на что-нибудь
```

В отличие от указателей, которые могут содержать нулевое значение, ссылки нулевыми быть не могут.

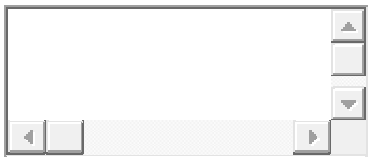
Ссылки на неконстантные значения могут быть инициализированы только неконстантными l-values. Они не могут быть инициализированы константными l-values или r-values.

```
1 int a = 7;
2 int &ref1 = a; // хорошо, a - это неконстантное l-value
3
4 const int b = 8;
5 int &ref2 = b; // не хорошо, b - это константное l-value
6
7 int &ref3 = 4; // не хорошо, 4 - это r-value
```

Обратите внимание, во втором случае вы не можете инициализировать неконстантную ссылку константным объектом – иначе вы могли бы изменить значение константного объекта через ссылку, что уже является нарушением понятия «константа».

Ссылки не могут быть переприсвоены

После инициализации изменить объект, на который указывает ссылка, нельзя. Рассмотрим следующий фрагмент:



```
1 int value1 = 7;
2 int value2 = 8;
3
4 int &ref = value1; // хорошо, ref - теперь псевдоним для value1
5 ref = value2; // присваиваем 8 (значение переменной value2) переменной value1 - здесь НЕ изменяется объект, на который ссылается ссылка!
```

Обратите внимание, во втором стейтменте (5 строчка) выполняется не то, что вы могли бы ожидать! Вместо переприсваивания ref (ссылаться на переменную value2), значение из value2 присваивается переменной value1 (на которое и ссылается ref).

Ссылки в качестве параметров в функциях

Ссылки чаще всего используются в качестве **параметров** в функциях. В этом контексте ссылка-параметр работает как псевдоним аргумента, и сам аргумент не копируется при передаче в параметр. Это в свою очередь

улучшает производительность, если аргумент слишком большой или затратный для копирования.

В **82 уроке** мы говорили о том, что передача аргумента-указателя функции, позволяет функции при разыменовании этого указателя *напрямую* изменять значение аргумента.

Ссылки работают аналогично. Поскольку ссылка-параметр — это псевдоним аргумента, то функция, использующая ссылку-параметр, может изменять аргумент, переданный ей, также *напрямую*:

```
#include <iostream>

// ref - это ссылка на переданный аргумент, не копия аргумента
void changeN(int &ref)
{
	ref = 8;
}

int main()
{
	int x = 7;

	std::cout << x << '\n';

	changeN(x); // обратите внимание, этот аргумент не обязательно должен быть ссылкой

	std::cout << x << '\n';
	return 0;
}
```

Результат:

```
7
8
```

Когда аргумент x передан в функцию, то параметр функции ref становится ссылкой на аргумент x. Это позволяет функции изменять значение x непосредственно через ref! Обратите внимание, переменная x не обязательно должна быть ссылкой.

Совет: Передавайте аргументы в функцию через неконстантные ссылки-параметры, если они должны быть изменены функцией в дальнейшем.

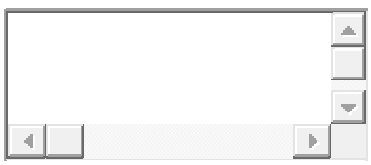
Основным недостатком использования неконстантных ссылок в качестве параметров в функции является то, что аргумент должен быть неконстантным l-value, т.е. константой или литералом он быть не может. Мы поговорим об этом подробнее (и то как это обойти) в следующем уроке.

Ссылки как более легкий способ доступа к данным

Второе (гораздо менее используемое) применение ссылок заключается в более легком способе доступа к вложенным данным. Рассмотрим следующую **структуру**:

```
1  struct Something
2  {
3      int value1;
4      float value2;
5  };
6
7  struct Other
8  {
9      Something something;
10     int otherValue;
11 };
12
13 Other other;
```

Предположим, нам нужно работать с полем value1 структуры Something переменной other структуры Other (звучит сложно, но такое также встречается на практике). Обычно доступ к этому полю осуществлялся бы через `other.something.value1`. А если есть много отдельных случаев доступа к этому члену? Код может стать несколько громоздким и беспорядочным. Ссылки же предоставляют более легкий способ доступа:



```
1 int &ref = other.something.value1;
```

2 // ref теперь может использоваться вместо other.something.value1

Таким образом, следующие два стейтмента идентичны:

```
1 other.something.value1 = 7;
2 ref = 7;
```

Ссылки позволяют сделать ваш код более чистым и понятным.

Ссылки против указателей

Ссылки с указателями имеют интересное соотношение. Ссылка – это тот же указатель, который неявно разыменовывается при доступе к значению, на которое он указывает (под капотом ссылки реализованы компилятором с помощью указателей). Таким образом, в следующем:

```
1 int value = 7;
2 int *const ptr = &value;
3 int &ref = value;
```

*ptr и ref обрабатываться будут одинаково. Т.е. это одно и то же:

