Лабораторная работа № 1 Динамические массивы объектов

Цель:

Получить практические навыки работы с динамическими массивами объектов

Ключевые положения

Указатель на объект

Указатель на объект– это *переменная*, которая хранит адрес объекта в памяти компьютера. Для объявления указателя на объект используется тот же синтаксис, как и в случае объявления *указателя на переменную* любого типа – следует добавить *звёздочку* после идентификатора типа объекта, непосредственно или через пробел:

Samp*p; //объявление указателя на тип **Samp**

При объявлении более чем одного указателя звёздочка ставится перед каждым из них. Как и другие переменные, помимо объявления, указатели требуют *инициализации*, так как в противном случае они содержат непредсказуемые значения. При инициализации указателям присваивается значение нуль (**NULL**)или *конкретный адрес*. Адрес указателю можно присвоить с помощью *операции адреса* (&). Разыменование указателя выполняется с помощью *операции разыменования* (*).
Доступ к отдельному ***public***-элементу объекта можно получить напрямую, используя *операцию точка* **(.),**или через указатель на этот объект – в этом случае необходимо использовать *операцию стрелка* **(–>).**

// **Пример 1**. Объявление и использование указателя на объект класса.

```
#include <iostream>
using namespace std;
class Samp
{
     int x;
public:
     int y;
     void setx(int n){x = n;}
     void show();
};

void Samp::show()
{
     cout<<" x = "<<x<<" y = "<<y<<endl;
}


int main()
{
     Samp ob, *p; // создание объекта ob и указателя p
```

```
     ob.setx(1); // прямой доступ к членам объекта ob
     ob.y = 1; ob.show(); // x = 1 y= 1
     p = &ob; // присваивание p адреса объекта ob
     p->setx(5); // вызов функции через указатель p
     p->y = 5; p->show(); // x = 5 y= 5
     (*p).y = 55; p->show(); // x = 5 y= 55
}
```

В данном примере выражение **(*p)** разыменовывает указатель, разрешая доступ к содержимому ячейки памяти по адресу, хранящемуся в**p**.

Указатель на массив объектов

Для объявления ***указателя на массив объектов*** используется тот же синтаксис, что и для объявления указателя на массив переменных любого типа, например:

Sampob[5]; // массив объектов **ob**

Samp*p=ob; //**p** присваивается адрес массива объектов **ob**

p = &ob[0]; // или так

// **Пример 2.** Использование указателя на массив объектов.

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

class Samp
{
     int x;
public:
     Samp(int n){x = n;} // конструктор
     int getx(){return x;}
};

int main()
{
     Samp ob[4] = { 1, 2, 3, 4 };
     Samp *p = ob; // p присваивается адрес массива ob
     for(int i = 0; i < 4; i++) // чтение массива объектов ob
     cout<< p[i].getx()<<' '; // 1 2 3 4
     cout<<endl;
}
```

Динамические объекты

Для создания ***динамического объекта***(как и для других данных встроенных типов) используют *операцию* ***new*** и *указатель на объект*.
Операция ***new*** выделяет память для объекта в специальной области памяти, называемой **кучей**, и вызывает конструктор. В качестве операнда выступает название типа, а

результатом является адрес выделенной памяти. Например, так будет создан новый динамический объект типа **Samp**, на который указывает указатель ***p***:

```
Samp * p;
 p = new Samp();
или
Samp * p = new Samp();
```

Созданный таким образом объект существует до тех пор, пока память не будет явно освобождена с помощью *операции* ***delete***, которую можно поместить в деструкторе. В качестве операнда должен быть задан адрес, возвращённый *операцией* ***new*** (указатель ***p***):

```
delete p;
```

Если свободной памяти для создания нового объекта недостаточно, *операция* ***new*** возвращает нулевой указатель (p=**NULL**). Поэтому программа должна всегда проверять *успешность создания динамического объекта*:

```
if(!p)
{
        cout<<"Error: no memory"<<endl;
        exit(1);
}
```

C++ позволяет при создании динамических объектов инициализировать их аналогично переменных любых других типов:

```
Samp * p = new Samp(5, 6); //x = 5 y = 6
```

Объекты, созданные с помощью *операции* ***new***, называются ***динамическими объектами***, потому что они создаются и уничтожаются в процессе работы программы. Динамические объекты размещаются в специальной области памяти –***куче***). Для доступа к элементам объекта применяется *операция стрелка*(**->**).

Динамические массивы объектов

Динамический массив объектов– это массив, для которого выделяется память и определяется его размер во время выполнения программы.

Динамический массив объектов создаётся с помощью *операции* ***new*** и *указателя*. В качестве операнда операции выступает название типа (имя класса) и размер массива, заданный в квадратных скобках после типа, а результатом выполнения операции является адрес начала выделенной памяти под массив объектов:

```
Samp *p = new Samp[размер массива];
```

Размер массива в *операции* ***new*** может быть переменной, значение которой может быть вычислено в ходе выполнения программы. Адрес начала массива хранится в указателе ***p***. Если памяти недостаточно, то *операция* ***new*** возвращает значение **NULL**. Поэтому

программа должна всегда проверять успешность создания динамического массива объектов (аналогично, как это делали для динамических объектов). После завершения использования массива объектов имеет смысл освободить память, выделенную под массив объектов, с помощью *операции* ***delete***:

delete[] p;

Доступ к элементам динамического массива осуществляется так же, как и к элементам обычного массива.

Из-за ряда чисто технических причин ***невозможно явно инициализировать динамические массивы***. Поэтому, если в классе есть конструктор с инициализацией, то необходимо перегрузить конструктор под версию без инициализации (т.е. создать конструктор по умолчанию). В противном случае возникает ошибка компиляции.

// **Пример 4.**Создание динамического объекта, массива объектов.

// Перегрузка конструктора для поддержки динамического массива объектов.

```
#include <iostream>
using namespace std;
class Samp
{ // класс Samp
     int x;
public:
     Samp(){x = 0;} // конструктор по умолчанию
     Samp(int n){x = n;} // конструктор с параметром
     void set(int m){x = m;} // функция установки x
     int get(){return x;} // функция чтения x
};

int main()
{
     Samp *q = new Samp(5); // создание динам. объекта
     cout<<q->get()<<endl; // 5
     int size; // size – размер динам. массива
     cout<<"Vvedi size: "; cin>>size;
     Samp *p = new Samp[size]; //создание динам. массива
объектов
     if(!p)
     {
          cout<<"No memory"<<endl;
          return(1);
     }

     for(int i = 0; i < size; i++)
     { // инициализация массива
          p->set(i); // вариант 1
          p++;
```

```
    }

    // for(int i = 0; i < size; i++) // вариант 2
    //   p[i].set(i*2);
    // for(int i = 0; i < size; i++) // вариант 3
    //   (p+i)->set(i*3);
    // for(int i = 0; i < size; i++) // вариант 4
    //   (*(p+i)).set(i*4);

    for(int i = 0; i < size; i++) // этот цикл нужен
        p--; // только для варианта 1!!!!
    cout << "Array=" << endl; // вывод динам.массива
    for(int i=0; i<size; i++)
        cout<<p[i].get()<<' ';
    cout << endl;
    delete[] p;
    return 0;
}
```

Объявление динамического массива

Массивы, создаваемые в динамической памяти, будем называть *динамическими* (размерность становится известна в процессе выполнения программы). При описании массива после имени в квадратных скобках задается количество его элементов (размерность), например int a[10]. Размерность массива может быть задана только константой или константным выражением.

При описании массив можно инициализировать, то есть присвоить его элементам начальные значения, например:

```
int a[10] = {1, 1, 2, 2, 5, 100};
```

Если инициализирующих значений меньше, чем элементов в массиве, остаток массива обнуляется, если больше — лишние значения не используются. Элементы массивов нумеруются с нуля, поэтому максимальный номер элемента всегда на единицу меньше размерности. Номер элемента указывается после его имени в квадратных скобках, например, a[0], a[3].

Если до начала работы программы неизвестно, сколько в массиве элементов, в программе следует использовать динамические массивы. Память под них выделяется с помощью операции new или функции malloc в динамической области памяти во время выполнения программы. Адрес начала массива хранится в переменной, называемой указателем. Например:

```
int n = 10;
int *arr1 = new int[n];
```

Во второй строке описан указатель на целую величину, которому присваивается адрес начала непрерывной области динамической памяти, выделенной с помощью операции new. Выделяется столько памяти, сколько необходимо для хранения n величин типа int. Величина n может быть переменной. Инициализировать динамический массив нельзя.

Обращение к элементу динамического массива осуществляется так же, как и к элементу обычного. Если динамический массив в какой-то момент работы программы перестает быть нужным и мы собираемся впоследствии использовать эту память повторно, необходимо освободить ее с помощью операции delete[], например: delete [] a; (размерность массива при этом не указывается).

