УЗБЕКСКОЕ АГЕНТСТВО СВЯЗИ И ИНФОРМАТИЗАЦИИ
ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
Кафедра "Технология программирования"

МЕТОДИЧЕСКОЕ ПОСОБИЕ ДЛЯ ЛАБОРАТОРНЫХ ЗАНЯТИЙ ПО ДИСЦИПЛИНЕ

"Объектно-ориентированные языки программирования"

для студентов направлений образования

5523500 – Информатика и информационные технологии

5523600 – Информационная безопасность

5523700 – Электронная коммерция

Ташкент 2008

Авторы: Ш. А. Назиров, Р. В. Кабулов, Н. А. Арипова. Методическое пособие по лабораторным занятиям дисциплины «Объектно – ориентированные языки программирования». /ТУИТ. 95 с. Ташкент, 2008.

Цель пособия – закрепить знания, полученные при изучении теоретической части курсов и получить практические навыки разработки и отладки объектно-ориентированных программ. Пособие состоит из восьми лабораторных работ. Работы посвящены объектно- ориентированному программированию с использованием классов, наследования, перегрузки операций, шаблонов и исключительных ситуаций. Кроме этого даны работы по использованию стандартной библиотеки потоковых классов и библиотеки STL. Пособие предназначено для преподавателей и студентов высших учебных заведений.

Напечатано на основе утверждения учебно-методическим советом Ташкентского Университета Информационных Технологий.

№	Тема	Часы
1.	Структуры и динамические массивы	2
2.	Классы и объекты	2
3.	Наследование	2
4.	Перегрузка операций и шаблоны классов	2
5.	Потоковые классы и исключительные ситуации	2
6.	Функции и методы высших порядков	2
7.	Стандартная библиотека шаблонов STL	2
8.	Алгоритмы STL	2
	Итого	16

Рецензент:
доцент УзНУ
кандидат физ. –мат. наук, А. Хайларов

Старший научный сотрудник
Института Математики и
Информационных технологии
кандидат физ. –мат. наук, Ф.Н.Нуралиев

Лабораторная работа №1

Тема: Структуры и динамические массивы.

Цель. Структуры и динамические массивы. Знакомство с динамическими информационными структурами на примере однонаправленного списка.

I.Краткие теоретические сведения

Структуры

Структура – это объединенное в единое целое множество поименованных элементов данных. Элементы структуры (поля) могут быть различного типа, они все должны иметь различные имена.

Для инициализации структур значения ее полей перечисляют в фигурных скобках.

Примеры:

1. **struct Student**
{
char name[20];
int kurs;
float rating;
};
Student s={"Иванов",1,3.5};
2. **struct**
{
char name[20];
char title[30];
float rate;
}employee={"Петров", "директор",10000};

Присваивание структур

Для переменных одного и того же структурного типа определена операция присваивания. При этом происходит поэлементное копирование.

Student ss=s;

Доступ к элементам структур

Доступ к элементам структур обеспечивается с помощью уточненных имен:

Имя_структуры.имя_элемента

employee.name – указатель на строку «Петров»;

employee.rate – переменная целого типа со значением 10000

Динамические массивы.

При традиционном определении массива:

тип имя_массива [количество_элементов];

общее количество памяти, выделяемой под массив, задается определением и равно количество_элементов * sizeof(тип).

Но иногда бывает нужно чтобы память под массив выделялась для решения конкретной задачи, причем ее размеры заранее не известны и не могут быть фиксированы.

Формирование массивов с переменными размерами можно организовать с помощью указателей и средств динамического распределения памяти двумя способами:

1) с использованием библиотечных функций, описанных в заголовочных файлах alloc.h и stdlib.h (стандартный Си);
2) с использованием операций new и delete (Си++).

Формирование динамических массивов с использованием операций new и delete

Для динамического распределения памяти используются операции new и delete. Операция

new имя_типа

или

new имя_типа инициализатор

позволяет выделить и сделать доступны свободный участок памяти, размеры которого соответствуют типу данных, определяемому именем типа. В выделенный участок заносится значение определяемое инициализатором, который не является обязательным параметром. В случае успешного выделения памяти операция возвращает адрес начала выделенного участка памяти, если участок не может быть выделен, то возвращается NULL.

Примеры:

1) int *i;
   i=new int(10);
2) float *f;
   f=new float;
3) int *mas=new[5];

В примерах 1, 2 показано как выделить память под скалярные переменные, пример 3 показывает выделение памяти под массив переменных.

Операция delete указатель освобождает участок памяти ранее выделенный операцией new.

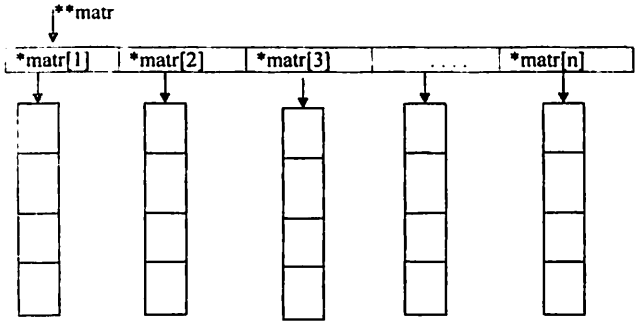
Пример:

Функция для формирования двумерного динамического массива

```
int ** make_matr(int n)
{
int **matr;
int i,j;
matr=new int*[n];
for (i=0;i<n;i++)
```

```
{
matr[i]=new int[n];
for (j=0;j<n;j++)
matr[i][j]=random(10);
}
return matr;
}
```

При формировании матрицы сначала выделяется памяти для массива указателей на одномерные массивы, а затем в цикле с параметром выделяется память под n одномерных массивов.



Чтобы освободить память необходимо выполнить цикл для освобождения одномерных массивов

```
for(int i=0;i<n;i++)
delete matr[i];
```

После этого освобождаем память на которую указывает указатель matr

```
delete [] matr;
```

Динамические структуры данных

Во многих задачах требуется использовать данные, у которых конфигурация, размеры и состав могут меняться в процессе выполнения программы. Для их представления используют динамические информационные структуры. К таким структурам относят:

9. **линейные списки;**
10. стеки;
11. очереди;
12. бинарные деревья;

Они отличаются способом связи отдельных элементов и допустимыми операциями. Динамическая структура может занимать несмежные участки динамической памяти.

Наиболее простой динамической структурой является линейный однонаправленный список, элементами которого служат объекты структурного типа.

Линейный однонаправленный список

Описание простейшего элемента такого списка выглядит следующим образом:

```
struct имя_типа
{
информационное поле;
адресное поле;
};
```

Информационное поле – это поле любого, ранее объявленного или стандартного, типа;

адресное поле – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес следующего элемента списка.

Информационных полей может быть несколько.

Примеры.

```
1. struct Node
{
int key;//информационное поле
Node*next;//адресное поле
};
2. struct point
{
char*name;//информационное поле
int age;//информационное поле
point*next;//адресное поле
};
```

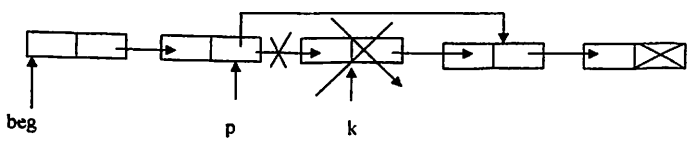
Каждый элемент списка содержит ключ, который идентифицирует этот элемент. Ключ обычно бывает либо целым числом, либо строкой.

Над списками можно выполнять следующие операции:

- начальное формирование списка (создание первого элемента);
- добавление элемента в конец списка;
- добавление элемента в начало списка;
- удаление элемента с заданным номером;
- чтение элемента с заданным ключом;
- вставка элемента в заданное место списка (до или после элемента с заданным ключом);
- упорядочивание списка по ключу

и др.

Удаление из однонаправленного списка элемента с номером k .



II. Постановка задачи

1. Написать программу, в которой создаётся одномерный динамический массив и выполнить его обработку в соответствии со своим вариантом.

2. Написать программу, в которой создаётся двумерный динамический массив и выполнить его обработку в соответствии со своим вариантом.

3. Написать программу, в которой создаются динамические структуры и выполнить их обработку в соответствии со своим вариантом.

Для каждого вариант разработать следующие функции:

1. Создание списка.
2. Добавление элемента в список (в соответствии со своим вариантом).
3. Удаление элемента из списка (в соответствии со своим вариантом).
4. Печать списка.
5. Запись списка в файл.
6. Уничтожение списка.
7. Восстановление списка из файла.

III. Варианты

1.1. Сформировать одномерный массив. Удалить из него элемент с заданным номером, добавить элемент с заданным номером;

1.2. Сформировать двумерный массив. Удалить из него столбец с заданным номером, добавить в него столбец с заданным номером;

1.3. Сформировать однонаправленный список слов. Удалить из него слово с заданным номером, добавить слово с заданным номером;

2.1. Сформировать одномерный массив. Удалить из него элемент с заданным ключом, добавить элемент перед элементом с заданным ключом;

2.2. Сформировать двумерный массив. Удалить из него строку с заданным ключом, добавить строку перед строкой заданным ключом;

2.3. Сформировать однонаправленный список слов. Удалить из него слово с заданным ключом, добавить слово перед словом с заданным ключом;

3.1. Сформировать одномерный массив. Удалить из него К элементов, начиная с заданного номера, добавить элемент перед элементом с заданным ключом;

3.2. Сформировать двумерный массив. Удалить из него К строк, начиная с заданного номера, добавить строку перед строкой с заданным ключом;

3.3. Сформировать однонаправленный список слов. Удалить из него К слов, начиная с заданного номера, добавить слово перед словом с заданным ключом.

4.1. Сформировать одномерный массив. Удалить из него элемент с заданным номером, добавить К элементов, начиная с заданного номера;

4.2. Сформировать двумерный массив. Удалить из него строку с заданным номером, добавить К строк, начиная с заданного номера;

4.3. Сформировать однонаправленный список слов. Удалить из него слово с заданным номером, добавить К слов, начиная с заданного номера;

5.1. Сформировать одномерный массив. Удалить из него К элементов, начиная с заданного номера, добавить К элементов, начиная с заданного номера;

5.2. Сформировать двумерный массив. Удалить из него К столбцов, начиная с заданного номера, добавить К столбцов, начиная с заданного номера;

5.3. Сформировать однонаправленный список слов. Удалить из него К слов, начиная с заданного номера, добавить К слов, начиная с заданного номера;

6.1. Сформировать одномерный массив. Удалить из него элемент с заданным номером, добавить элемент в начало массива.

6.2. Сформировать двумерный массив. Удалить из него строку с заданным номером, добавить строку в начало массива.

6.3. Сформировать однонаправленный список слов. Удалить из него слово с заданным номером, добавить слово в начало массива.

7.1. Сформировать одномерный массив. Удалить из него первый элемент, добавить элемент в конец массива.

7.2. Сформировать двумерный массив. Удалить из него первую строку, добавить строку в конец массива.

7.3. Сформировать однонаправленный список слов. Удалить из него первое слово, добавить слово в конец массива.

8.1. Сформировать одномерный массив. Удалить из него элемент после элемента с заданным номером, добавить К элементов в начало массива.

8.2. Сформировать двумерный массив. Удалить из него элемент после элемента с заданным номером, добавить К элементов в начало массива.

8.3. Сформировать однонаправленный список слов. Удалить из него слово после слова с заданным номером, добавить К слов в начало массива.

9.1. Сформировать одномерный массив. Удалить из него К элементов перед элементом с заданным номером, добавить К элементов в конец массива.

9.2. Сформировать двумерный массив. Удалить из него К строк перед строкой с заданным номером, добавить К столбцов в конец массива.