```
1 *ptr = 7;
2 ref = 7;
```

Поскольку ссылки должны быть инициализированы валидными объектами (они не могут быть нулевыми) и не могут быть изменены позже, то они, как правило, безопаснее указателей (так как риск разыменования нулевого указателя отпадает). Однако они немного ограничены в функциональности, по сравнению с указателями.

Если определенное задание может быть решено с помощью как ссылок, так и указателей, то лучше использовать ссылки. Указатели следует использовать в ситуациях, только когда ссылки недостаточно эффективны (например, при **динамическом выделении памяти**).

Итого

Ссылки позволяют определять псевдонимы для других объектов или значений. Ссылки на неконстантные значения могут быть инициализированы только неконстантными l-values. Они не могут быть переприсвоены после инициализации.

Ссылки чаще всего используются в качестве параметров в функциях, когда мы либо хотим изменить значение аргумента, либо хотим избежать его затратного копирования.

Лабораторная работа №8.

Тема: Директивы и их типы

1. Цель лабораторной работы

1.Освоить работу с директивами и их типами.
2.Освоить возвращение из функции нескольких значений числовых значений через указатели.
3.Освоить работу с проектом и подключением внешних функций.
4.Разобраться с моделями памяти в Си.

2.Используемое программное обеспечение

Для выполнения лабораторной работы используется Borland C версия 3.1 и контролирующая программа lab8.exe.

3. Порядок выполнения лабораторной работы

Запустить программу генерации задания lab8 и Borland C.
1. Зарегистрироваться в контролирующей программе.
2. Получить задание (генерируется контролирующей программой).
3. Составить программу.
4. Проверить правильность работы программы с помощью контролирующей программы и ручного расчета.
5. Проверить работу программы в отладчике.

Теоретический часть:

Препроцессор — это специальная программа, являющаяся частью компилятора языка Си. Она предназначена для предварительной обработки текста программы. Препроцессор позволяет включать в текст программы файлы и вводить макроопределения.
Работа препроцессора осуществляется с помощью специальных директив (указаний). Они отмечаются знаком решетка #. По окончании строк, обозначающих директивы в языке Си, точку с запятой можно не ставить.

Основные директивы препроцессора

#include — вставляет текст из указанного файла
#define — задаёт макроопределение (макрос) или символическую константу
#undef — отменяет предыдущее определение
#if — осуществляет условную компиляцию при истинности константного выражения
#ifdef — осуществляет условную компиляцию при определённости символической константы
#ifndef — осуществляет условную компиляцию при неопределённости символической константы

#else — ветка условной компиляции при ложности выражения
#elif — ветка условной компиляции, образуемая слиянием else и if
#endif — конец ветки условной компиляции
#line — препроцессор изменяет номер текущей строки и имя компилируемого файла
#error — выдача диагностического сообщения
#pragma — действие, зависящее от конкретной реализации компилятора.

Директива #include
Директива #include позволяет включать в текст программы указанный файл. Если файл является стандартной библиотекой и находится в папке компилятора, он заключается в угловые скобки <>.
Если файл находится в текущем каталоге проекта, он указывается в кавычках "". Для файла, находящегося в другом каталоге необходимо в кавычках указать полный путь.

```
#include <stdio.h>
#include "func.c"
```

Директива #define

Директива #define позволяет вводить в текст программы константы и макроопределения.
Общая форма записи

#define Идентификатор Замена

Поля **Идентификатор** и **Замена** разделяются одним или несколькими пробелами.
Директива #define указывает компилятору, что нужно подставить строку, определенную аргументом **Замена**, вместо каждого аргумента **Идентификатор** в исходном файле. Идентификатор не заменяется, если он находится в комментарии, в строке или как часть более длинного идентификатора.

```
#include <stdio.h>
#define A 3
int main()
{
  printf("%d + %d = %d", A, A, A+A); // 3 + 3 = 6
  getchar();
```

```
  return 0;
}
```

В зависимости от значения константы компилятор присваивает ей тот или иной тип. С помощью суффиксов можно переопределить тип константы:

- U или u представляет целую константу в беззнаковой форме (unsigned);
- F (или f) позволяет описать вещественную константу типа float;
- L (или l) позволяет выделить целой константе 8 байт (long int);
- L (или l) позволяет описать вещественную константу типа long double

```
#define A 280U   // unsigned int
#define B 280LU  // unsigned long int
#define C 280    // int (long int)
#define D 280L   // long int
#define K 28.0   // double
#define L 28.0F  // float
#define M 28.0L  // long double
```

Вторая форма синтаксиса определяет макрос, подобный функции, с параметрами. Эта форма допускает использование необязательного списка параметров, которые должны находиться в скобках. После определения макроса каждое последующее вхождение
идентификатор(аргумент1, ..., агрументn)

замещается версией аргумента **замена**, в которой вместо формальных аргументов подставлены фактические аргументы.

Пример на Си: Вычисление синуса угла

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.14159265
#define SIN(x) sin(PI*x/180)
int main()
{
  int c;
  system("chcp 1251");
  system("cls");
  printf("Введите угол в градусах: ");
  scanf("%d", &c);
  printf("sin(%d)=%lf", c, SIN(c));
  getchar(); getchar();
  return 0;
}
```

Результат выполнения

Отличием таких макроопределений от функций в языке Си является то, что на этапе компиляции каждое вхождение идентификатора замещается соответствующим кодом. Таким образом, программа может иметь несколько копий одного и того же кода, соответствующего идентификатору. В случае работы с функциями программа будет содержать 1 экземпляр кода, реализующий указанную функцию, и каждый раз при обращении к функции ей будет передано управление.
Отменить макроопределение можно с помощью директивы #undef.