```
delete[] arr1;
```

При необходимости создания многомерных динамических массивов сначала необходимо с помощью операции new выделить память под n указателей (вектор, элемент которого - указатель), при этом все указатели располагаются в памяти последовательно друг за другом. После этого необходимо в цикле каждому указателю присвоить адрес выделенной области памяти размером, равным второй границе массива

```
arr2=new int*[row]; // arr2 - указатель на массив указателей на
одномерные массивы
for(i=0;i<row;i++)
     arr2[i]=new int[col]; // каждый элемент массива указывает
на одномерный
for (i=0; i<row;i++)
     for (j=0;j<col;j++)
          arr2[i][j] = i + j;// присвоение значений элементам
```

Освобождение памяти от двумерного динамического массива:

```
for(i=0;i<row;i++)        //удаление всех одномерных
     delete[] arr2[i];    // массивов
delete[] arr2;       // удаление массива указателей на одномерные
массивы
```

Домашнее задание

Базовый уровень.

Составить программы - одномерные массивы: задания 1-25. Массивы создаются в динамической области памяти с использованием операций NEW и DELETE. Ввод исходных данных: реальный размер массивов и их значения. Обращение к элементам массива – через косвенную адресацию.

1. Заданы два массива A(5) и B(4). Первым на печать вывести массив, сумма значений которого окажется наименьшей.
2. Заданы два массива A(5) и B(4). Первым на печать вывести массив, произведение значений которого окажется наименьшим.

3. Заданы два массива A(5) и B(5). В каждом из массивов найти наименьшее значение и прибавить его ко всем элементам массивов. На печать вывести исходные и преобразованные массивы.
4. Заданы два массива A(5) и B(5). В каждом из массивов найти наибольшее значение и вычесть его из всех элементов массивов. На печать вывести исходные и преобразованные массивы.
5. Заданы два массива A(5) и B(5). В каждом из массивов найти среднее арифметическое всех элементов массивов. На печать вывести исходные массивы и найденные значения.
6. Заданы два массива A(5) и B(4). Первым на печать вывести массив, содержащий наибольшее значение. Напечатать также это значение и его порядковый номер.
7. Заданы два массива A(5) и B(5). Подсчитать в них количество отрицательных элементов и первым на печать вывести массив, имеющий наименьшее их количество.
8. Заданы два массива A(5) и B(5). Подсчитать в них количество положительных элементов и первым на печать вывести массив, имеющий наименьшее их количество.
9. Заданы два массива A(5) и B(5). Подсчитать в них количество отрицательных элементов и первым на печать вывести массив, имеющий наибольшее их количество.
10. Заданы два массива A(5) и B(5). Подсчитать в них количество положительных элементов и первым на печать вывести массив, имеющий наибольшее их количество.
11. Заданы два массива A(5) и B(5). Подсчитать в них количество элементов, больших значения t и первым на печать вывести массив, имеющий наименьшее их количество.
12. Заданы два массива A(5) и B(5). Подсчитать в них количество элементов, меньших значения t и первым на печать вывести массив, имеющий наименьшее их количество.
13. Заданы два массива A(5) и B(5). Подсчитать в них количество элементов, больших значения t и первым на печать вывести массив, имеющий наибольшее их количество.
14. Заданы два массива A(5) и B(5). В каждом из массивов найти наименьшее значение и умножить на него все элементы массивов. На печать вывести исходные и преобразованные массивы.
15. Заданы два массива A(5) и B(5). В каждом из массивов найти наибольшее значение и умножить на него все элементы массивов. На печать вывести исходные и преобразованные массивы.
16. Заданы два массива A(5) и B(5). В каждом из массивов найти наименьшее значение и разделить на него все элементы массивов. На печать вывести исходные и преобразованные массивы.
17. Заданы два массива A(5) и B(5). В каждом из массивов найти наибольшее значение и разделить на него все элементы массивов. На печать вывести исходные и преобразованные массивы.
18. Заданы два массива A(5) и B(5). Подсчитать в них количество элементов, кратных двум и первым на печать вывести массив, имеющий наибольшее их количество.
19. Заданы два массива A(5) и B(5). Подсчитать в них количество элементов, кратных трем и первым на печать вывести массив, имеющий наибольшее их количество.
20. Заданы два массива A(5) и B(5). Подсчитать в них количество элементов, меньших значения t и первым на печать вывести массив, имеющий наибольшее их количество.

21. Задан массив A(10). Получить из него массив B, состоящий из элементов массива A, которые больше 0.
22. Задан массив A(10). Получить из него массив B, состоящий из элементов массива A, которые меньше 0.
23. Задан массив A(10). Получить из него массив B, состоящий из элементов массива A, которые кратны двум.
24. Задан массив A(10). Получить из него массив B, состоящий из элементов массива A, которые больше значения T.
25. Задан массив A(10). Получить из него массив B, состоящий из элементов массива A, которые кратны трем.

Средний уровень

Составить программы - двухмерные массивы: задания 26-50. Массивы создаются в динамической области памяти с использованием операций NEW и DELETE. Ввод исходных данных: реальный размер массивов и их значения. Обращение к элементам массива – через косвенную адресацию.

26.Дан массив A(n,n).Найти число элементов массива a(i,j)>t и просуммировать все эти элементы.
27.Дан одномерный массив A(n).Сформировать массив B(k),состоящий из a(i) > t. На печать вывести исходный массив, сформированный массив и его размерность.
28.Дан массив A(n,n). Вычислить сумму всех неотрицательных элементов, а также их количество.
29. Дан массив A(n,n). Вычислить сумму всех отрицательных его элементов и их количество.
30. Дан массив A(n,n). Сформировать вектор B(к) из a(i,j) < 0. На печать вывести исходный массив, полученный вектор и его размерность.
31. Дан массив A(n,n). Написать программу его поворота на 900 относительно его центра. На печать вывести исходный и повернутый массивы.
32. Дан массив A(n,n). Написать программу его поворота на 1800 относительно его центра. На печать вывести исходный и повернутый массивы.
33. Дан массив A(n,n). Написать программу его поворота на 2700 относительно его центра. На печать вывести исходный и повернутый массивы.
34. Дан массив A(n,n). Найти сумму всех его элементов, расположенных выше главной диагонали.
35. Дан массив A(n,n). Найти сумму всех его элементов, расположенных ниже главной диагонали.
36. Дан массив A(n,n). Найти сумму всех его элементов, расположенных выше диагонали, противоположной главной.
37. Дан массив A(n,n). Найти сумму всех его элементов, расположенных ниже диагонали, противоположной главной.

38. Задана матрица A(n,n). Найти суммы и произведения элементов, стоящих на главной и противоположной (побочной) диагоналях.
39. Задана матрица A(n,n), состоящая из нулей и единиц. Подсчитать количество нулей и единиц в этой матрице.
40. Задана матрица A(n,n). Переставить местами к-ю и i-ю строки, а затем l-й и j-й столбцы.
41. Задан массив действительных чисел A(n). Необходимо каждый элемент массива разделить на среднее арифметическое этих элементов. На печать вывести исх. и преобразов. массивы.
42. Задан массив A(n). Получить массив B(к), состоящий из элементов массива A, которые делятся на 3. Подсчитать количество элементов массива B.
43. Задана матрица A(n,n). Получить матрицу B=A^2.Элемент b[I][j] определяется как сумма от поэлементного произведения I-й строки на j-й столбец матрицы A.
44. Вычислить первую норму матрицы A(n,n) , определяемую как $\|A\| = \max_i \sum \|a[i][j]\|$, т.е. максимальная сумма из сумм элементов по столбцам
45. Вычислить вторую норму матрицы A(n,n), определяемую как максимальная сумма из сумм элементов по строкам $\|A\| = \max_j \sum \|a[i][j]\|$.
46. Задан двухмерный массив целых чисел A размером N на M. Найти сумму элементов, расположенных на главной диагонали.
47. Задан двухмерный массив целых чисел A размером N на M. Найти произведение элементов, расположенных на главной диагонали.
48. Задан двухмерный массив целых чисел A размером N на M. Найти максимальный элемент и поменять его с элементом A[1,1].
49. Задан двухмерный массив целых чисел A размером N на M. Найти минимальный элемент и поменять его с элементом A[1,1].
50. Задан двухмерный массив целых чисел A размером N на M. Найти максимальный элемент и поменять его с последним.

Высокий уровень

Создать программу работы с динамическим массивом указателей на объекты класса Person, содержащего скрытые поля: имя (name), возраст (age), специальность (spatiality) и публичные методы set() get() для каждого поля данных, входящих в объект класса Manager. Массив заполняется в порядке поступления людей, а каждый объект Person создаётся в куче независимо. Если массив заполнен, то увеличить массив вдвое. Объект класса Manager должен выводить список людей отсортированных по каждому из трёх полей данных.

Для приведенных выше задач составить алгоритм и программу консольного приложения на C++.

Лабораторное задание

1. Запустить интегрированную среду разработки
2. Ввести и отладить разработанное дома программное обеспечение
3. Продемонстрировать преподавателю работоспособную программу
4. Оформить протокол.

Ключевые вопросы

1. Что такое указатель на объект и как его объявить и инициализировать?
2. Что такое динамический массив объектов и чем он отличается от статического?
3. Как создать, инициализировать и удалить динамический объект?
4. Каковы особенности создания, инициализации и удаления динамического массива объектов?

Содержание протокола

1. Тема
2. Цель работы.
3. Исходное задание
4. Алгоритм работы программы.
5. Исходный код программы.
6. Результаты выполнения задания
7. Краткие выводы.

Лабораторная работа №2

Тема: Передача объектов в функции

Цель: Освоить приёмы программирования с использованием передачи объектов в функции.

Ключевые положения

Синтаксически объекты передаются в функции так же, как и простые переменные. Для этого объявляют параметр функции, который имеет тип класса. Затем используют объект этого класса в качестве аргумента при вызове функции.

Передача объекта в функцию по значению. По умолчанию происходит передача объектов в функцию по значению. При этом компилятор создаёт в функции *копию* переданного ей объекта, и функция работает с копией. Поэтому изменение копии объекта внутри функции не влияет на сам объект-оригинал.

// **Пример.1. П**ередача объекта в функцию по значению.