9.3. Сформировать однонаправленный список слов. Удалить из него К слов перед словом с заданным номером, добавить К слов в конец массива.

10.1. Сформировать одномерный массив. Добавить в него элемент с заданным номером, удалить К элементов из конца массива

10.2. Сформировать двумерный массив. Добавить в него строку с заданным номером, удалить К столбцов из конца массива

10.3. Сформировать однонаправленный список слов. Добавить в него слово с заданным номером, удалить К слов из конца массива.

11.1. Сформировать одномерный массив. Удалить из него элемент с заданным ключом, добавить элемент с указанным номером.

11.2. Сформировать двумерный массив. Удалить из него столбец с заданным ключом, добавить строку с указанным номером.

11.3. Сформировать однонаправленный список слов. Удалить из него слово с заданным ключом, добавить слово с указанным номером.

12.1. Сформировать одномерный массив. Удалить элемент из него элементы с одинаковыми ключевыми полями. Добавить элемент после элемента с заданным ключевым полем.

12.2. Сформировать двумерный массив. Удалить из него строки с одинаковыми ключевыми полями. Добавить элемент после элемента с заданным ключевым полем.

12.3. Сформировать однонаправленный список слов. Удалить элемент из него слова с одинаковыми ключевыми полями. Добавить слово после слова с заданным ключевым полем.

13.1. Сформировать одномерный массив. Удалить из него К первых элементов, добавить элемент после элемента с указанным номером.

13.2. Сформировать двумерный массив. Удалить из него К первых строк, добавить столбец с указанным номером.

13.3. Сформировать однонаправленный список слов. Удалить из него К первых слов, добавить слово после слова с указанным номером.

14.1. Сформировать одномерный массив. Удалить из него К элементов с указанными номерами. Добавить К элементов с указанными номерами.

14.2. Сформировать двумерный массив. Удалить из него К строк с указанными номерами. Добавить К строк с указанными номерами.

14.3. Сформировать однонаправленный список слов. Удалить из него К слов с указанными номерами. Добавить К слов с указанными номерами.

15.1. Сформировать одномерный массив. Удалить К элементов из конца массива. Добавить элемент после элемента с заданным ключом.

15.2. Сформировать двумерный массив. Удалить из него К строк из конца массива, добавить строку после строки с заданным ключом;

15.3. Сформировать однонаправленный список слов. Удалить К слов из конца массива. Добавить слово после слова с заданным ключом.

16.1. Сформировать одномерный массив. Удалить элемент с заданным ключом. Добавить К элементов в конец массива.

16.2. Сформировать двумерный массив. Удалить столбец с заданным ключом. Добавить К столбцов в конец массива.

16.3. Сформировать однонаправленный список слов. Удалить слово с заданным ключом. Добавить К слов в конец массива.

17.1. Сформировать одномерный массив. Удалить элемент с заданным номером. Добавить К элементов в начало массива.

17.2. Сформировать двумерный массив. Удалить столбец с заданным номером. Добавить К столбцов в начало массива.

17.3. Сформировать однонаправленный список слов. Удалить слово с заданным номером. Добавить К слов в начало массива.

18.1. Сформировать одномерный массив. Удалить элемент с заданным ключом. Добавить К элементов в начало массива.

18.2. Сформировать двумерный массив. Удалить столбец с заданным ключом. Добавить К столбцов в начало массива.

18.3. Сформировать однонаправленный список слов. Удалить слово с заданным ключом. Добавить К слов в начало массива.

19.1. Сформировать одномерный массив. Удалить К элементов с заданными номерами. Добавить К элементов в начало массива.

19.2. Сформировать двумерный массив. Удалить К столбцов с заданными номерами. Добавить К столбцов в начало массива.

19.3. Сформировать однонаправленный список слов. Удалить К слов с заданными номерами. Добавить К слов в начало массива.

20.1. Сформировать одномерный массив. Удалить элемент с заданным ключом. Добавить по К элементов в начало и в конец массива.

20.2. Сформировать двумерный массив. Удалить строку с заданным ключом. Добавить по К строк в начало и в конец массива.

20.3. Сформировать однонаправленный список слов. Удалить слово с заданным ключом. Добавить по К слов в начало и в конец массива.

21.1. Сформировать одномерный массив. Удалить элементы перед и после элемента с заданным ключом. Добавить по К элементов в начало и в конец массива.

21.2. Сформировать двумерный массив. Удалить строки перед и после строки с заданным ключом. Добавить по К строк в начало и в конец массива.

21.3. Сформировать однонаправленный список слов. Удалить слова перед и после слова с заданным ключом. Добавить по К слов в начало и в конец массива.

22.1. Сформировать одномерный массив. Удалить элемент с заданным ключом. Добавить К элементов перед элементом с заданным ключом.

22.2. Сформировать двумерный массив. Удалить строку с заданным ключом. Добавить К строк перед строкой с заданным ключом.

22.3. Сформировать однонаправленный список слов. Удалить слово с заданным ключом. Добавить К слов перед словом с заданным ключом.

23.1. Сформировать одномерный массив. Удалить элемент с заданным ключом. Добавить К элементов после элемента с заданным ключом.

23.2. Сформировать двумерный массив. Удалить строку с заданным ключом. Добавить К строк после строки с заданным ключом.

23.3. Сформировать однонаправленный список слов. Удалить слово с заданным ключом. Добавить К слов после слова с заданным ключом.

24.1. Сформировать одномерный массив. Удалить элемент с заданным номером. Добавить по К элементов перед и после элемента с заданным ключом.

24.2. Сформировать двумерный массив. Удалить столбец с заданным номером. Добавить по К столбцов перед и после столбца с заданным ключом.

24.3. Сформировать однонаправленный список слов. Удалить слово с заданным номером. Добавить по К слов перед и после слова с заданным ключом.

25.1. Сформировать одномерный массив. Удалить элемент с заданным ключом. Добавить К элементов перед элементом с заданным номером.

25.2. Сформировать двумерный массив. Удалить столбец с заданным ключом. Добавить К столбцов перед столбцом с заданным номером.

25.3. Сформировать однонаправленный список слов. Удалить слово с заданным ключом. Добавить К слов перед словом с заданным номером.

IV. Содержание отчета

1. Титульный лист: название дисциплины; номер и наименование работы; фамилия, имя, отчество студента; дата выполнения.
2. Постановка задачи.
3. Функции для формирования массива, печати массива, преобразования массива, удаления массива.
4. Результаты выполнения работы.

V. Методические указания

Динамический массив

1. Ввести размер массива;
2. Сформировать массив с помощью операции new;
3. Заполнить массив (можно с помощью датчика случайных чисел);
4. Выполнить задание варианта, сформировать новый массив-результат;
5. Напечатать массив-результат;
6. Удалить динамические массивы с помощью операции delete.

Список

1. Написать функцию для создания списка. Функция может создавать пустой список, а затем добавлять в него элементы.

2. Написать функцию для печати списка. Функция должна предусматривать вывод сообщения, если список пустой.

3. Написать функции для удаления и добавления элементов списка в соответствии со своим вариантом.

4. Выполнить изменения в списке и печать списка после каждого изменения.

5. Написать функцию для записи списка в файл.

6. Написать функцию для уничтожения списка.

7. Записать список в файл, уничтожить его и выполнить печать (при печати должно быть выдано сообщение "Список пустой").

8. Написать функцию для восстановления списка из файла.

9. Восстановить список и распечатать его.

10. Уничтожить список.

11. Создание и печать однонаправленного списка

```
#include <iostream.h>
#include<string.h>
//описание структуры
struct point
{char *name;//информационное поле
int age;//информационное поле
point*next;//адресное поле
};

point* make_point()
//создание одного элемента
{
 point*p=new(point);//выделить память
 char s[20];
 cout<<"\nEnter the name:";
 cin>>s;
p->name=new char[strlen(s)+1];//выделить память под динамическую строку символов
 strcpy(p->name,s);//записать информацию в строку символов
 cout<<"\nEnter the age";
 cin>>p->age;
 p->next=0;//сформировать адресное поле
return p;
}
void print_point(point*p)
//печать информационных полей одного элемента списка
{
 cout<<"\nNAME:"<<p->name;
 cout<<"\nAGE:"<<p->age;
 cout<<"\n-------------------------\n";
}
```

```
point* make_list(int n)
//формирование списка из n элементов
{
 point* beg=make_point();//сформировать первый элемент
 point*r;
 for(int i=1;i<n;i++)
 {
        r=make_point();//сформировать следующий элемент
        //добавление в начало списка
        r->next=beg;//сформировать адресное поле
        beg=r;//изменить адрес первого элемента списка
 }
 return beg;//вернуть адрес начала списка
}
int print_list(point*beg)
//печать списка, на который указывает указатель beg
{
 point*p=beg;//p присвоить адрес первого элемента списка
int k=0;//счетчик количества напечатанных элементов
 while(p)//пока нет конца списка
 {
        print_point(p);//печать элемента, на который указывает элемент p
        p=p->next;//переход к следующему элементу
k++;
 }
return k;//количество элементов в списке
}

void main()
{
 int n;
 cout<<"\nEnter the size of list";
        cin>>n;
 point*beg=make_list(n);//формирование списка
if(!print_list(beg)) cout<<"\nThe list is empty";}//печать списка
```

12. Удаление из однонаправленного списка элемента с номером k .

```
point*del_point(point*beg,int k)
//удаление элемента с номером к
{
 point*p=beg,//поставить вспомогательную переменную на начало списка
*r;//вспомогательная переменная для удаления
 int i=0;//счетчик элементов в списке
 if(k==0)
```

```
    {//удалить первый элемент
        beg=p->next;
        delete[]p->name;//удалить динамическое поле name
        delete[]p;//удалить элемент из списка
        return beg;//вернуть адрес первого элемента списка
    }
    while(p)//пока нет конца списка
    {
        if(i==k-1)//дошли до элемента с номером k-1, чтобы поменять его
поле next
        {//удалить элемент
            r=p->next;//поставить r на удаляемый элемент
            if(r)//если p не последний элемент
            {
            p->next=r->next;//исключить r из списка
            delete[]r->name;//удалить динамическое поле name
            delete[]r;//удалить элемент из списка
            }
        else p->next=0;//если p -последний элемент, то в поле next
присвоить NULL
        }
        p=p->next;//переход к следующему элементу списка
        i++;//увеличить счетчик элементов
    }
   return beg;//вернуть адрес первого элемента
   }
```

Лабораторная работа №2

Тема: Классы и объекты.

Цель. Получить практические навыки реализации классов на C++. Выполнить исследование вызовов конструкторов и деструкторов. Получить практические навыки создания объектов-массивов и списков.

I. Краткие теоретические сведения

Класс

С точки зрения синтаксиса, класс в C++ – это структурированный тип, образованный на основе уже существующих типов.

Функции – это методы класса, определяющие операции над объектом.

Данные – это поля объекта, образующие его структуру. Значения полей определяет состояние объекта.

Доступность компонентов класса

Для изменения видимости компонент в определении класса можно использовать спецификаторы доступа: **public, private, protected**.

Общедоступные (public) компоненты доступны в любой части программы. Они могут использоваться любой функцией как внутри данного класса, так и вне его.

Собственные (private) компоненты локализованы в классе и не доступны извне. Они могут использоваться функциями – членами данного класса и функциями – "друзьями" того класса, в котором они описаны.

Защищенные (protected) компоненты доступны внутри класса и в производных классах.

Пример:

```
class complex
{
 double re, im;              // private по умолчанию
 public:
 double real(){return re;}
 double imag(){return im;}
 void set(double x,double y){re = x; im = y;}
};
```

Конструктор

Для инициализации объектов класса в его определение можно явно включить специальную компонентную функцию, называемую **конструктором**.