Однако при использовании таких макроопределений следует соблюдать осторожность, например

```
#include <stdio.h>
#define sum(A,B) A+B
int main()
{
  int a, b, c, d;
  a = 3; b = 5;
  c = (a + b) * 2; // c = (a + b)*2
  d = sum(a, b) * 2; // d = a + b*2;
  printf(" a = %d\n b = %d\n", a, b);
  printf(" c = %d \n d = %d \n", c, d);
  getchar();
  return 0;
}
```

Результат выполнения:

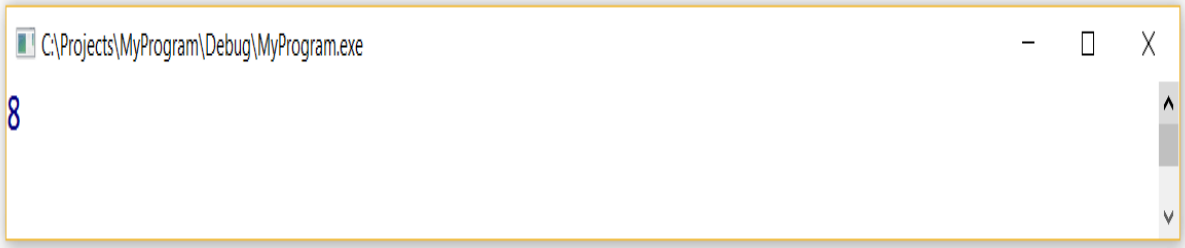
По умолчанию текст макроопределения должен размещаться на одной строке. Если требуется перенести текст макроопределения на новую строку, то в конце текущей строки ставится символ "обратный слеш" — \.

```
#include <stdio.h>
#define sum(A,B) A + \
                 B
int main()
{
  int a, b, c, d;
  a = 3; b = 5;
  c = (a + b) * 2; // c = (a + b)*2
  d = sum(a, b) * 2; // d = a + b*2;
  printf(" a = %d\n b = %d\n", a, b);
  printf(" c = %d \n d = %d \n", c, d);
  getchar();
  return 0;
}
```

Кроме того, директива #define позволяет замещать часть идентификатора. Для указания замещаемой части используется ##.

```
#include <stdio.h>
#define SUM(x,y) (a##x + a##y)
int main()
{
  int a1 = 5, a2 = 3;
  printf("%d", SUM(1, 2)); // (a1 + a2)
  getchar();
  return 0;
}
```

Результат выполнения:



Условная компиляция

Директивы #if или #ifdef/#ifndef вместе с директивами #elif, #else и #endif управляют компиляцией частей исходного файла.

Если указанное выражение после #if имеет ненулевое значение, в записи преобразования сохраняется группа строк, следующая сразу за

директивой #if. Синтаксис условной директивы следующий:

```
#if константное выражение
  группа операций
#elif константное выражение
  группа операций
#else
  группа операций
#endif
```

Отличие директив #ifdef/#ifndef заключается в том, что константное выражение может быть задано только с помощью #define.

У каждой директивы #if в исходном файле должна быть соответствующая закрывающая директива #endif. Между директивами #if и #endif может располагаться любое количество директив #elif, однако допускается не более одной директивы #else. Директива #else, если присутствует, должна быть последней перед директивой #endif.

Пример

```
#include <stdio.h>
#include <stdlib.h>
#define P 2
int main()
{
 system("chcp 1251");
 system("cls");
#if P==1
 printf("Выполняется ветка 1");
#elif P==2
 printf("Выполняется ветка 2, P=%d", P);
#else
 printf("Выполняется другая ветка, P=%d", P);
#endif
 getchar();
 return 0;
}
```

Результат выполнения

Литература

1. Аксёнкин М.А., Целобёнок О.Н. Язык С. Минск: "Уіверсітэцкае", 1995.
2. Бочков С.О., Субботин Д.М. Язык программирования Си для персонального компьютера. М СП: "Диалог" "Радио и связь", 1990 г.
3. Прокофьев Б.П. и др. Графические средства Turbo C и Turbo C++. М: "Финансы и статистика", 1992 г.
4. Справочник по функциям Borland C++ 3.1/4.0. Киев: Диалектика, 1994.
5. Страуструп Б. Язык программирования Си++. М: Радио и связь, 1991.
6. Красов А.В. Программирование на языке Си. Ч.1. СПб, 2001(или более ранние издания).
7. Красов А.В. Программирование на языке Си. Ч2. Использование графики и программная обработка данных. СПбГУТ 2001(или более ранние издания).
8. Красов А.В. Программирование на Borland C под Ms Windows. СПб, СПбГУТ 2001 (или более ранние издания).