```
#include <iostream>
using namespace std;
class Samp
{
     int x;
public:
     Samp(int n){x = n;} // конструктор
     void set(int n){x = n;} // функция установки x
     int get(){return x;} // функция чтения x
};

void fun(Samp ob)
{ //внешняяфункция,
  // параметр – объект ob
     ob.set(100); // изменяем копию объекта
     // или ob.set(ob.get()+90);
     cout<<ob.get()<<endl; // 100
}

int main()
{
     Samp obj(10); // obj.x= 10
     fun(obj); // передачаобпо значению fun()
     cout<<obj.get()<<endl; // 10
     return 0;
}
```

При вызове функции, в момент создания копии объекта, конструктор не вызывается, так как он используется для инициализации элементов объекта, а копия создаётся для уже существующего (а значит – проинициализированного) объекта.

При завершении работы функции созданная копия объекта удаляется и вызывается деструктор копии.

При передаче объекта в функцию по значению создаётся копия объекта. Конструктор копии не вызывается. Деструктор копии вызывается.

Передача объекта в функцию по указателю. При передаче объекта в функцию по указателю происходит передача в функцию адреса объекта. При этом функция может изменить значение элементов объекта, используемого в вызове. Функция fun() из примера 1 и её вызов будут иметь следующий вид:

```
…
     void fun(Samp *p){ // параметр – указатель
     p->set(p->get() + 50); // или просто p->set(60)
     cout<<p->get ()<<endl; // x = 60
}

int main()
{
     Samp ob(10); // создание объектаob,x = 10
     fun(&ob); // передача ob по указателю
     cout<<ob.get()<<endl; // x = 60
     return 0;
}
```

Передача объекта в функции по ссылке. Ссылка является скрытым указателем и работает как другое имя объекта. При передаче объекта по ссылке изменения объекта внутри функции влияют на исходный объект и сохраняются при завершении работы функции.

Ссылка не является аналогом указателя, поэтому при передаче объекта по ссылке для доступа к его элементам используется *операция точка* **(.)**, а не *операция стрелка***(->)**.

```
…
     void fun(Samp &ref){ // параметр – ссылка
     ref.set(ref.get() * ref.get()); // изменяем объект
     // или так – ref.setx(100);
     cout << ref.get() << endl; // x = 100
}

int main()
{
     Samp ob(10); // создание объекта, ob.x = 10
     fun(ob); // передача объекта ob по ссылке
     cout << ob.getx() << endl; // x = 100
}
```

Следует заметить, что передача объекта по ссылке не всегда возможна.

```
// Пример 2. Определение класса для работы с рациональной
дробью.
// Нахождение суммы двух рациональных дробей. Использование
конструктора.

#include <iostream>
```

```
using namespace std;

class Frac
{
     double a, b;
public:
     Frac(int n, int m){a = n; b = m;} // конструктор с
параметром
     void show(){cout<<a<<'/'<<b<<endl;} // функция вывода
     double geta(){return a;}
     double getb(){return b;}
};

double sumFrac(Frac s1, Frac s2)
{
     double z = (s1.geta()*s2.getb() +
s1.getb()*s2.geta())/(s1.getb()*s2.getb());
     return z;
}