Имя этой компонентной функции по правилам языка C++ должно совпадать с именем класса. Такая функция автоматически вызывается при определении или размещении в памяти с помощью оператора new каждого объекта класса.

Пример:

complex(double re1 = 0.0,double im1 = 0.0){re = re1; im = im1;}

Конструктор выделяет память для объекта и инициализирует данные – члены класса.

Конструктор имеет ряд особенностей:

• Для конструктора не определяется тип возвращаемого значения. Даже тип void не допустим.

•Указатель на конструктор не может быть определен, и соответственно нельзя получить адрес конструктора.

•Конструкторы не наследуются.

Конструктор всегда существует для любого класса, причем, если он не определен явно, он создается автоматически. По умолчанию создается конструктор без параметров и конструктор копирования. Если конструктор описан явно, то конструктор по умолчанию не создается. По умолчанию конструкторы создаются общедоступными (public).

Параметром конструктора не может быть его собственный класс, но может быть ссылка на него (T&). Без явного указания программиста конструктор всегда автоматически вызывается при определении (создании) объекта. В этом случае вызывается конструктор без параметров. Для явного вызова конструктора используются две формы.

Первая форма допускается только при не пустом списке фактических параметров. Она предусматривает вызов конструктора при определении нового объекта данного класса:

complex ss (5.9,0.15);

Вторая форма вызова приводит к созданию объекта без имени:

complex ss = complex (5.9,0.15);

Существуют два способа инициализации данных объекта с помощью конструктора. Первый способ, передача значений параметров в тело конструктора. Второй способ предусматривает применение списка инициализаторов данного класса. Этот список помещается между списком параметров и телом конструктора. Каждый инициализатор списка относится к конкретному компоненту и имеет вид:

имя_данного (выражение)

Примеры:

```
class A
{
 int i; float e; char c;
public:
 A(int ii, float ee, char cc) : i(8), e( i * ee + ii ),c(cc){}
...
};
```

Класс “символьная строка”.

class string

```
{
 char *ch; // указатель на текстовую строку
 int len;     // длина текстовой строки
public:
 // конструкторы
 // создает объект – пустая строка
 string(int N = 80): len(0){ch = new char[N+1]; ch[0] = '\0';}
 // создает объект по заданной строке
 string(const char *arch){len = strlen(arch);
    ch = new char[len+1]; strcpy(ch,arch);}
 // компоненты-функции
 // возвращает ссылку на длину строки
 int& len_str(void){return len;}
 // возвращает указатель на строку
 char *str(void){return ch;}
...};
```

Здесь у класса string два конструктора – перегружаемые функции.

По умолчанию создается также конструктор копирования вида T::T(const T&), где T – имя класса. Конструктор копирования вызывается всякий раз, когда выполняется копирование объектов, принадлежащих классу. В частности он вызывается:

а) когда объект передается функции по значению;

б) при построении временного объекта как возвращаемого значения функции;

в) при использовании объекта для инициализации другого объекта.

Если класс не содержит явным образом определенного конструктора копирования, то при возникновении одной из этих трех ситуаций производится побитовое копирование объекта. Побитовое копирование не во всех случаях является адекватным. Именно для таких случаев и необходимо определить собственный конструктор копирования. Например, в классе string:

```
string(const string& st)
{len=strlen(st.len);
ch=new char[len+1];
strcpy(ch,st.ch); }
```

Деструктор

Динамическое выделение памяти для объекта создает необходимость освобождения этой памяти при уничтожении объекта. Специальный компонент класса – **деструктор** класса обеспечивает автоматическое освобождение памяти.

Имя деструктора совпадает с именем его класса, но предваряется символом "~" (тильда).

Деструктор не имеет параметров и возвращаемого значения. Вызов деструктора выполняется не явно (автоматически), как только объект класса уничтожается.

Если в классе деструктор не определен явно, то компилятор генерирует деструктор по умолчанию, который просто освобождает память, занятую данными объекта. В тех случаях, когда требуется выполнить освобождение и других объектов памяти, например область, на которую указывает ch в объекте string, необходимо определить деструктор явно: ~string(){delete []ch;}

Так же, как и для конструктора, не может быть определен указатель на деструктор.

Указатели на объекты

После определения объектов можно определить указатель на объекты.

Пример:

complex A(5.2,2.7);

complex* PA=&A;

Для обращения к общедоступным элементам объекта можно использовать операцию -> или разыменование и операцию точка.

***PA.real() или PA->real;**

Указатель this

Когда функция-член класса вызывается для обработки данных конкретного объекта, этой функции автоматически и неявно передается указатель на тот объект, для которого функция вызвана. Этот указатель имеет имя **this** и неявно определен в каждой функции как **x* this.**

Класс x можно эквивалентным образом описать так:

```
class x {
int m;
public:
int readm() { return this->m; }
};
```

При ссылке на члены использование this излишне. Главным образом this используется при написании функций членов, которые манипулируют непосредственно указателями.

Статические члены класса

В классах возможно определение компонент. которые являются общими для всех объектов данного класса. Такие компоненты должны быть определены в классе, как **статические (static)**. Статические данные классов не дублируются при создании объектов, т.е. каждый статический компонент существует в единственном экземпляре. Доступ к статическому компоненту возможен только после его инициализации.

Например:

int complex : : count = 0;

Это предложение должно быть размещено в глобальной области после определения класса. Только при инициализации статическое данное класса

получает память и становится доступным. Обращаться к статическому данному класса можно обычным образом через имя объекта

complex a; a.count=5;

Но к статическим компонентам можно обращаться и тогда, когда объект класса еще не существует. Доступ к статическим компонентам возможен не только через имя объекта, но и через имя класса

complex:: count=5;

Однако так можно обращаться только к *public* компонентам.

Для обращения к *private* статической компоненте извне можно с помощью **статических компонентов-функций**. Эти функции можно вызвать через имя класса.

Массив объектов как класс

Реализовать массив объектов можно с помощью динамических массивов объектов. Класс "массив объектов" содержит в этом случае указатель на объекты.

```
class StudentsArray
    {
    int size;
    Student *b
    public:
    StudentsArray (int k);//Конструктор
    ~StudentsArray (); // Деструктор
    void Input (); //Вставляет элементы в массив
    void Print (); //Позволяет просмотреть массив.
    int Size(); //Показывает число элементов в массиве.
    int Empty(); //Показывает, есть ли хотя бы один элемент в массиве.
    Student at(int i); //Доступ по индексу с проверкой.
    }
```

Линейный однонаправленный список

Наиболее простой динамической структурой является линейный однонаправленный список объектов.

Элемент списка должен содержат поле связи с последующим элементом.

Пример:

```
class Student
{
char*name;//информационное поле
int age;//информационное поле
public:
point*next;// поле связи
point():name(0),age(0),next(0){}// конструктор
...
};
```

Класс "список объектов" содержит в этом случае указатель на начало списка.

```
class StudentsList
    {
     int size;
     Student *beg;
     public:
     StudentsList (int k);//Конструктор
     ~StudentsList (); // Деструктор
     void Input (); //Вставляет элементы в массив
     void Print (); //Позволяет просмотреть массив.
     int Size(); //Показывает число элементов в массиве.
     int Empty(); //Показывает, есть ли хотя бы один элемент в массиве.
     }
```

II. Постановка задачи.

1. Написать и отладить программу с использованием класса представляющим массив объектов.

a) Определить пользовательский класс в соответствии с вариантом задания с динамическими полями.

b) Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.

c) Создать класс массив объектов. Изменить конструктор и деструктор применительно к динамическому определению пользовательского класса.

d) Просмотреть массив.

2. Написать и отладить программу с использованием класса представляющим список объектов.

a) Определить пользовательский класс в соответствии с вариантом задания с динамическими полями.

b) Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.

c) Создать класс список. Изменить конструктор и деструктор применительно к динамическому определению пользовательского класса.

d) Просмотреть список.

e) Удалить элементы с заданным значением, добавить K элементов в начало списка.

e) Просмотреть список.

3. Написать и отладить программу для работы с классом для шифрования и дешифрования на основе метода указанного в варианте.

a) Определить пользовательский класс в соответствии с вариантом задания.

b) Определить в классе следующие конструкторы: без параметров, с параметрами, копирования. Предусмотреть в конструкторе установку ключей.

c) Определить в классе компоненты-функции для шифровки и дешифровки текста в строчном виде.

d) Реализовать в программе шифрование и дешифрование строки содержащей буквы, цифры и знаки препинания.

III. Варианты

	Класс	Метод шифрования
1	РАБОЧИЙ (имя, номер цеха, разряд, стаж)	С XOR
2	ЖУРНАЛ (имя, периодичность, вид, издательство)	Аффинный
3	ТОВАР (имя, количество, стоимость, страна)	Цезаря
4	КВИТАНЦИЯ (номер, дата, сумма, адресат)	Двухключевой
5	ЦЕХ (имя, начальник, количество рабочих, завод)	Линейный
6	ПЕРСОНА (имя, возраст, пол, национальность)	Квадратичный
7	КОРАБЛЬ (имя, водоизмещение, тип, возраст)	Стандартный
8	ГОРОД (название, республика, область, статус)	Зеркальная
9	КАРТИНА (название, художник, год, галерея)	Четная
10	ПОСТОЯЛЕЦ (имя, страна, возраст, цель поездки)	Двухключевой
11	ТЕЛЕФОН (имя абонента, номер, адрес, тип)	Линейный
12	ФУТБОЛИСТ (имя, возраст, кем играет, голы)	Квадратичный
13	ДИСК (название, объем, цена, страна)	Стандартный
14	УЧЕНИК (имя, возраст, школа, класс)	Зеркальная
15	ДИСЦИПЛИНА (имя, объем часов, курс, вид)	Четная
16	БЛЮДО (название, вид, калорийность, цена)	Фон Неймана
17	МАРШРУТ(имя, количество городов, время, цена)	Подстановки
18	ЭКЗАМЕН (имя студента, дата, предмет, оценка)	Паскаля
19	АДРЕС (имя, улица, номер дома, номер квартиры)	Вижинера
20	СЛУЖАЩИЙ (имя, возраст, отдел, рабочий стаж)	Вернама
21	СТРАНА (имя, правления, правитель, площадь)	Фон Неймана
22	ПРЕПОДАВАТЕЛЬ (имя, кафедра, стаж, нагрузка)	Подстановки
23	ВОЕННЫЙ (имя, звание, род войск, срок службы)	Паскаля
24	ЖИВОТНОЕ (имя, класс, средний вес, возраст)	Вижинера
25	ДЕТАЛЬ (имя, шифр, узел, стоимость)	Вернама

IV. Содержание отчета

1. Титульный лист: название дисциплины; номер и наименование работы; фамилия, имя, отчество студента; дата выполнения.

2. Постановка задачи. Следует дать конкретную постановку, т.е. указать, какой класс должен быть реализован, какие должны быть в нем конструкторы, компоненты-функции и т.д.

3. Определение пользовательского класса с комментариями.

4. Реализация конструкторов и деструктора.

5. Фрагмент программы, показывающий использование указателя на объект и указателя на функцию с объяснением.

6. Листинг основной программы, в котором должно быть указано, в каком месте и какой конструктор или деструктор вызываются.

V.Методические указания

1. Пример класса с статическими компонентами.

```
#include <iostream.h>
class TPoint
{
 double x,y;
 static int N;  // статический компонент – данное : количество точек
  public:

 TPoint(double x1 = 0.0,double y1 = 0.0){N++; x = x1; y = y1;}
 static int& count(){return N;} // статический компонент-функция
};
int TPoint : : N = 0; //инициализация статического компонента-данного
void main(void)
{TPoint A(1.0,2.0);
 TPoint B(4.0,5.0);
 TPoint C(7.0,8.0);
 cout<< \nОпределены "<<TPoint : : count()<<"точки"; }
```

2.Методы массива как класса.

Конструктор:

```
StudentsArray:: StudentsArray (int k)
{ if (k>0) {b=new Students[k]; size=k;}};
```

Деструктор:

```
StudentsArray:: ~StudentsArray ()
{ delete[] Students;};
```

Вставка элементов в массив:

```
void StudentsArray:: Input()
{ for(int i=0;i<=n; i++) b[i].input();};
```

Просмотр массива:

```
void StudentsArray:: Print()
{ for(int i=0;i<=n; i++) b[i].print();};
```

Число элементов в массиве:

```
int StudentsArray:: Size (){return size;};
```

Есть ли хотя бы один элемент в массиве:

```
int StudentsArray:: Empty() {if(n>0) return 1;else return 0;};
```

Доступ по индексу с проверкой:

Например

```
Student StudentsArray:: at(int i) {if(i<size&&i>-1) return b[i]; };
```

3.Методы класса список объектов

Конструктор:

```
StudentsList:: StudentsList (int k)
{
  if ( k>0 )
{
beg = 0; size=k;
for (int i = 0; i < size; i++) { Student*p= new Student(); p->next=beg; beg=p; };
  }
 }
```

Деструктор

```
StudentsList:: ~StudentsList ()
{
  while ( size>0) {
    Student *ptr = beg;
    beg =beg ->next;
    size - -;
    delete ptr;}
 beg = 0; }
```

Ввод элементов списка

```
void StudentsList::Input ()
{ Student*p=beg;
 for (int i = 0; i < size; i++) { p->input (); p=p->next()); cout << endl;} }
```

Просмотр списка

```
void StudentsList::Print ()
{ Student*p=beg;
for (int i = 0; i < size; i++) { p->print(); p=p->next()); cout << endl;} }
```

Лабораторная работа №3

Тема: Наследование

Цель. Получить практические навыки создания иерархии классов с использованием наследования. Получить практические навыки использования дружественных классов. Показать использование абстрактных классов и виртуальных функций.

I. Краткие теоретические сведения

Объекты класса как члены

Если класс содержит объекты других классов, то параметр для конструктора члена указывается в определении (но не в описании) конструктора класса. Конструктор для члена будет вызываться до выполнения тела того конструктора, который задает для него список параметров.

Пример:

```
class A
{
 int i;
public:
 A(int ii) {i=ii;}
...
};
```

Класс содержащий объект класса A:

```
class B
{
 int k;
A member;
public:
 B(int ii, int kk): member(ii){k=kk;}
...
};
```

Аналогично можно задать параметры для конструкторов других членов (если есть еще другие члены):

```
class classdef {
 table members;
 table friends;
 int no_of_members;
 // ...
 classdef(int size);
 ~classdef();
};
```

Списки параметров для членов отделяются друг от друга запятыми (а не двоеточиями), а список инициализаторов для членов можно задавать в произвольном порядке:

```
classdef::classdef(int size)
. friends(size), members(size), no_of_members(size)
{
 // ...
}
```

Конструкторы вызываются в том порядке, в котором они заданы в описании класса.

Определение друзей класса

C++ позволяет друзьям определенного класса обращаться к частным элементам этого класса. Чтобы указать C++, что один класс является другом (friend) другого класса, необходимо просто поместить ключевое слово friend и имя соответствующего класса-друга внутрь определения этого другого класса. Например, приведенный ниже класс book объявляет класс librarian своим другом. Поэтому объекты класса librarian могут напрямую обращаться к частным элементам класса book, используя оператор точку:

```
class book
{
char title [64] ;
char author[64];
char catalog[64];
public:
book (char *, char *, char *);
void show_book(void);
friend librarian;
};
```

Ограничение количества друзей

Если доступ к частным данным другого класса необходим только нескольким функциям класса, C++ позволяет указать, что только определенные функции дружественного класса будут иметь доступ к частным элементам. Предположим, что только функциям change_catalog и get_catalog необходим доступ к частным элементам класса book. Внутри определения класса book можно ограничить доступ к частным элементам только этими двумя функциями, как показано ниже:

```
class book
{
public:
  book(char *, char *, char *);
  void show_book(void);
  friend char *librarian::get_catalog(book);
  friend void librarian: :change_catalog( book *, char *);
private:
```

```
  char title[64];
  char author[ 64 ];
  char catalog[64];
};
```

Наследование

Наследование – это механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса.

Существующие классы называются **базовыми**, а новые – **производными**. Производный класс наследует описание базового класса; затем он может быть изменен добавлением новых членов, изменением существующих функций-членов и изменением прав доступа. С помощью наследования может быть создана иерархия классов, которые совместно используют код и интерфейсы.

Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах.

В иерархии производный объект наследует разрешенные для наследования компоненты всех базовых объектов (*public, protected*).

Допускается множественное наследование – возможность для некоторого класса наследовать компоненты нескольких никак не связанных между собой базовых классов. В иерархии классов соглашение относительно доступности компонентов класса следующее:

private – член класса может использоваться только функциями – членами данного класса и функциями – "друзьями" своего класса. В производном классе он недоступен.

protected – то же, что и ***private,*** но дополнительно член класса с данным атрибутом доступа может использоваться функциями-членами и функциями – "друзьями" классов, производных от данного.

public – член класса может использоваться любой функцией, которая является членом данного или производного класса, а также к ***public*** - членам возможен доступ извне через имя объекта.

Следует иметь в виду, что объявление friend не является атрибутом доступа и не наследуется.

В производном классе унаследованные компоненты получают статус доступа **private**, если новый класс определен с помощью ключевого слова **class**, и статус **public**, если с помощью **struct**.

Явно изменить умалчиваемый статус доступа при наследовании можно с помощью атрибутов доступа – *private, protected и public*, которые указываются непосредственно перед именами базовых классов.

Если класс B определен следующим образом:

```
class B { protected: int t;
public: char u;
};
```

то можно ввести следующие производные классы:

class M: protected B { ... }; // t, и u наследуется как protected

```
class P: public B { ... }; // t - protected, и u- public
class D: private B { ... }; // t, и u наследуется как private
struct F: private B { ... }; // t, и u наследуется как private
struct G: public B { ... }; t - protected, и u - public
```

Конструкторы и деструкторы производных классов.

Поскольку конструкторы не наследуются, при создании производного класса наследуемые им данные-члены должны инициализироваться конструктором базового класса. Конструктор базового класса вызывается автоматически и выполняется до конструктора производного класса. Параметры конструктора базового класса указываются в определении конструктора производного класса. Таким образом происходит передача аргументов от конструктора производного класса конструктору базового класса.

Например.

```
class Basis
{ int a,b;
public:
Basis(int x,int y){a=x;b=y;}
};
class Inherit:public Basis
{int sum;
public:
Inherit(int x,int y, int s):Basis(x,y){sum=s;}
};
```

Объекты класса конструируются снизу вверх: сначала базовый, потом компоненты-объекты (если они имеются), а потом сам производный класс. Таким образом, объект производного класса содержит в качестве подобъекта объект базового класса.

Уничтожаются объекты в обратном порядке: сначала производный, потом его компоненты-объекты, а потом базовый объект.

Таким образом, порядок уничтожения объекта противоположен по отношению к порядку его конструирования.

Абстрактные классы

Абстрактным называется класс, в котором есть хотя бы одна чистая (пустая) виртуальная функция.

Чистой виртуальной функцией называется компонентная функция, которая имеет следующее определение:

virtual тип имя_функции (список_формальных_параметров) = 0;

Чистая виртуальная функция ничего не делает и недоступна для вызовов. Ее назначение – служить основой для подменяющих ее функций в производных классах. Абстрактный класс может использоваться только в качестве базового для производных классов.

Механизм абстрактных классов разработан для представления общих понятий, которые в дальнейшем предполагается конкретизировать. При этом построение иерархии классов выполняется по следующей схеме. Во главе иерархии стоит абстрактный базовый класс. Он используется для наследования интерфейса. Производные классы будут конкретизировать, и реализовать этот интерфейс. В абстрактном классе объявлены чистые виртуальные функции, которые, по сути есть **абстрактные методы**.

Пример.

```
class Base{
public:

Base();                 // конструктор по умолчанию
Base(const Base&);  // конструктор копирования
virtual ~Base();    // виртуальный деструктор
virtual void Show()=0;    // чистая виртуальная функция
// другие чистые виртуальные функции
protected: // защищенные члены класса
private:
// часто остается пустым, иначе будет мешать будущим разработкам
};
class Derived: virtual public Base{
public:
Derived();                      // конструктор по умолчанию
Derived(const Derived&);     // конструктор копирования
Derived(параметры);          // конструктор с параметрами
virtual ~Derived();             // виртуальный деструктор
void Show();                    // переопределенная виртуальная функция
// другие переопределенные виртуальные функции
// другие перегруженные операции
protected:
// используется вместо private, если ожидается наследование
private:
// используется для деталей реализации
};
```

II. Постановка задачи.

1. Определить иерархию классов с использованием абстрактных классов для фигуры с вырезом. Создать класс для основной фигуры и для выреза. Реализовать методы для вычисления площади и периметра фигуры. Путем наследования создать класс для фигуры с вырезом. Реализовать методы для вычисления площади и периметра фигуры с вырезом.

2. Реализовать класс. Определить в классе дружественный класс со статическими методами. Создать массив объектов. С помощью статических методов дружественного класса вывести следующее:

Вывести все объекты, у которых значение критерия превышает заданное значение.

Найти объект с указанным значением критерия.

Подсчитать количество объектов, у которых значение критерия не превышает заданное значение.

3. Создать класс для трехмерной фигуры с использованием абстрактных классов (в соответствии с вариантом).

Реализовать в классе методы вычисления поверхности и объема.

Путем наследования добавить метод определения лежит заданная точка внутри фигуры, на границе или вне фигуры.

Применить указанные методы в программе.