int main()
{
     int x,y;
     cout<<"Input numerator and denominator for 1 fraction: ";
     cin>>x>>y;
     Frac ob1(x, y); ob1.show();
     cout<<"Input numerator and denominator for 2 fraction: ";
     cin>>x>>y;
     Frac ob2(x, y); ob2.show();
     double rez = sumFrac(ob1, ob2);
     cout << "rez= " << rez << endl;
     return 0;
}
```

Домашнее задание

Базовый уровень

Создать класс, содержащий вещественные переменные x, y, z и методы double func(void) (вычисляющий функцию в соответствии с вариантом задания), void setx(double), void sety(double), void setz(double) (задающие переменные)

1. $$t = \frac{2\cos\left(x - \frac{\pi}{6}\right)}{0.5 + \sin^2 y}\left(1 + \frac{z^2}{3 - z^2/5}\right).$$

   При x=14.26, y=-1.22, z=3.5×10^{-2} **t=0.564849.**

2. $$u = \frac{\sqrt[3]{8 + |x-y|^2 + 1}}{x^2 + y^2 + 2} - e^{|x-y|}\left(tg^2 z + 1\right)^x.$$

При x=-4.5, y=0.75×10^{-4} ,z=0.845×10^2 **u=-55.6848.**

3. $$v = \frac{1 + \sin^2(x+y)}{\left|x - \frac{2y}{1 + x^2 y^2}\right|} x^{|y|} + \cos^2\left(arctg\frac{1}{z}\right).$$

При x=3.74×10^{-2} , y=-0.825, z=0.16×10^2 , **v=1.0553.**

4. $$w = \left|\cos x - \cos y\right|^{(1+2\sin^2 y)}\left(1 + z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4}\right).$$

При x=0.4×10^4 ,y=-0.875, z=-0.475×10^{-3} **w=1.9873.**

5. $$\alpha = \ln\left(y^{-\sqrt{|x|}}\right)\left(x - \frac{y}{2}\right) + \sin^2 arctg(z).$$

При x=-15.246, y=4.642×10^{-2} , z=20.001×10^2 α **=-182.036.**

6. $$\beta = \sqrt{10\left(\sqrt[3]{x} + x^{y+2}\right)}\left(\arcsin^2 z - |x - y|\right)$$

При x=16.55×10^{-3} , y=-2.75, z=0.15 β **=-40.630.**

7. $$\gamma = 5 arctg(x) - \frac{1}{4}\arccos(x)\frac{x + 3|x-y| + x^2}{|x-y|z + x^2}.$$

При x=0.1722, y=6.33, z=3.25×10^{-4} γ **=-205.305.**

8. $$\varphi = \frac{e^{|x-y|}|x-y|^{x+y}}{arctg(x) + arctg(z)} + \sqrt[3]{x^6 + \ln^2 y}.$$

При x=-2.235×10^{-2} , y=2.23, z=15.221 φ **=39.374.**

9. $$\psi = \left| x^{\frac{y}{x}} - \sqrt[3]{\frac{y}{x}} \right| + (y-x)\frac{\cos y - \dfrac{z}{(y-x)}}{1+(y-x)^2}.$$

При x=1.825×10^{2}, y=18.225, z=-3.298×10^{-2} ψ **=1.2131.**

10. $$b = y^{\sqrt[3]{|x|}} + \cos^3(y)\frac{|x-y|\left(1+\dfrac{\sin^2 z}{\sqrt{x+y}}\right)}{e^{|x-y|}+\dfrac{x}{2}}.$$

При x=6.251, y=0.827, z=25.001 **b=0.7121.**

11. $$c = 2^{(y^x)} + (3^x)^y - \frac{y\left(arctgz - \dfrac{\pi}{6}\right)}{|x| + \dfrac{1}{y^2+1}}.$$

При x=3.251, y=0.325, z=0.466×10^{-4} **c=4.25.**

12. $$f = \frac{\sqrt[4]{y+\sqrt[3]{x-1}}}{|x-y|(\sin^2 z + tgz)}.$$

При x=17.421, y=10.365×10^{-3}, z=0.828×10^{5} **f=0.33056.**

13. $$g = \frac{y^{x+1}}{\sqrt[3]{|y-2|}+3} + \frac{x+\dfrac{y}{2}}{2|x+y|}(x+1)^{-1/\sin z}.$$

При x=12.3×10^{-1}, y=15.4, z=0.252×10^{3} **g=82.8257.**

14. $$h = \frac{x^{y+1}+e^{y-1}}{1+x|y-tgz|}(1+|y-x|) + \frac{|y-x|^2}{2} - \frac{|y-x|^3}{3}.$$

При x=2.444, y=0.869×10^{-2}, z=-0.13×10^{3} h = **-0.49871.**

Средний уровень

Создать консольное приложение принимающее с консоли ввод рациональных дробей, вычисляющее сумму и разность двух рациональных дробей и выводящей результат в виде рациональной дроби. Каждая дробь должна быть реализована в виде класса FracNum содержащего целые числитель (numerator) и знаменатель (denominator), конструктор, позволяющий инициализировать числитель и знаменатель, методы get и set для числителя и знаменателя и метод show выводящий на экран рациональную дробь в виде: ***numerator / denominator***. В функцию сложения передавать объекты - дроби через указатель. В функцию вычитания через ссылку а в функцию отображения результата - по значению.

Высокий уровень

Создать консольное приложение принимающее с консоли ввод смешанных дробей, вычисляющее сумму, разность, произведение и частное двух смешанных дробей и выводящей результаты в виде смешанной дроби. Каждая дробь должна быть реализована в виде класса MixNum содержащего целую часть (base) числитель (numerator) и знаменатель (denominator), конструктор, позволяющий инициализировать целую часть числитель и знаменатель, методы get и set для целой части, числителя и знаменателя и метод show выводящий на экран смешанную дробь в виде ***base | numerator / denominator***. В функцию сложения передавать объекты - дроби через указатель. В функцию вычитания через ссылку а в функцию отображения результата - по значению.

Добавить в класс MixNum скрытый метод, позволяющий автоматически упрощать смешанную дробь. Для поиска наибольшего общего делителя двух чисел использовать алгоритм Евклида.

Лабораторное задание

1. Запустить интегрированную среду разработки
2. Ввести и отладить разработанное дома программное обеспечение
3. Продемонстрировать преподавателю работоспособную программу
4. Оформить протокол.

Ключевые вопросы

1. Описать процесс передачи объекта в функцию по значению и особенности этого процесса.
2. Описать процесс передачи объекта в функцию через указатель и особенности этого процесса.
3. Описать процесс передачи объекта в функцию через ссылку и особенности этого процесса.

Содержание протокола

1. Тема
2. Цель работы.
3. Исходное задание
4. Алгоритм работы программы.
5. Исходный код программы.
6. Результаты выполнения задания

7. Краткие выводы.

Лабораторная работа №3

Тема Использование дружественных функций

Цель Освоить приёмы программирования с использованием дружественных функций

Ключевые положения

В общем случае только функции класса имеют доступ к закрытым членам этого класса. Однако в **C++** существует возможность *разрешить доступ к закрытым членам* класса функциям, которые *не являются членами класса*. Для этого достаточно объявить эти функции **дружественными** по отношению к рассматриваемому классу. Для того чтобы объявить функцию дружественной некоторому классу, нужно в этот класс включить её прототип, перед которым поставить ключевое слово ***friend***.

// **Пример 3.1**. Использование дружественной функции.

```
#include <iostream>
using namespace std;
class Samp
{
     int x, y;
public:
     Samp(int n, int m){x = n; y = m;} // конструктор
     friend int fun(Samp ob); // друж. функция, параметр –
объект
};

int fun(Samp ob)
{
     if (ob.x % ob.y == 0)
          return 1;
     else
          return 0; // прямой доступ к x и y
}

int main()
{
     Samp ob(10, 5); // создание объекта ob1
     if(fun(ob)) cout<<"ob.x kratno ob.y"<<endl;
     else cout<<"ob.x no kratno ob.y"<<endl;
     return 0;
}
```

В данной программе дружественная функция **fun()** вызывается *обычным образом*. Так как дружественная функция не является членом того класса, для которого она дружественна, то её нельзя вызвать, используя имя объекта и *операции точка* **(.)** или *стрелка* **(->)**.

Дружественная функция– это функция, определённая вне класса, но имеющая доступ *ко всем членам класса.*

Функция может быть дружественной более чем к одному классу.

// **Пример 3.2.** Функция **fun()** дружественна классу **A** и классу **B**.

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

class B;

class A
{
     int data;
public:
     A(){data = 3;}
     friend int fun(A o_a, B o_b); // дружественная функция
};

class B
{
     int data;
public:
     B(){data = 7;}
     friend int fun(A o_a, B o_b);
};

int fun(A o_a, B o_b)
{
return(o_a.data + o_b.data);
}

int main()
{
     A ob_a; B ob_b;
     cout << fun(ob_a, ob_b) << endl; // 10
     return 0;
}
```

Следует обратить внимание на то, что в программе функция fun() определена только один раз, так же используется опережающее объявление для класса B в связи с тем, что параметром функции fun() является объект класса B до определения класса.
Функция может быть членом одного класса и дружественной другому классу. Рассмотрим пример.

// **Пример 3.3**. Функция **fun(*)** член класса **A** и дружественна классу **B**.

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

```
class B;
class A
{
     int data;
public:
     A(){data = 3;}
     int fun(B o_b); // функция fun()– член классаA
};

class B
{
     int data;
public:
     B(){data = 7;}
     friend int A::fun(B o_b); // функция fun()– друг классаB
};

int A::fun(B o_b){return(data + o_b.data);}

int main()
{
     A ob_a; B ob_b; // вызов функции fun()через операцию
     cout << ob_a.fun(ob_b) << endl; // точка и объект класса A
     return 0;
}
```

Правила использования дружественных функций:

- функция не является членом класса, но имеет доступ ко всем (в том числе и к *закрытым*) членам класса через объект, объявленный внутри функции или переданный ей в качестве аргумента;
- функция вызывается как обычная функция;
- прототип функции может быть объявлен как в открытой, так и в закрытой части класса, но она будет *всё равно открытой*;
- функция может быть дружественной более чем к одному классу;
- функция может быть членом одного класса и дружественной другому;
- дружественной функции указатель ***this*** не передаётся.

// **Пример 3.4.** Нахождение суммы двух рациональных дробей.

// Использование конструктора и **friend**-функции.

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

class Frac
{
     double a;
```

```
     double b;
public:
     Frac(){}; // конструктор по умолчанию
     Frac(int n, int m); // конструктор с параметром
     void show(); // функция вывода
     friend Frac sumFrac(Frac s1, Frac s2);
};

Frac::Frac(int n, int m)
{ // определение конструктора
a = n; b = m;
}

void Frac::show()
{
cout<<a<<"/"<<b<<endl;
}

// друж. функция sumFrac()
Frac sumFrac(Frac s1, Frac s2)
{
     Frac s3;
     s3.a = (s1.a * s2.b + s1.b * s2.a);
     s3.b = (s1.b * s2.b);
     return s3;
}

int main()
{
     int x, y;
     cout<<" Input numerator and denominator. for 1 fraction: ";
     cin>>x>>y;
     Frac ob1(x, y); ob1.show();
     cout<<"Vvedi chisl. i znam. for 2 fraction: ";
     cin>>x>>y;
     Frac ob2(x, y); ob2.show();
     Frac ob3 = sumFrac(ob1, ob2);
     cout<<"sum = ";
     ob3.show();
     return 0;
}
```

Домашнее задание

Создать консольное приложение принимающее с консоли ввод рациональных дробей, вычисляющее сумму и разность двух рациональных дробей и выводящей результат в виде рациональной дроби. Каждая дробь должна быть реализована в виде класса FracNum содержащего числитель (numerator) и знаменатель (denominator), конструктор, позволяющий инициализировать числитель и знаменатель, методы get и set

для числителя и знаменателя и метод show выводящий на экран рациональную дробь. Функции сложения и вычитания реализовать, как дружественные функции классов рациональных дробей.

Задание повышенной сложности: добавить в класс FracNum скрытый метод, позволяющий автоматически упрощать рациональную дробь. Для поиска наибольшего общего делителя двух чисел использовать алгоритм Евклида.

Лабораторное задание

1. Запустить интегрированную среду разработки
2. Ввести и отладить разработанное дома программное обеспечение
3. Продемонстрировать преподавателю работоспособную программу
4. Оформить протокол.

Ключевые вопросы

1. Что такое - дружественные функции и каковы особенности их использования?
2. Синтаксис описания дружественных классу функций.
3. Как сделать метод одного класса дружественным другому?
4. Каковы правила использования дружественных функций.

Содержание протокола

1. Тема
2. Цель работы.
3. Исходное задание
4. Алгоритм работы программы.
5. Исходный код программы.
6. Результаты выполнения задания
7. Краткие выводы.

Лабораторная работа №4

Тема Потоковый ввод-вывод

Цель Освоить возможности форматирования потокового ввода-вывода

Ключевые положения

Стандартные потоки ввода–вывода

При включении в программу заголовочного файла **<iostream>** для в/ы информации в стандартные потоки автоматически будут созданы следующие объекты:

- **cin**– объект класса **istream,** соответствует *стандартному потоку ввода,* даёт возможность программе вводить данные с клавиатуры;
- **cout**– объект класса **ostream,** соответствует *стандартному потоку вывода*, даёт возможность программе выводить данные на экран;
- **cerr**, **clog**– объекты класса **ostream,** соответствуют *стандартным потокам вывода сообщений об ошибках,* позволяют программе выводить на экран сообщения об ошибках.

*Операция сдвига влево***(<<)**используется для ***вывода в поток***и называется *операцией вставки в поток*. *Операция сдвига вправо***(>>)** используется для ***ввода из потока*** и называется *операцией извлечения из потока*. Эти операции обычно применяются к стандартным потокам **cin** и **cout**:

…

int a;cin>>a; // ввод с клавиатуры

cout<<"a = "<<a; // вывод на экран

Операции *извлечения из потока и вставки в поток* в качестве результата своего выполнения формируют соответственно ссылки на объект типа **istream** или **ostream**, что позволяет создавать цепочки операций.

Как и для других перегруженных операций, для *операций извлечения* и *вставки* невозможно определить приоритеты, поэтому в необходимых случаях используются скобки:

cout<<(i<<j); //*операция* << в скобках означает сдвиг влево

Данные при вводе из потока должны разделяться *пробельными символами*(*пробелами, знаками табуляции или символом перехода на новую строку*). Ввод прекращается, если очередной символ оказался недопустимым.

Операции << *и* >>перегружены для всех встроенных типов данных, строк и значений указателей, что позволяет автоматически выполнять ввод–вывод в соответствии с типом величин:

int i = 1; double d;

cin>>d; // символы из потока преобразуются в **double**

cout<<i<<' '<<d; // **int** и **double** преобразуются в символы

Числовые значения можно вводить в десятичной или шестнадцатеричной системе счисления (с префиксом**0x**), со знаком или без знака. Вещественные числа представляются в форме с фиксированной точкой или в виде мантиссы и порядка.

При *вводе строк* извлечение символов из потока происходит до ближайшего пробела. Вместо него в строку заносится *нуль-символ*–**'\0'**, который является признаком конца строки (**см. главу 10**).

Под любую величину при выводе отводится столько позиций, сколько требуется для её представления. Чтобы отделить одну величину от другой, используются пробелы:

cout<<i<<' '<<j<<""<<k;

Поскольку ввод буферизирован, помещение в буфер ввода происходит после нажатия клавиши ***Enter*** строки, после чего из буфера выполняется операция извлечения из потока. Это даёт возможность исправлять введённые символы до того, как нажата клавиша ***Enter***.

Форматирование данных

До сих пор при *вводе* или *выводе* информации в наших примерах программ действовали параметры форматирования, которые по умолчанию использует система ввода–вывода **C++**. И если они не устраивают программиста, то он может сам управлять форматом представления данных, причём разными способами.

В потоковых классах форматирование выполняется тремя способами – с помощью *флагов, манипуляторов и специальных функций форматирования*.

Флаги форматирования

В системе *ввода–вывода* **C++** каждый поток связан с набором *флагов форматирования*, которые управляют процессом форматирования данных.Если флаг формата установлен, реализуется соответствующая ему функция.

Флаги устанавливаются функцией **setf(),** а сбрасываются функцией **unsetf()**. Функции являются членами класса **ios.** Вызов функций происходит относительно конкретного потока, поэтому каждый поток отдельно поддерживает своё собственное состояние формата. Функции могут устанавливать или сбрасывать сразу несколько флагов. При сбросе флага используется формат по умолчанию:

поток. **setf**(ios::флаг1 |ios::флаг2 |ios::флаг3 …); // установка флагов

поток. **unsetf**(ios::флаг); // сброс флага

Наиболее часто используемые *флаги форматирования*:

skipws при вводе пробельные символы игнорируются;

left при выводе выравнивание по левому краю поля;

rigth при выводе выравнивание по правому краю поля;

dec десятичная система счисления (по умолчанию);

oct восьмеричная система счисления;

hex шестнадцатеричная система счисления;

scientific вывод вещественных чисел в форме мантиссы и порядка с **6** знаками после точки;

fixed вывод вещественных чисел в обычной форме с **6** десятичными знаками после точки;

showpoint вывод вещественных чисел с десятичной точкой и дробной частью, по умолчанию всего **6** знаков;

Функции форматирования

Кроме флагов форматирования в классе **ios** определены *три функции-члена*, которые устанавливают параметры формата:

- **width(**int len)– устанавливает ширину (*width*) поля вывода;
- **precision(**int num**) –** устанавливает точность (*precision*) при выводе, т.е. число цифр после точки, если вывод в форме мантиссы и порядка, иначе – общее количество цифр;
- **fill(**char ch**) –** задаёт символ заполнения (*fill*) поля вывода.

По умолчанию при выводе любого значения оно занимает столько позиций, сколько символов выводится; для вещественных чисел точность равна**6**цифрам; свободные поля заполняются пробелами.

Манипуляторы ввода-вывода

В системе *ввода–вывода* **C++** имеется ещё один способ изменения параметров форматирования, связанных с потоком. Он реализуется с помощью специальных функций, *называемых манипуляторами*, которые включаются в выражение ввода–вывода. При использовании манипуляторов, *которые принимают аргументы*, необходимо включить в программу заголовочный файл **<iomanip>.** Наиболее часто используемые манипуляторы:

endl при выводе перейти на новую строку;

flush вывести и очистить все промежуточные буферы;

dec вывод чисел в *десятичной* системе счисления

oct вывод чисел в *восьмеричной* системе счисления;

hex вывод целых чисел в **16**-ой системе счисления;

setw(int w) задаёт ширину поля вывода, равную **w**;

ws пропуск начальных пробелов;

left при выводе выравнивание по левому краю поля;

rigth при выводе выравнивание по правому краю поля;

setfill(char ch**)** устанавливает символ заполнения **ch**;

setprecision(int p**)** задаёт число знаков после десятичной точки,

равное ***p*** позициям, если вывод в форме мантиссы и порядка. Иначе – общее количество цифр.

// **Пример 4.1.** Форматирование данных.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
// вывод с использованием установок по умолчанию
cout<<123.23<<"ABCDEF"<<100<<endl;
// меняем формат с помощью флагов форматирования
cout.unsetf(ios::dec); // требуется не для всех компиляторов
cout.setf(ios::hex|ios::scientific); // вывод в форме мантиссы и
порядка
cout<<123.23<<"ABCDEF"<<100<<endl;
// использование функций формата
cout.width(10); // ширина поля 10
cout<<"ABCDEF"<<endl; // выравнивание вправо
cout.width(10);
cout.precision(4); // точность 4 цифры после точки
cout<<123.456789<<endl;
// использование манипуляторов
cout<<hex<<100<<endl; // вывод в 16-ой с/с
cout<<oct<<10<<endl; //вывод в 8-ой с/с
cout<<setw(10)<<left<<"ABCDEF "<<endl; //ширина поля 10
```

```
cout<<setprecision(2)<<1234.5678<<endl;
return 0;
}

Результат:
123.23ABCDEF100
1.232300e+002ABCDEF64
    ABCDEF
1.2346e+002
64
12
ABCDEF
1.23e+003
```