III. Варианты заданий.

1.1 Квадрат с прямоугольным вырезом
1.2 инженер – стаж;
1.3 куб

2.1 Равносторонний треугольник с круговым вырезом
2.2 преподаватель- нагрузка;
2.3 шар

3.1 Круг с прямоугольным вырезом
3.2 рабочий – разряд;
3.3 равносторонняя пирамида

4.1 Эллипс квадратным вырезом;
4.2 деталь – количество;
4.3 цилиндр

5.1 Прямоугольник с круговым вырезом
5.2 завод – количество рабочих;
5.3 конус

6.1 Произвольный треугольник с круговым вырезом;
6.2 учебник – тираж;
6.3 параллелепипед

7.1 Прямоугольник с квадратным вырезом;
7.2 тест – количество правильных ответов;
7.3 призма основание прямоугольный треугольник

8.1 Прямоугольный треугольник с прямоугольным вырезом;
8.2 мегаполис – количество жителей;
8.3 призма основание равносторонний треугольник

9.1 Параллелограмм с эллиптическим вырезом;
9.2 игрушка – цена;
9.3 призма основание равнобедренный треугольник

10.1 Эллипс круговым вырезом;
10.2 чек- сумма;
10.3 призма основание параллелограмм

11.1 Круг с вырезом в виде центрального сектора;
11.2 экспресс- максимальная скорость
11.3 призма основание ромб

12.1 Ромб с эллиптическим вырезом;
12.2 дизель – мощность;
12.3 пирамида основание равнобедренный треугольник

13.1 Ромб с круговым вырезом;
13.2 республика – год основания;
13.3 призма основание произвольный треугольник

14.1 Правильная трапеция с круговым вырезом;
14.2 птица – дальность полёта;
14.3 призма основание ромб

15.1 Правильная трапеция с круговым вырезом;
15.2 корвет – скорость;
15.3 пирамида основание равнобедренный треугольник

16.1 Правильный многоугольник ромбовидным и с круговым вырезами;
16.2 вратарь – количество пропущенных голов
16.3 усеченная равносторонняя пирамида

17.1 Правильный многоугольник с треугольным и эллиптическим вырезами;
17.2 генерал- срок службы;
17.3 усеченный конус

18.1 Прямоугольник с прямоугольным и треугольным вырезами
18.2 кинофильм - кассовый сбор
18.3 усеченная пирамида основание правильный многоугольник

19.1 Прямоугольник с круговым и эллиптическим вырезами;
19.2 антибиотик – срок годности;
19.3 призма основание произвольный многоугольник

20.1 Круг с круговым и квадратным вырезами;
20.2 университет – количество студентов
20.3 усеченная пирамида основание равносторонний треугольник

21.1 Круг с квадратным и ромбовидным вырезами;
21.2 скульптор – количество произведений;
21.3 усеченная пирамида основание равнобедренный треугольник

22.1 Круг с эллиптическим и треугольным вырезами
22.2 клиника – количество коек
22.3 усеченная пирамида основание произвольный треугольник

23.1 Прямоугольник с треугольным и круговым вырезами;
23.2 винчестер – объем памяти
23.3 усеченная пирамида основание правильный пятиугольник

24.1 Эллипс треугольными и эллиптическими вырезами;
24.2 сотовый телефон – абонентская плата
24.3 усеченная пирамида основание правильный шестиугольник

25.1 Эллипс круговым и прямоугольным вырезами;
25.2 персональный компьютер – скорость процессора
25.3 призма основание правильный многоугольник

IV. Содержание отчета

1. Титульный лист: название дисциплины; номер и наименование работы; фамилия, имя, отчество студента; дата выполнения.
2. Постановка задачи. Следует дать конкретную постановку, т.е. указать, какие классы должны быть реализованы, какие должны быть в них конструкторы, компоненты-функции и т.д.
3. Иерархия классов в виде графа.
4. Определение пользовательских классов с комментариями.
5. Реализация конструкторов с параметрами и деструктора.
6. Листинг демонстрационной программы.

V. Методические указания

1. Геометрические фигуры должны быть созданы через координаты точек. Например шар должен быть задан через координаты центра и радиус шара.

2. Необходимо использовать объект структуры или класса Point (Точка).

```
Class Point
{ double x, y, z;
  Public:
  Point(x=x1; y1=0; z1=0) { x=x1; y=y1; z=z1;}
...
}
```

3. В классах для фигур с вырезом в конструкторе предусмотреть проверку корректности данных.

4. Примеры иерархии классов:

TObject (абстр. класс)

Геометрическая фигура (абстр. класс)

Трехмерная фигура(абстр. класс)

Шар

5. Пример определения добавленного абстрактного класса

```
class TObject
{
public:
virtual void Print()=0;
};
```

6. Абстрактный класс для геометрической фигуры

```
class TGraph:TObject
{
public:
virtual void Draw ()=0;
void Move ();
void Rotate();
};
```

Лабораторная работа №4

Тема: Перегрузка операций и шаблоны классов.

Цель: Получить практические навыки перегрузки операций в языке C++. Получить практические навыки создания шаблонов и использования их в программах C++.

I. Краткие теоретические сведения

Перегрузка операций

В языке C++ для перегрузки операций используется ключевое слово operator, с помощью которого определяется специальная операция-функция (operator function).

Формат операции-функции:

тип_возвр_значения operator знак_операции (специф_параметров)
{операторы_тела_функции}

Перегрузка унарных операций

- Любая унарная операция ⊕ может быть определена двумя способами: либо как компонентная функция без параметров, либо как глобальная (возможно дружественная) функция с одним параметром. В первом случае выражение ⊕ Z означает вызов Z.operator ⊕ (), во втором – вызов operator ⊕(Z).
- Унарные операции, перегружаемые в рамках определенного класса, могут перегружаться только через нестатическую компонентную функцию без параметров. Вызываемый объект класса автоматически воспринимается как операнд.
- Унарные операции, перегружаемые вне области класса (как глобальные функции), должны иметь один параметр типа класса. Передаваемый через этот параметр объект воспринимается как операнд.

Синтаксис:

а) в первом случае (описание в области класса):

тип_возвр_значения operator знак_операции

б) во втором случае (описание вне области класса):

тип_возвр_значения operator знак_операции(идентификатор_типа)

Примеры.

```
1) class person
{ int age;
 ...
public:
 ...
 void operator++(){ ++age;}
 };
```

```
2) class person
{ int age;
...
  public:
...
friend void operator++(person&);
};
```

```
void main()
{class person jon;
 ++jon;}
```

```
void person::operator++(person& ob)
  {++ob.age;}
  void main()
  {class person jon;
  ++jon;}
```

Перегрузка бинарных операций

- Любая бинарная операция ⊕ может быть определена двумя способами: либо как компонентная функция с одним параметром, либо как глобальная (возможно дружественная) функция с двумя параметрами. В первом случае **x⊕y** означает вызов **x.operator⊕(y)**, во втором – вызов **operator ⊕(x,y).**

- Операции, перегружаемые внутри класса, могут перегружаться только нестатическими компонентными функциями с параметрами. Вызываемый объект класса автоматически воспринимается в качестве первого операнда.
- Операции, перегружаемые вне области класса, должны иметь два операнда, один из которых должен иметь тип класса.

Пример.

```
1) class person{...};
class adresbook
{ // содержит в качестве компонентных данных множество объектов типа
  //person, представляемых как динамический массив, список или дерево
   ...
public:
  person&  operator[](int);  //доступ к i-му объекту
};
person& adresbook : : operator[](int i){. . .}
void main()
{class adresbook persons;
 class person record;
 ...
 record = persons [3];
}
```

Перегрузка операции присваивания

Операция отличается тремя особенностями:

- операция не наследуется;
- операция определена по умолчанию для каждого класса в качестве операции поразрядного копирования объекта, стоящего справа от знака операции, в объект, стоящий слева.
- операция может перегружаться только в области определения класса. Это гарантирует, что первым операндом всегда будет леводопустимое выражение.

Формат перегруженной операции присваивания:

имя_класса& *operator*=(**имя_класса&**);

Отметим две важные особенности функции *operanor*=. Во-первых, в ней используется параметр-ссылка. Это необходимо для предотвращения создания копии объекта, передаваемого через параметр по значению. В случаи создания копии. она удаляется вызовом деструктора при завершении работы функции. Но деструктор освобождает распределенную память, еще необходимую объекту, который является аргументом. Параметр-ссылка помогает решить эту проблему.

Во-вторых, функция operator=() возвращает не объект, а ссылку на него. Смысл этого тот же, что и при использовании параметра-ссылки.

Шаблоны классов.

Шаблон класса (иначе параметризованный класс) используется для построения родового класса. Подобно тому, как класс определяет правила построения и формат отдельных объектов, шаблон класса определяет способ построения отдельных классов. В определении класса, входящего в шаблон, имя класса является не именем отдельного класса, а параметризованным именем семейства классов.

Общая форма объявления параметризованного класса:

template <class тип_данных> class имя_класса { . . . };

Основные свойства шаблонов классов

* Компонентные функции параметризованного класса автоматически являются параметризованными. Их не обязательно объявлять как параметризованные с помощью *template*.

* Дружественные функции, которые описываются в параметризованном классе, не являются автоматически параметризованными функциями, т.е. по умолчанию такие функции являются дружественными для всех классов, которые организуются по данному шаблону.

* Если *friend*-функция содержит в своем описании параметр типа параметризованного класса, то для каждого созданного по данному шаблону класса имеется собственная *friend*-функция.

* В рамках параметризованного класса нельзя определить *friend*-шаблоны (дружественные параметризованные классы).

* С одной стороны, шаблоны могут быть производными (наследоваться) как от шаблонов, так и от обычных классов, с другой стороны, они могут использоваться в качестве базовых для других шаблонов или классов.

* Шаблоны функций, которые являются членами классов, нельзя описывать как *virtual*.

* Локальные классы не могут содержать шаблоны в качестве своих элементов.

Компонентные функции параметризованных классов

Реализация компонентной функции шаблона класса, которая находится вне определения шаблона класса, должна включать дополнительно следующие два элемента:

*Определение должно начинаться с ключевого слова *template*, за которым следует такой же *список_параметров_типов* в угловых скобках, какой указан в определении шаблона класса.

*За *именем_класса*, предшествующим операции области видимости (::), должен следовать *список_имен_параметров* шаблона.

*template<список_типов>тип_возвр_значения имя_класса<список_* имен_ параметров> : : имя_функции(список_параметров){ . . . }

Шаблон массива

Реализация шаблона класса array для работы с объектами классов с перегруженными операциями.

```
    template<class T > class array
    {
  T *data;
  int size;
  int index;
public:
  array(int size);
  int push_back (T);
  int size() { return index; }
  ~array () { delete []data; }
  T sum(void);
  T avg();
  void show_array ();
  };
    template<class T > array<T >::array(int size)
    {   data = new T[size];
  if (data == NULL)
      {
   cerr << "Недостаточно памяти - программа завершается" << endl;
   exit(l);
  }
      array::size = size;
       array::index = 0;
}
    template<class T > int array<T >::push_back (T value)
    {
 if (index == size)
 return(-1); // Массив полон
 else
      {
   data[index] = value;
   index++;
```

```
    return(0); // Успешно
  }
}
    template<class T > T array<T >::sum()
    {   T sum = 0;
  for (int i = 0; i < index; i++) sum += data[i];
  return(sum);
}
    template<class T > T array<T >::avg()
    {   T sum=0;
  for (int i = 0; i < index; i++) sum += data[i];
  return (sum / index);
}
    template<class T > void array<T >:: show_array()
    {
  for (int i = 0; i < index; i++) cout << data[i] << ' ';
  cout << endl;
}
```

II. Постановка задачи

1. Выбрать класс в соответствии с вариантом. Дополнить определение класса заданными перегруженными операциями в соответствии с вариантом. Реализовать эти операции. Выполнить тестирование.

2. Создать шаблон группы объектов. Определить и реализовать методы для вычисления суммы, среднего значения, максимального и минимального значения, а также сортировки. Реализовать эти операции для объекта, выбранного класса. Выполнить тестирование.

3. Выполнить то же самое для шаблона список.

III. Варианты

1. Класс – интервал даты (год, месяц, день). Дополнительно перегрузить следующие операции:

+,-,*,\ – арифметические операции;

==,!=,<,>,<=,>= - сравнение интервалов дат;

2. Класс – интервал времени (час, минута, секунда). Дополнительно перегрузить следующие операции:

+,-,*,\ – арифметические операции;

==,!=,<,>,<=,>= - сравнение интервалов времени;

3. Класс – деньги (величина, единица, курс по отношению к доллару). Дополнительно перегрузить следующие операции:

+,-,*,\ – арифметические операции;

==,!=,<,>,<=,>= - сравнение денежных знаков по курсу;

4. Класс – угол (градус, минута, секунда). Дополнительно перегрузить следующие операции:

+,-,*,\ – арифметические операции;

==,!=,<,>,<=,>= - сравнение углов;

5. Память компьютера (мегабайт, килобайт, байт). Дополнительно перегрузить следующие операции:

+,-,*,\ – арифметические операции;

==,!=,<,>,<=,>= - сравнение памяти;

6. Класс – комплексное число в алгебраической форме. Дополнительно перегрузить следующие операции:

+,-,*,\ – бинарные арифметические операции;

++, -- - унарные арифметические операции;

==,!=,<,>,<=,>= - сравнение комплексных чисел по их модулю;

7. Класс – комплексное число в тригонометрической форме. Дополнительно перегрузить следующие операции:

+,-,*,\ – бинарные арифметические операции;

++, -- - унарные арифметические операции;

==,!=,<,>,<=,>= - сравнение комплексных чисел по их модулю;

8. Класс – комплексное число в экспоненциальной форме. Дополнительно перегрузить следующие операции:

+,-,*,\ – бинарные арифметические операции;

++, -- - унарные арифметические операции;

==,!=,<,>,<=,>= - сравнение комплексных чисел по их модулю;

9. Класс – рациональное число-дробь. Дополнительно перегрузить следующие операции:

+,-,*,\ – бинарные арифметические операции;

++, -- - унарные арифметические операции;

==,!=,<,>,<=,>= - сравнение рациональных чисел;

10. Класс – десятичное число с фиксированной точкой. Дополнительно перегрузить следующие операции:

+,-,*,\ – бинарные арифметические операции;

++, -- - унарные арифметические операции;

==,!=,<,>,<=,>= - сравнение десятичных чисел;

11. Класс – интервал вещественных чисел [a,b]. Дополнительно перегрузить следующие операции:

+ – сложение ([a,b]+ [c,d]=[a+c,b+d]),

- – вычитание ([a,b] - [c,d]=[a-c,b-d]),

* – умножение ([a,b] * [c,d]=[min(a*c,a*d,b*c,b*d), max(a*c,a*d,b*c,b*d)]),

/ – деление ([a,b] / [c,d]=[a,b]*[1/c,1/d], при условии c>0 или d<0)

==,!=,<,>,<=,>= - сравнение интервалов ([a,b]==[c,d] истинно если a=c и b=d, [a,b]<=[c,d] истинно если a<=c и b<=d);

12. Класс – вычет по простому модулю p. Дополнительно перегрузить следующие операции:

+,-,*,\ – арифметические операции над вычетами;

==,!=,<,>,<=,>= - сравнение вычетов;

13. Класс – квадратная вещественная матрица размером 2x2. Дополнительно перегрузить следующие операции:

+,-,* – матричные операции;

++ – транспонирование матрицы;

==,!=,<,>,<=,>= - сравнение матриц по модулю;

14. Класс – конденсатор. (C – ёмкость, Umax – максимальное допустимое напряжение). Дополнительно перегрузить следующие операции:

+,* – последовательное и параллельное соединение;

-,\ – обратные операции;

==,!=,<,>,<=,>= - сравнение по максимально допустимому заряду;

15. Класс проводник. (R – сопротивление, Umax – максимальное допустимое напряжение). Дополнительно перегрузить следующие операции:

+,* – последовательное и параллельное соединение;

-,\ – обратные операции;

==,!=,<,>,<=,>= - сравнение по максимальной допустимой мощности;

16. Класс – n мерный вектор. Дополнительно перегрузить следующие операции:

+,- – векторное сложение и вычитание;

* – скалярное умножение векторов;

++,-- – увеличение и уменьшение размерности вектора;

==,!=,<,>,<=,>= - сравнение векторов по модулю;

[] – доступ по индексу;

size () – размерность вектора;

17. Класс – n мерный битовый вектор. Дополнительно перегрузить следующие операции:

&,|,^ – побитовые бинарные операции;

! – побитовое отрицание;

++,-- – увеличение и уменьшение размерности вектора;
==,!=,<,>,<=,>= - лексикографическое сравнение;

18. Класс – трехмерный вектор в полярных координатах. Дополнительно перегрузить следующие операции:

+,- – векторное сложение и вычитание;
* – скалярное умножение векторов;
==,!=,<,>,<=,>= - сравнение векторов по модулю;

19. Класс – множество строк. Дополнительно перегрузить следующие операции:

+,-,*,\ – теоретико – множественные операции;
() – проверка принадлежности строки множеству;
==,!=,<,>,<=,>= - сравнение языков по мощности;

20. Класс – событие – подмножество чисел от 1 до n. (Целочисленный массив от 1 до n. При этом a[i]=1 если i входит в событие иначе a[i]=0). Дополнительно перегрузить следующие операции:

+,-,*,\ – теоретико – множественные операции над событиями;
() – проверка принадлежности числа событию;
p() – вычисление вероятности события (число элементов деленное на n).
==,!=,<,>,<=,>= - сравнение событий по вероятности;

21. Класс – множество вещественных чисел. Дополнительно перегрузить следующие операции:

+,-,*,\ – теоретико – множественные операции;
() – проверка принадлежности числа множеству;
==,!=,<,>,<=,>= - сравнение множеств по мощности;

22. Класс – произвольный массив слов:

+ – поэлементная конкатенация;
- – поэлементное удаление подстроки;
==,!=,<,>,<=,>= - лексикографическое сравнение;

19. Класс – конечный язык. Дополнительно перегрузить следующие операции:

+ – объединение языков;
* - конкатенация языков;
^ - возведение в целочисленную степень;
() – проверка принадлежности слова языку;
==,!=,<,>,<=,>= - сравнение языков по мощности;

24. Класс – произвольная битовая матрица. Дополнительно перегрузить следующие операции:

+,-,*,\ – матричные побитовые операции;

++ – транспонирование матрицы.

==,!=,<,>,<=,>= - сравнение матриц по модулю;

25. Класс – произвольная вещественная матрица размером n x m. Дополнительно перегрузить следующие операции:

+,-,*,\ – матричные операции;

++ – транспонирование матрицы;

* – умножение матрицы на число;

* – умножение числа на матрицу;

==,!=,<,>,<=,>= - сравнение матриц по модулю (значению определителя);

IV. Содержание отчета

1. Титульный лист: название дисциплины; номер и наименование работы; фамилия, имя, отчество студента; дата выполнения.
2. Конкретное задание с указанием номера варианта, реализуемого класса и операций.
3. Определение класса.
4. Обоснование включения в класс нескольких конструкторов, деструктора и операции присваивания.
5. Объяснить выбранное представление памяти для объектов реализуемого класса.
6. Реализация перегруженных операций с обоснованием выбранного способа (функция – член класса, внешняя функция, внешняя дружественная функция).
7. Тестовые данные и результаты тестирования.

V. Методические указания

1. Использование шаблона класса array для создания класса, работающего со значениями типа int..

```
#include <iostream.h>
#include "array.h"
void main()
{
// Массив из 100 элементов
array<long> numbers(100);
// Массив из 200 элементов
array<float > values(200);
int i;
for (i = 0; i < 50; i++) numbers.push_back(i);
```

```
  numbers.show_array();
  cout << "Сумма чисел равна " << numbers.sum () << endl;
  cout << "Среднее значение равно " << numbers.avg () << endl;
  for (i = 0; i < 100; i++) values.push_back (i * 100);
  values.show_array();
  cout << "Сумма чисел равна." << values.sum() << endl;
  cout << "Среднее значение равно " << values.avg() << endl;
}
```

2. Если объект в списке не имеет поле связи, такую связь надо предусмотреть в отдельном шаблоне класса:

```
template <class T>
  class list_item
  {
    T  value;
    list_item * next;
  public:
    list_item( T value, list_item *item = 0 ) : value( value ) {
      if ( !item )
        next = 0;
      else {
        next = item->next;
        item-> next = this;
      }
    }
    T value() { return value; }
    list_item* next() { return  next; }
    void next( list_item *link ) { next = link; }
    void value( T new_value ) { value = new_value; }
    };
```

Класс список теперь может быть определен следующим образом:

```
template <class T>
class slist
{
  list_item<T> *  at_front;
  list_item<T> *  at_end;
  int  size;
public:
list()  : at_front( 0 ), at_end( 0 ),  size( 0 ) {};
  list( const list& );
  list& operator=( const list& );
  ~list() { remove_all(); }
  void push_back( T value );
  int size() { return _size; }
  void remove_all();
```

```
    T sum(void);
    T max ();
    void show_list ();
  };
  void slist:: remove_all()
  {
    while ( size>0) {
      ilist_item *ptr = _at_front;
      at_front = at_front->next();
      size - -;
      delete ptr;}
   at_front = at_end = 0;
    }

  void slist:: push_back (T  value )
  {
    if ( size==0 )
      at_end = at_front = new list_item( value );
    else  at_end = new list_item(value, at_end );
   size++;
  };
    template<class T > T array<T >::sum()
    {
 T sum(); T*p=at_front;
 for (int i = 0; i < size; i++) { sum += p->value(); p=p->next();
 return(sum);
}
    template<class T > T array<T >::max ()
    {
 T s=at_front->value(); T*p=at_front;
 for (int i = 0; i < size; i++) if (p->value()>s) s=p->value();
 return (s);
}
    template<class T > void array<T >::show_slist ()
    { T*p=at_front;
 for (int i = 0; i < size; i++) { T s=p->value(); s.print(); p=p->next()};
 cout << endl;
}
```

Лабораторная работа №5

Тема: Потоковые классы и исключительные ситуации.

Цель: Научиться программировать ввод и вывод в C++, используя объекты потоковых классов стандартной библиотеки C++. Научиться управлять исключительными ситуациями.

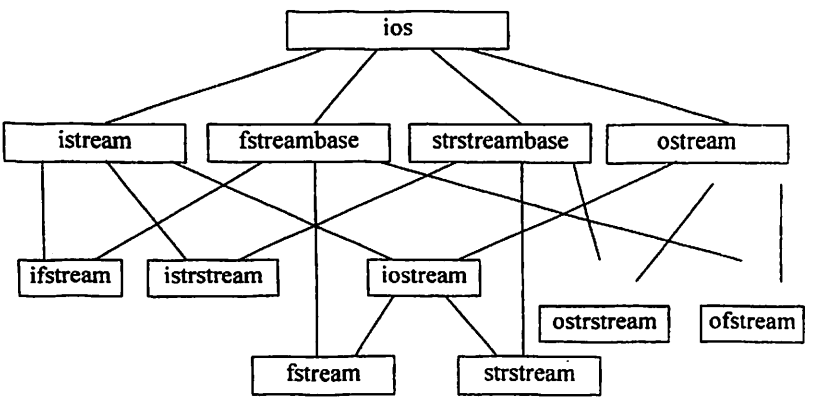
I. Краткие теоретические сведения

Потоковые классы в C++.

Библиотека потоковых классов C++ построена на основе двух базовых классов: **ios** и **streambuf** Класс streambuf обеспечивает организацию и взаимосвязь буферов ввода-вывода, размещаемых в памяти, с физическими устройствами ввода-вывода. Методы и данные класса streambuf программист явно обычно не использует. Этот класс нужен другим классам библиотеки ввода-вывода. Он доступен и программисту для создания новых классов на основе уже существующих.

Класс ios содержит средства для форматированного ввода-вывода и проверки ошибок.

Схема нерархии



istream – класс входных потоков;
ostream – класс выходных потоков;
iostream – класс ввода-вывода;
istrstream – класс входных строковых потоков;
ifstream – класс входных файловых потоков и т.д.

Потоковые классы, их методы и данные становятся доступными в программе, если в неё включен нужный заголовочный файл.

iostream.h – для ios, ostream, istream.
strstream.h – для strstream, istrstream, ostrstream
fstream.h – для fstream, ifstream, ofstream

Базовые потоки ввода-вывода

Для ввода с потока используются объекты класса istream, для вывода в поток – объекты класса ostream.