Домашнее задание

Создать консольное приложение, позволяющее по запросу пользователя преобразовывать целые десятичные, шестнадцатеричные, восьмеричные, двоичные числа из любой системы счисления в любую другую. Исходные данные выравниваются по левому краю, а результаты по правому.

Задача повышенной сложности: добавить в программу операции сложения, вычитания, умножения, целочисленного деления и остатка от деления на цело с отображением условия и результата в выбранной системе счисления.

Лабораторное задание

1. Запустить интегрированную среду разработки
2. Ввести и отладить разработанное дома программное обеспечение
3. Продемонстрировать преподавателю работоспособную программу
4. Оформить протокол.

Ключевые вопросы

1. Какие стандартные потоки ввода-вывода создаются при объявлении библиотеки <iostream>?
2. Что такое флаги форматирования? Привести примеры.
3. Что такое манипуляторы форматирования? Провести примеры.
4. Какие Вы можете назвать функции форматирования?

Содержание протокола

1. Тема
2. Цель работы.
3. Исходное задание
4. Алгоритм работы программы.
5. Исходный код программы.
6. Результаты выполнения задания
7. Краткие выводы.

Лабораторная работа №5

Тема Файловый ввод-вывод

Цель Освоить работу с текстовыми файлами через потоковый ввод-вывод с последовательным доступом

Ключевые положения

Использование файлов в программах

Файл – это совокупность данных на внешнем носителе. В **C++** *обмен информацией с файлами происходит через потоки*.

Для организации работы с файлами в программах C++ необходимо подключение заголовочного файла <**fstream**>, в котором определены три *класса файловых потоков*:

ifstream для ввода данных из файла;

ofstream для вывода данных в файл;

fstream для ввода–вывода данных в файл.

Эти классы являются производными от классов **istream**, **ostream, iostream** соответственно**,** и поэтому также имеют доступ ко всем операциям ввода–вывода, которые определены классом **ios** (см.**главу 15**).

По способу доступа файлы разделяют *на последовательные*, чтение и запись в которых производится с начала файла байт за байтом, и файлы *с произвольным доступом*, допускающие чтение и запись данных в произвольном порядке.

По способу организации различают –***текстовые*** и ***двоичные файлы***. Текстовые файлы удобны для чтения, для их редактирования можно воспользоваться обычным редактором, их можно легко перенести с одной компьютерной системы в другую. Однако при записи или считывании данных может происходить некоторое преобразование символов.

В двоичных (или бинарных) файлах запись или считывание данных выполняется без какого-либо преобразования, поэтому данные сохраняются более точно, занимают меньше места, и обработка их выполняется быстрее. Однако при переносе в другую компьютерную систему возможны проблемы, если в новой системе применяется другое внутреннее представление данных.

Обмен информацией с текстовыми файлами обычно реализуется через *текстовые потоки*, а с двоичными файлами – через *двоичные потоки*.

Обмен информацией с файлом предполагает выполнение следующих действий:

- создание потока;
- открытие файла (связь файла с потоком);

- обмен (ввод–вывод) информацией с файлом;
- закрытие файла (разрыв файла с потоком).

Для создания потока необходимо в программе создать объект соответствующего класса – для *потока ввода* необходимо объявить объект типа **ifstream, д**ля *потока вывода*– объект типа **ofstream.** Потоки, которые реализуют одновременно ввод и вывод, должны быть объявлены как объекты типа **fstream**:

ifstream fin; // входной поток (объект) **fin**

ofstream fout; // выходной поток (объект) **fout**

fstream fio; // поток ввода–вывода (объект) **fio**

Для открытия файла его нужно связать с потоком. Это можно сделать с помощью функции**open()**:

ofstream fout;// создание потока (объекта) **fout**

fout.**open**("vix.dat", ios::out);// открытие файла **vix.dat**

В большинстве же случаев для открытия файла используют конструкторы классов, которые *автоматически открывают заданный файл*:

ifstream fin("vx.dat", ios::in); // открытие файла **vx.dat** для ввода

ofstreamfout("vix.dat", ios::out); // открытие файла**vix.dat**для вывода

fstreamfio("vvx.dat", ios::in|ios::out); // открытие**vvx.dat**для в-ы

Конструкторы создают объекты соответствующего класса, открывают файл с указанным именем и связывают файл с потоком. Первый аргумент у конструкторов – ***имя файла***, и это единственно обязательный аргумент. Второй аргумент задаёт***режим доступа*** к файлу.

*Режим доступа*к файлу служит для описания характера использования файла – для чтения, для записи и т. д. В классе**ios**определены константы для указания режима доступа к файлу:

ios::in открыть файл только для чтения;

ios::out открыть файл только для записи;

ios::app открыть файл для добавления в конец файла;

ios::trunc если файл существует, удалить его;

ios::binary открытьфайл в двоичном режиме.

Режимы доступа можно объединять с помощью операции **ИЛИ**.

По умолчанию объекты класса **ofstream** открыты для вывода, а класса **ifstream**– для ввода, поэтому режимы **out** и **in** можно опускать. *По умолчанию* все файлы открываются в *текстовом режиме*.

Любой файл в **C++**, независимо от того, что в нём содержится – отформатированный текст или неформатированные данные – может быть открыт как в текстовом, так и в двоичном режиме. Но всё же, если необходимо сохранить данные в двоичном виде, лучше использовать двоичные файлы. В **C++** это делается путём использования константы **ios::binary** для указания режима доступа к файлу.

Прежде чем начать обмен данными с файлом, следует убедиться, была ли ***операция открытия файла успешной,*** так как можно допустить ошибку в имени файла или в указании пути к файлу. Например, проверить правильность открытия файла **vx**.**dat** можно следующим образом:

```
if(!fin){cout<<"File no open"; exit(1);}
```

Любой аргумент функции **exit()**, отличный от нуля, показывает, что программа прекратила выполнение из-за ошибки.

Только после того, как поток (объект) успешно соединён с файлом (*т.е. файл открыт*), можно выполнять *обмен информацией с файлом*.

Для закрытия файла(*т.е. отсоединения потока от файла*) используется функция **close(),** которая не имеет параметров и возвращаемого значения:

```
fin.close(); // закрытие файла vx.dat
```

Обнаружить конец файла можно с помощью функции **eof().** Функция возвращает значение ***true,*** если был достигнут конец файла, в противном случае функция возвращает значение ***false***.

Ввод–вывод текстовых файлов

Текстовый файл– это последовательность строк символов, разделённых *пробельными символами* (' ' – пробел; '**\t**' – горизонтальная табуляция; '**\v**' – вертикальная табуляция; '**\n**' – новая строка; '**\r**' – возврат каретки (перевод курсора в начало строки); '**\f**' – перевод страницы).