В классе istream определены следующие функции:

- istream& get(char* buffer,int size,char delimiter='\n');

Эта функция извлекает символы из istream и копирует их в буфер. Операция прекращается при достижении конца файла, либо при скопировании size символов, либо при обнаружении указанного разделителя. Сам разделитель не копируется и остается в streambuf. Последовательность прочитанных символов всегда завершается нулевым символом.

- istream& read(char* buffer,int size);

Не поддерживает разделителей, и считанные в буфер символы не завершаются нулевым символом.

- istream& getline(char* buffer,int size, char delimiter='\n');

Разделитель извлекается из потока, но в буфер не заносится. Это основная функция для извлечения строк из потока. Считанные символы завершаются нулевым символом.

- istream& get(streambuf& s,char delimiter='\n');

Копирует данные из istream в streambuf до тех пор, пока не обнаружит конец файла или символ-разделитель, который не извлекается из istream. В s нулевой символ не записывается.

- istream get (char& C);

Читает символ из istream в C. В случае ошибки C принимает значение 0XFF.

- int get();

Извлекает из istream очередной символ. При обнаружении конца файла возвращает EOF.

- int peck();

Возвращает очередной символ из istream, не извлекая его из istream.

- int gcount();

Возвращает количество символов, считанных во время последней операции неформатированного ввода.

- istream& putback(C)

Если в области get объекта streambuf есть свободное пространство, то туда помещается символ C.

- istream& ignore(int count=1,int target=EOF);

Извлекает символ из istream, пока не произойдет следующее:

– функция не извлечет count символов;

– не будет обнаружен символ target;

– не будет достигнуто конца файла.

В классе ostream определены следующие функции:

• ostream& put(char C);

Помещает в ostream символ C.

• ostream& write(const char* buffer,int size);

Записывает в ostream содержимое буфера. Символы копируются до тех пор, пока не возникнет ошибка или не будет скопировано size символов. Буфер записывается без форматирования. Обработка нулевых символов ничем не отличается от обработки других. Данная функция осуществляет передачу необработанных данных (бинарных или текстовых) в ostream.

• ostream& flush();

Сбрасывает буфер streambuf.

Для прямого доступа используются следующие функции установки позиции чтения - записи.

При чтении

• istream& seekg(long p);

Устанавливает указатель потока get (не путать с функцией) со смещением p от начала потока.

• istream& seekg(long p,seek_dir point);

Указывается начальная точка перемещения.

enum seek_dir{beg,curr,end}

Положительное значение p перемещает указатель get вперед (к концу потока), отрицательное значение p – назад (к началу потока).

• long tellg();

Возвращает текущее положение указателя get.

При записи

•ostream& seekp(long p);

Перемещает указатель put в streambuf на позицию p от начала буфера streambuf.

• ostream& seekp(long p,seek_dir point);

Указывает точка отсчета.

• long tellp();

Возвращает текущее положение указателя put.

Помимо этих функций в классе istream перегружена операция >>, а в классе ostream <<. Операции << и >> имеют два операнда Левым операндом является объект класса istream (ostream), а правым – данное, тип которого задан в языке.

Форматирование

Непосредственное применение операций ввода << и вывода >> к стандартным потокам cout, cin, cerr, clog для данных базовых типов приводит к использованию "умалчиваемых" форматов внешнего представления пересылаемых значений.

Форматы представления выводимой информации и правила восприятия данных при вводе могут быть изменены программистом с помощью флагов

форматирования. Эти флаги унаследованы всеми потоками из базового класса ios. Флаги форматирования реализованы в виде отдельных фиксированных битов и хранятся в protected компоненте класса long x_flags. Для доступа к ним имеются соответствующие public функции.

Кроме флагов форматирования используются следующие protected компонентные данные класса ios:

int x_width – минимальная ширина поля вывода.

int x_precision – точность представления вещественных чисел (количество цифр дробной части) при выводе;

int x_fill – символ-заполнитель при выводе, пробел – по умолчанию.

Для получения (установки) значений этих полей используются следующие компонентные функции:

```
int width();
int width(int);
int precision();
int precision(int);
char fill();
char fill(char);
```

Манипуляторы

Манипуляторами называются специальные функции, позволяющие модифицировать работу потока. Особенность манипуляторов состоит в том, что их можно использовать в качестве правого операнда операции >> или <<. В качестве левого операнда, как обычно, используется поток (ссылка на поток), и именно на этот поток воздействует манипулятор.

```
// для этих манипуляторов требуется #include <ionamip>
setfill( ch)        Заполнять пустое место символом ch
setprecision( n )   Установить точность вывода числа с плавающей точкой
равной n
setw( w )           Установить ширину поля ввода или вывода равной w
setbase( b )        Выводить целые числа по основанию b
* обозначает состояние потока по умолчанию
```

Состояние потока

Каждый поток имеет связанное с ним состояние. Состояния потока описываются в классе ios в виде перечисления enum.

```
public:
enum io_state{
goodbit, //нет ошибки 0X00
eofbit,  //конец файла 0X01
failbit,  //последняя операция не выполнилась 0X02
badbit, //попытка использования недопустимой операции 0X04
hardfail//фатальная ошибка 0X08
};
```

Флаги, определяющие результат последней операции с объектом ios. содержатся в переменной state. Получить значение этой переменной можно с помощью функции int rdstate().

Кроме того, проверить состояние потока можно следующими функциями:

int bad(); 1, если badbit или hardfail
int eof(); 1, если eofbit
int fail(); 1, если failbit, badbit или hardfail
int good(); 1, если goodbit

Если операция >> используется для новых типов данных, то при её перегрузке необходимо предусмотреть соответствующие проверки.

Файловый ввод-вывод

Для открытия файла в *режиме добавления* необходимо при его открытии указать второй параметр, как показано ниже:

fstream output_file("FILENAME.EXT", ios::app);

В данном случае параметр *ios::app* указывает *режим открытия файла.*

Значения режимов открытия.

Режим открытия	Назначение
ios::app	Открывает файл в режиме добавления, располагая файловый указатель в конце файла.
ios::ate	Располагает файловый указатель в конце файла. Чтение больше не допустимо, выводные данные записываются в конец файла
ios::in	Указывает открыть файл для ввода.
ios::nocreate	Если указанный файл не существует, не создавать файл и возвратить ошибку.
ios::noreplace	Если файл существует, операция открытия должна быть прервана и должна возвратить ошибку.
ios::out	Указывает открыть файл для вывода.
ios::trunc	Сбрасывает (перезаписывает) содержимое, существующего файла.

Следующая операция открытия файла открывает файл для вывода, используя режим *ios::noreplace,* чтобы предотвратить перезапись существующего файла:

ifstream output_file("FIlename.EXT", ios::out | ios::noreplace);

Если при создании потока он не присоединен к файлу, то присоединить существующий поток к файлу можно функцией

void open(const char* name,int mode,int p=filebuf::openprot);

Функция

void fstreambase::close();

сбрасывает буфер потока, отсоединяет поток от файла и закрывает файл. Эту функцию необходимо явно вызвать при изменении режима работы с потоком. Автоматически она вызывается только при завершении программы.

Таким образом, создать поток и связать его с файлом можно следующими способами:

1. Создается объект fstream (ifstream, ofstream)

fstream stream;

Открывается файл и связывается с потоком

stream.open("имя",ios::in);

2. Создается объект fstream, одновременно открывается файл, который связывается с потоком

fstream stream("имя",ios::in);

Исключительные ситуации

Исключительная ситуация (exception) представляет собой неожиданное событие — ошибку — в программе. В программах каждая исключительная ситуация определяется как класс. Например, следующие ситуации определяют три исключительные ситуации для работы с файлами:

```
class file_open_error {};
class file_read_error {};
class file_write_error {};
```

Исключительные ситуации могут использовать переменные и функции-элементы класса. Каждая исключительная ситуация соответствует классу.

Проверка исключительной ситуации

Прежде чем программы могут обнаружить и отреагировать на исключительную ситуацию, необходимо использовать оператор C++ try для разрешения обнаружения исключительной ситуации. Например, следующий оператор try разрешает обнаружение исключительной ситуации для вызова функции file_copy:

```
try
{
 file_copy("SOURCE.TXT", "TARGET.TXT");
};
```

Сразу же за оператором try программа должна разместить один или несколько операторов catch, чтобы определить, какая исключительная ситуация имела место (если она вообще была):

```
catch (file_open_error)
{
 cerr << "Ошибка открытия исходного или целевого файла" << endl;
 exit(1);
}
```

В данном случае независимо от типа ошибки код просто выводит сообщение и завершает программу. Если вызов функции прошел успешно и исключительная ситуация не выявлена, C++ просто игнорирует операторы catch.

Генерация исключительной ситуации

Сам C++ не генерирует исключительные ситуации. Их генерируют программы, используя оператор C++ throw. Например, при открытии файла программа может проверить условие возникновения ошибки и сгенерировать исключительную ситуацию **throw file_open_error()**.

При использовании исключительных ситуаций, программа проверяет условие возникновения ошибки и, если необходимо, генерирует исключительную ситуацию, используя оператор throw. Когда C++ встречает оператор throw, он активизирует соответствующий обработчик исключительной ситуации (функцию, чьи операторы определены в классе исключительной ситуации). После завершения функции обработки исключительной ситуации C++ возвращает управление первому оператору, который следует за оператором try, разрешившим обнаружение исключительной ситуации. Далее, используя операторы catch, программа может определить, какая именно исключительная ситуация возникла, и отреагировать соответствующим образом.

Использование элементов данных исключительной ситуации

В предыдущих примерах программы, используя оператор catch, могли определить, какая именно исключительная ситуация имела место, и отреагировать соответствующим образом. Например, в случае с исключительной ситуацией file_open_error программе необходимо знать имя файла, который вызвал ошибку. Чтобы сохранить подобную информацию об исключительной ситуации, программы могут просто добавить элементы данных в класс исключительной ситуации. Если в дальнейшем программа сгенерирует исключительную ситуацию, она передаст эту информацию функции обработки исключительной ситуации в качестве параметра, как показано ниже:

```
    throw file_open_error(source);
throw file_read_error(344);
```

В обработчике исключительной ситуации эти параметры могут быть присвоены соответствующим переменным класса (очень похоже на конструктор). Например, следующие операторы изменяют исключительную ситуацию file_open_error, чтобы присвоить имя файла, который вызвал ошибку, соответствующей переменной класса:

```
    class file_open_error
    {
public:
  file_open_error(char *filename) { strcpy(file_open_error::filename, filename);
}
```

```
  char filename[255] ;
};
```

Обработка неожиданных исключительных ситуаций

По умолчанию если программа генерирует исключительную ситуацию, которая не улавливается (программа не имеет соответствующего обработчика исключительной ситуации), то запустится стандартный обработчик, предоставляемый языком C++. В большинстве случаев стандартный обработчик завершит программу.

Исключительные ситуации и классы

При создании класса можно определить исключительные ситуации, характерные для данного класса. Чтобы создать исключительную ситуацию, характерную для конкретного класса, необходимо включить эту исключительную ситуацию в качестве одного из общих (public) элементов класса. Например, следующее описание класса string определяет две исключительные ситуации:

```
    class string
    {
public:
  string(char *str);
  void fill_string(*str);
  void show_string(void);
  int string_length(void);
  class string_empty { } ;
  class string_overflow {};
private:
  int length;
  char string[255];
};
```

Как видно, этот класс определяет исключительные ситуации string_empty и string_overflow. В программе можно проверить наличие исключительной ситуации, используя оператор глобального разрешения и имя класса, как показано ниже:

```
    try
    {
  some_string.fill_string(some_long_string);
};
catch (string::string_overflow)
    {
  cerr << "Превышена длина строки, символы отброшены" << endl;
}
```

II. Постановка задачи

1. Написать программу для создания объектов пользовательского класса (ввод исходной информации с клавиатуры с использованием перегруженной

операции ">>) и сохранения их в потоке (файле), чтения объектов из потока. сохранения их в массиве и просмотра массива. Для просмотра объектов использовать перегруженную для cout операцию <<.

a) Определить пользовательский тип данных (класс). Определить и реализовать в нем конструкторы, деструктор, операции присваивания, ввода и вывода для стандартных потоков.

b) Предусмотреть в программе вывод сообщения о количестве сохраненных объектов и о длине полученного файла в байтах.

c) Реализовать обработку исключительных ситуаций.

d) Выполнить тестирование программы.