Для создания текстового файла определяют объект класса **ofstream**:

```
ofstream fout("test.txt", ios::out); // открытиефайла test.txt для записи
```

Для чтения текстового файла определяют объект класса **ifstream**:

```
ifstream fin("test.txt", ios::in); // открытие файла test.txt для чтения
```

Использование операций << и >>

После открытия файла чтение и запись данных можно выполнить с помощью операций >> и <<, которые для файлов выполняются так же, как и для *стандартных потоков ввода–вывода* (см**. главу 15**). Только в этом случае *потоки* **cin** и **cout** нужно заменить *потоками*, *которые связаны с файлом*. Различие заключается в том, как создаются потоки и как они привязываются к нужным файлам.

Как и в случае стандартных потоков, при считывании строки *операцией* **>>** извлечение символов из файлового потока происходит до ближайшего пробела, и вместо него в строку заносится символ '**\0**'.

// **Пример 16.1.**Использование операций << и >>.

```
// Создание и чтение текстовых файлов.
#include<iostream>
#include<fstream> // подключение библиотеки <fstream>,
// в которой описаны классы ifstream,ofstream
using namespace std;
int main()
{
int n;
cout<<"Input n: "<<endl; cin>>n;
srand(n);
cout<<"Write to file test"<<endl;
ofstream fout("test.txt", ios::out); // открытие файла test для
записи
if(!fout){cout<<"Error"; return(1);} // проверка открытия файла
for(int i = 1; i <= n; i++)
     {
     int x = rand() % 15 - 5;
     fout<<x<<endl; // запись в файл test
     cout<<"x = "<<x<<endl;// для контроля вывод на экран
     }
fout.close(); // закрытие файла test
cout<<"Read file test"<<endl;
int z;
ifstream fin("test.txt", ios::in); // открытие файла testдля
чтения
if(!fin){cout<<"oshibka"; exit(1);} // проверка открытия файла
while(!fin.eof())
     { // проверка на конец файла и
     fin>>z; // чтение файла test
     if(!fin.eof())cout<<"z = "<<z<<endl;
     }
fin.close(); // закрытие файла
return 0;
}
```

При использовании *операций*<< и >>для реализации файлового ввода–вывода, информация *форматируется так же*, как и для стандартных файлов.

Домашнее задание

Создать консольную программу - англо-русского словаря dictonary.exe, которая создаёт словарь в виде текстового файла. Каждое слово с переводом занимает в текстовом файле одну строку. Все строки в файле должны быть отсортированы по алфавиту. Программа должна выполнять следующие функции:

- добавить слово в словарь;
- удалить слово из словаря;
- найти и вывести на экран слова по маске поиска.

Задание повышенной сложности:

- файл должен содержать строки, как англо-русского, так и русско-английского словаря;
- сортировка слов должна игнорировать различие строчных и прописных букв;
- поиск английских слов только по англо-русской части, а поиск русских переводов - только по русско-английской части словаря.

Примечание: чтобы вставить строку в файл требуется переписать строки из одного файла в другой, вставив в нужное место новую строку. После этого старый файл уничтожить, а новый переименовать.

Лабораторное задание

1. Запустить интегрированную среду разработки
2. Ввести и отладить разработанное дома программное обеспечение
3. Продемонстрировать преподавателю работоспособную программу
4. Оформить протокол.

Ключевые вопросы

1. Какие Вам известны классы для файлового ввода-вывода в библиотеке <fstream>?
2. Какие константы из класса ios и как управляют доступом к файллам?
3. Как программа может выяснить - подключился файл к потоку или нет?
4. Как с помощью потока ввода определить достигнут ли конец файла?
5. Как работает функция eof() во входном файловом потоке?

Содержание протокола

1. Тема
2. Цель работы.
3. Исходное задание
4. Алгоритм работы программы.
5. Исходный код программы.
6. Результаты выполнения задания
7. Краткие выводы.

Лабораторная работа №6

Тема

Цель

Ключевые положения

Посимвольный ввод–вывод

Функции **get()** и**put()**является членами всех потоковых классов соответственно для ввода и для вывода. Эти функции имеют множество форматов, но чаще всего используют следующие их версии:

istream& **get**(char &ch);

ostream& **put**(char ch);

Функция **get()** считывает один символ из соответствующего потока и помещает его значение в переменную **ch**. Она возвращает ссылку на поток, связанный с предварительно открытым файлом. *При достижении конца этого файла значение ссылки станет равным нулю.*Функция **put()** записывает символ **ch** в поток и возвращает ссылку на этот поток.

Функции **get()** *считываетсимвол*из файла. Функция**put()** *записываетсимвол*в файл.

// **Пример 16.2.**Использование функций**put(), get()**.

```
// Посимвольный ввод-вывод файла.
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
char *p = "Vsem privet";
ofstreamfout ("date.txt", ios::out); // открыть файл для вывода
if (!fout)
     {cout<<"No file open\n"; return(1);}
while(*p) fout.put(*p++); // запись в файлстроки посимвольно
fout.close();
char simv;
ifstream fin("date.txt", ios::in); // открыть файл для чтения
if (!fin)
     {cout<<"No open\n"; return(1);}
while(fin)
     { // при достижении конца файла потоковый
     fin.get(simv); // объект fin примет значение false, которое
     if(fin)cout<<simv; // остановит выполнение цикла while
     }
cout<<endl;fin.close();
```

```
return(0);
}
```

Построчный ввод–вывод

Обычно построчное чтение и запись файлов работают быстрее посимвольных действий. Для чтения строки из файла используется функция **getline()** класса **ifstream.** Функция читает строку (в том числе и разделители), пока не встретит *символ новой стоки* **'\n'**, и помещает её в буфер. Имя буфера передаётся функции как первый аргумент. Максимальный размер буфера задаётся как второй аргумент функции.

// Пример 16.3.Использование функции **getline()** для чтения файла.

```
// Построчный ввод-вывод файла.
#include <iostream>
#include <fstream>
using namespace std;
const int len = 40;
int main()
{
char buf[len];
ofstream fout("date.txt", ios::out); // открыть файл для вывода
if (!fout){cout<<"No file open\n"; exit(1);}
fout<<"1111 1111\n"; // запись строк в файл;
fout<<"22222 22222\n"; // строки можно вводить,
fout<<"33333333333\n"; // как обычно, функцией gets()
fout.close();
ifstream fin("date.txt", ios::in); // открыть файл для чтения
if(!fin){cout<<"No file open\n"; exit(1);}
while(!fin.eof())
     { // чтение, пока не конец файла
     fin.getline(buf, len);
     if(!fin.eof())cout<<buf<<endl;
     }
/*while(fin)
     {
     fin.getline(buf, len);
     if(fin)cout<<buf<<endl;
     }*/
fin.close();
}
```

Домашнее задание

Лабораторное задание

5. Запустить интегрированную среду разработки
6. Ввести и отладить разработанное дома программное обеспечение
7. Продемонстрировать преподавателю работоспособную программу

8. Оформить протокол.

Ключевые вопросы

Содержание протокола

8. Тема
9. Цель работы.
10. Исходное задание
11. Алгоритм работы программы.
12. Исходный код программы.
13. Результаты выполнения задания
14. Краткие выводы.

Лабораторная работа №7

Тема Ввод–вывод двоичных файлов

Цель

Ключевые положения

Двоичный файл – это набор двоичной информации. Для записи данных в файл создают объект класса **ofstream,** для чтения данных из файла – объект класса **ifstream,** а для в/в данных – объект класса **fstream.** Для того чтобы открыть файл как двоичный, необходимо задать режим доступа к файлу **ios::binary.**

Для записи данных в двоичный файл и их считывания можно использовать *операции >> и* <<. Например, записать значения типа ***double*** в файл и их считать можно следующим образом:

double d = 5.555; double s;

ofstream **fout**("test", ios::out | **ios::binary**); // открыть для вывода

fout<<d;

ifstream **fin**("test", ios::in | **ios::binary**); // открыть для ввода

fin>>s; cout<<"s ="<<s<<endl; ...

Язык **C++** также предоставляет широкий набор функций для работы с двоичными файлами, которые дают возможность точно реализовывать процессы считывания информации из файлов и записи в файлы.

С помощью функции **put()** можно*записать байт*, а с помощью функции**get() –** *считатьбайт*.

Для считывания и записи группы *байтов*(*блоков двоичных данных*) используют функции**read()** и **write(),** прототипыкоторых имеют следующий вид:

istream& **read(**char *** buf**, type **num);**

ostream& **write(**const char *** buf**, type **num);**

Функция **read()** считывает **num** байтов данных из связанного с файлом потока и помещает их в буфер, адресуемый параметром**buf.**Функция**write()** записывает**num** байтов данных в связанный с файлом поток из буфера, адресуемого параметром**buf.** Тип**type**должен быть определён как некоторая разновидность целочисленного типа.

При считывании конца файла функция **read()** возвращает значение**0.**

Приведение типа к **(char*)**в функциях**read()**и**write()** необходимо, если буфер ввода–вывода не определён как символьный массив, так как в**C++**указатель на один тип не преобразуется автоматически в указатель на другой тип.

Если при считывании данных конец файла достигнут до того, как было считано **num**символов, выполнение функции**read()**просто прекращается.

Узнать, сколько байтов было считано, можно с помощью функции **gcount().** *Функция возвращает количество символов, считанных во время последней операции двоичного ввода*.

// **Пример 16.4.** Запись в двоичный файл целых чисел. Чтение

```
// созданного файла. Добавление данных в конец файла.
#include<iostream>
#include<fstream> // подключение библиотеки <fstream.h>
using namespace std;
int main()

int n, a;
cout<<"Input n "; cin>>n;
ofstream fout("masBin", ios::binary);
cout<<"Input numbers\n";
for(int i = 0; i < n; i++)
     { // запись в файл
     cin>>a; fout.write((char*) &a, 4);
     }
fout.close();
ifstream fin("masBin", ios::binary);
while(fin)
     {
     fin.read((char*)&a, 4); // чтение файла
     if(fin)cout<<a<<' '; // вывод для контроляна экран
     }
cout<<endl; fin.close();
ofstream faout("masBin", ios::binary | ios::app);
int dop = 55;
faout.seekp(8, ios::beg); // всё равно добавление в конец файла
for(int i = 0;i < n;i++)
faout.write((char*) &dop, 4);
faout.close();
ifstream fain("masBin", ios::binary);
while(fain)
     {
     fain.read((char*)&a, 4); // чтение файла
     if(fain)cout<<a<<' '; // вывод для контроляна экран
     }
cout<<endl; fain.close();
}
```

Функцию **write()**можно применять для записи в файл целого массива, который к моменту записи в файл должен быть сформирован и заполнен данными, а функцию **read() –** для считывания целого массива.

// **Пример 16.5.** Использование функций **read(), write().**

// Чтение и запись блоков двоичных файлов.

```
#include <iostream>
#include <fstream>
using namespace std;
const int n = 5;
int main()
{
ofstream out("test", ios::out | ios::binary); // открытие файла
if (!out){cout<<"File no open\n "; return(1);}
int array[] = {1, 2, 3, 4, 5};
out.write((char *)array, sizeof(array)); // запись в файл блока
данных
out.close ();
ifstream in("test", ios::in |ios::binary);
if(!in) { cout<<"File no open\n"; exit(1);}
in.read((char *) array, sizeof(array)); // чтение из файла блока
for (int i = 0; i < n; i++)
cout<<array[i]<<' '; //1 2 3 4 5
cout<<'\n';
cout<<"Number of chars = "<<in.gcount()<<endl; // 20
in.close ();
return 0;
}
```

Домашнее задание

Лабораторное задание

9. Запустить интегрированную среду разработки
10. Ввести и отладить разработанное дома программное обеспечение
11. Продемонстрировать преподавателю работоспособную программу
12. Оформить протокол.

Ключевые вопросы

Содержание протокола

15. Тема
16. Цель работы.
17. Исходное задание
18. Алгоритм работы программы.

19. Исходный код программы.
20. Результаты выполнения задания
21. Краткие выводы.

Лабораторная работа №8

Тема Произвольный доступ к файлам

Цель

Ключевые положения

В **C++** можно получить доступ к файлу в произвольном порядке. Существует множество способов создания файлов с произвольным доступом, и наиболее простым из них является *требование, чтобы все записи в файле были одинаковой фиксированной длины*.

В системе ввода–вывода **C++** предусмотрена возможность управления двумя указателями, связанными с файлом. Эти, так называемые ***get-*** и ***put-****указатели,* определяют, в каком месте файла должна выполниться следующая операция ввода или вывода соответственно. При каждом выполнении операции ввода или вывода соответствующий указатель автоматически перемещается в указанную позицию. Как класс **istream**, так и класс **ostream,** содержат функции для управления этими указателями – функцию s**eekg()** и функцию **seekp(),** прототипы которых имеют следующий вид:

istream& **seekg**(offset, origin);

ostream& **seekp**(offset,origin);

Функция **seekg()** перемещает текущий ***get-***указатель соответствующего файла на ***offset-***байт относительно позиции, заданной параметром ***origin***.

Функция **seekp()** перемещает текущий ***put***-указатель соответствующего файла на ***offset-***байт относительно позиции, заданной параметром ***origin.*** Параметр ***origin*** определён как константа в классе **ios** и может принимать следующие значения:

ios:: beg поиск с начала файла

ios::cur поиск от текущей позиции в файле

ios::end поиск с конца файла

Используя функции **seekg()** и **seekp(),** можно получить доступ к информации в файле в произвольном порядке.

В общем случае произвольный доступ для операций ввода-вывода должен выполняться *только для файлов, открытых в двоичном режиме*. Преобразования символов, которые могут происходить в текстовых файлах, могут привести к тому, что запрашиваемая позиция файла не будет соответствовать его реальному содержимому.

Следующая программа демонстрирует использование функций **seekg()** и **seekp.** В программе записывается в указанную позицию символ ‘X’. Также в программе выполняется вывод файла с указанной позиции. При этом следует обратить внимание на

то, что обрабатываемый файл должен быть открыт для выполнения операций ввода–вывода.

// **Пример 16.6**. Использование функций **seekg()** и **seekp().**

// Произвольный доступ к файлу.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
char simv; char *p = "Vsem privet";
ofstream fout("date", ios::out | ios::binary);
if (!fout)
     {cout<<"No file open\n"; return(1);}
while(*p)fout.put(*p++); // запись в файл строки
fout.close();
fstream out("date",ios::in | ios::out |ios::binary);
out.seekp(2, ios::beg);
out.put('X') ; // запись символа 'X' в файл
out.seekg(0, ios::beg);
while(out)
     { // проверка на конец файла,
     out.get(simv); // чтение файла и вывод
     if(out)cout<<simv; // VsXm privet
     }
cout<<endl; out.close();
ifstream in("date",ios::in | ios::binary);
in.seekg(2, ios::beg);
while(in.get(simv)) // вывод файла с указанной позиции
cout<<simv;
in.close(); // Xm privet
cout<<endl;
return 0;
}
```

// Пример 16.7. Произвольный доступ. Функции read(), write().

```
#include <iostream>
#include <fstream>
using namespace std;
const int n = 5;
int main()
{
ofstream fout ("test", ios::out | ios::binary); // открытие
файла
if (!fout)
     { cout<<"File no open\n "; return(1);}
int array[] = {1, 2, 3, 4, 5};
```

```
fout.write((char *)array, sizeof(array)); // запись в файл
fout.close();
fstream out("test", ios::in | ios::out | ios::binary);
if(!out)
     {cout<<"File no open\n "; return 1;}
out.seekp(4, ios::beg); int z = 7;
out.write((char *)&z, sizeof(z)); // запись числа 7.
out.seekp(8, ios::beg);
out.put('A') ;out.close(); // запись символа 'A'
ifstream fin("test", ios::in | ios::binary);
if(!fin)
     {cout<<"File no open\n"; return 1;}
fin.read((char *) array, sizeof(array));
for(int i = 0; i < n; i++)
cout<<array[i]<<' '; // 1 765 4 5
cout<<endl;
fin.close();
return 0;
}
```

Домашнее задание

Лабораторное задание

13. Запустить интегрированную среду разработки
14. Ввести и отладить разработанное дома программное обеспечение
15. Продемонстрировать преподавателю работоспособную программу
16. Оформить протокол.

Ключевые вопросы

Содержание протокола

22. Тема
23. Цель работы.
24. Исходное задание
25. Алгоритм работы программы.
26. Исходный код программы.
27. Результаты выполнения задания
28. Краткие выводы.