2. Написать программу для корректировки (т.е. замены) записей в файле как указано в варианте.

Выполнить программу и просмотреть полученный файл.

3. Разработать нотацию для указанного варианта игры. Разработать классы для реализации игры. Предусмотреть варианты игры на двоих. В классе предусмотреть различные исключительные ситуации, которые могут, возникнут во время игры. С помощью классов решит следующую задачу. Во входном файле дан список ходов двух игроков. В отдельном файле, если это необходимо дано начальное расположение. Известно, что победил начинающий игрок. Проверить правильность записи игры. Наиболее общие исключительные ситуации:

Запись начального расположения фигур не соответствует нотации.
Неправильно дано начальное расположение фигур.
Запись хода не соответствует нотации.
Неправильный ход.
Выход за пределы поля игры.
Ход несуществующей фигурой
Начинающий игрок проиграл и.т.д.

III. Варианты

1.1 Абитуриент (имя, год рождения, набранный балл, средний балл аттестата).

1.2 Удалить элемент с указанным номером, добавить элемент после элемента с указанной фамилией.

1.3 Игра Гранди. Предметы раскладываются на произвольное количество куч. Ход разбиение любой кучи на две неравные части. Побеждает тот, кто сделал последний ход.

2.1 Сотрудник (имя, должность, год рождения, заработная плата).

2.2 Удалить элемент с указанной фамилией, добавить элемент после элемента с указанным номером.

2.3 Угадай слово по буквам. Неоткрытые буквы обозначаются звездочками *. Ход. назвать букву. Буквы в слове совпадающие с названной буквой открываются. Побеждает тот, кто первым назовёт слово.

3.1 Страна (название, столица, численность населения, занимаемая площадь).

3.1 Удалить все элементы с численностью населения меньше заданной, добавить элемент после элемента с указанным названием.

3.3 Дорожка. На концах дорожки длиной M находятся две разноцветные фишки. Ход продвинут свою фишку не более на K ходов вперёд или назад. У кого не остаётся хода тот проигрывает.

4.1 Государство (название, государственный язык, денежная единица, курс валюты).

4.2 Удалить элемент с указанным названием, добавить два элемента в конец файла.

4.3 Угадай число. Ход: Выбирается число m.

a) Если m=1 то m стирается, игрок получает очко.

b) Если m простое то m=m-1, игрок получает очко.

c) Если m=m1*m2 то вместо m записываются m1 и m2, игрок не получает очка. Побеждает тот кто набирает больше очков. Первоначально имеется одно число.

5.1 Человек (имя, домашний адрес, номер телефона, возраст).

5.2 Удалить все элементы с заданным возрастом, добавить элемент после элемента с заданным номером телефона.

5.3 Обобщенная игра Ним. Предметы раскладываются на произвольное количество куч. Ход – взять произвольное количество предметов из одной кучи (можно всё). Тот кто забирает последнюю взятку проигрывает.

6.1 Паспорт (имя, номер, дата выдачи, кем выдано).

6.2 Удалить элемент с указанным номером, добавить элемент после элемента с указанной фамилией.

6.3 Кто первым назовёт данное число. Сделать ход назвать натуральное число от 1 до K и прибавить его к сумме чисел названный в предыдущих ходах. При этом сумма не должна превышать M. Тот кто первым получить сумму равную M проигрывает.

7.1 Школьник (имя, класс, номер телефона, возраст).

7.2 Удалить все элементы с указанным возрастом, добавить элемент в начало файла.

7.3 Кто последним выберет предмет. Из обшей кучи M предметов выбирается не более K предметов. Проигрывает тот, кто последним выберет предмет.

8.1 Студент (имя, домашний адрес, группа, рейтинг).

8.2 Удалить все элементы, у которых рейтинг меньше заданного, добавить три элемента в конец файла.

8.3 Обобщенная игра Угадай слово по буквам. Слово вписано в прямоугольник по горизонтали или по вертикали. Неоткрытые буквы обозначаются звездочками *. Ход назвать букву. Если в названном слове имеются буквы совпадающие с названным, то они открываются. Побеждает то кто первым назовёт слово.

9.1 Группа (номер, курс, факультет, количество студентов).

9.2 Удалить все элементы с указанным курсом, добавить элемент после элемента с заданным номером.

9.3 Обобщенная игра Гранди. Предметы раскладываются на произвольное количество куч. Ход разбиение любой кучи на две неравные части. Тот кто сделал последний ход проигрывает.

10.1 Самолёт (марка, вместимость, компания, страна).

10.2 Удалить элементы с указанной маркой, добавить элемент перед элементом с заданной компанией.

10.3 Игра в города. Имеется список названий городов. Ход – назвать город, при этом первая буква названия должна совпадать с последней буквой предыдущего названия. Тот, кто не может назвать город проигрывает.

11.1 Покупатель (имя, домашний адрес, номер телефона, номер кредитной карточки).

11.2 Удалить три элемента из начала файла, добавить три элемента после элемента с указанной фамилией.

11.3 Кто наберет четное число предметов. Из обшей кучи М предметов выбирается не более К предметов. Победитель тот кто по окончании игры набирает четное число предметов.

12.1 Пациент (имя, домашний адрес, номер медицинской карты, номер страхового полиса).

12.2 Удалить элемент с заданным номером медицинской карты, добавить два элемента в начало файла.

12.3 Кто последним назовёт число. Сделать ход назвать натуральное число от 1 до К и прибавить его к сумме чисел названный в предыдущих ходах. Выигрывает тот кто первым получить сумму равную М.

13.1 Информация (носитель, объем, название, автор).

13.2 Удалить первый элемент с заданным объемом информации, добавить элемент перед элементом с указанным носителем.

13.3 Кто первым выберет предмет. Из обшей кучи М предметов выбирается не более К предметов. Побеждает тот, кто последним выберет предмет.

14.1 Видеокассета (название фильма, режиссер, продолжительность, цена).

14.2 Удалить все элементы с ценой выше заданной, добавить три элемента в конец файла.

14.3 Угадай слово. Неоткрытые буквы обозначаются звездочками *. Ход назвать слово. Если в названном слове имеются буквы загаданного слова, то они открываются. Побеждает тот, кто первым назовёт слово.

15.1 Музыкальный диск (название, автор, продолжительность, цена).

15.2 Удалить первый элемент с заданной продолжительностью, добавить два элемента после элемента с заданным номером.

15.3 Игра Ним. Частный вариант. Предметы раскладываются на произвольное количество куч. Ход – взять не более N предметов из одной кучи. Выигрывает тот, кто забирает последнюю взятку.

16.1 Спортивная команда (название. город. количество игроков, количество набранных очков).

16.2 Удалить все элементы с количеством очков меньше заданного, добавить три элемента в начало файла.

16.3 Шашки. Две белых шашки и две белых дамки против двоих черных шашек и дамки. Бить обязательно. Учитывать превращение в шашки в дамку.

17.1 Театр (название, адрес, вместимость, вид театрального искусства).

17.2 Удалить элемент с заданным названием, добавить два элемента после элемента с указанным номером.

17.3 Шашки. Четыре белых шашки против троих черных шашек. Бить обязательно. Учитывать превращение в дамку.

18.1 Стадион (название, год постройки, количество площадок, виды спорта).

18.2 Удалить все элементы, у которых год постройки больше заданного, добавить два элемента перед элементом с указанным номером.

18.3 Шахматы. Король, Конь, Ладьи и пешки против Короля, двух Коней и пешек.

19.1 Автомобиль (марка, серийный номер, регистрационный номер, год выпуска).

19.2 Удалить три элемента из начала файла, добавить элемент поле элемента с указанным регистрационным номером.

19.3 Морской бой. Поле каждого противника это символьная таблица в котором клетка занятая кораблем отмечается как 1. Если клетка под боем то она обозначается крестиком. Ход назвать координаты клетки поля противника.

20.1 Костюм (размер, страна, цена, цвет).

20.2 Удалить все элементы, у которых цена выше заданного, добавить элемент в начало файла.

20.3 Колония зайцев. Все белые шашки расположены в левом, все черные в правом углу доски. Цель передвинуть всю колонию на противоположный уголь. Сделать ход перепрыгнуть через несколько пешек.

21.1 Владелец автомобиля (имя, номер автомобиля, телефон, номер техпаспорта).

21.2 Удалить элемент с заданным номером, добавить два элемента перед элементом с заданной фамилией.

21.3 Крестики и нолики в большем поле. Поле – квадратная символьная таблица. Ход поставить крестик или нолик. Побеждает тот, кто первым поставить пять подряд идущих по горизонтали, вертикали или диагонали крестика или нолика.

22.1 Оборудование (название, помещение, материально ответственный, стоимость).

22.2 Удалить элемент с заданным названием, добавить два элемента перед элементом с заданной стоимостью.

22.3 Шахматы. Король, Конь, Слоны и пешки против Короля, двух Слонов, Коня и пешек.

23.1 Фильм (название, режиссер, год выпуска, приносимая прибыль).

23.2 Удалить элементы с прибылью меньше заданной, добавить элемент в начало файла.

23.3 Домино. Все костяшки распределены между игроками.

24.1 Товар (название, фирма, страна, стоимость).

24.2 Удалить два элемента из конца файла, добавить элемент после элемента с указанным названием.

24.3 Шахматы. Король, Ферзь, Конь и пешки против Короля, двух Слонов, Коня и пешек.

25.1 Книга (название, автор, год издания, количество страниц).

25.2 Удалить три элемента из начала файла, добавить элемент перед элементом с указанным названием.

25.3 Заяц (белая шашка) и Волки (четыре черных шашки). Задача зайца дойти до последней горизонтали. Задача волков окружить зайца.

IV. Содержание отчета

1. Титульный лист: название дисциплины, номер и наименование работы, фамилия, имя, отчество студента, дата выполнения..
2. Определение нотации для игры.
3. Классы для игры.

4. Исключительные ситуации для игры.
5. Стратегия игры с компьютером
6. Объяснение результатов.
7. Программа решения задания.
8. Результаты работы программы.
9. Объяснение результатов.

V. Методические указания

1. Перегрузка оператора вывода

```
class WordCount {
  friend ostream&
      operator<<( ostream&, const WordCount& );

public:
  WordCount( string word, int cnt=1 );
  // ...
private:
  string word;
  int occurs;
};
ostream& operator <<( ostream& os, const WordCount& wd )
{  // формат: <счетчик> слово
   os << "< " << " > " > "
     << wd.word;
   return os;
    }
```

2. Использование класса с перегрузкой оператора вывода

```
#include <iostream>
#include "WordCount.h"

int main()
{
  WordCount wd( "sadness", 12 );
  cout << "wd:\n" << wd << endl;
  return 0;
}
```

3. Использование функции-члена precision(int) и манипулятора setprecision()

```
#include <iostream>
#include <iomanip>
#include <math.h>
int main()
{
```

```
    cout << "Точность: "
       << cout.precision() << endl
       << sqrt(2.0) << endl;
     cout.precision(12);
            cout << "\nТочность: "
               << cout.precision() << endl
       << sqrt(2.0) << endl;
            cout << "\nТочность: "  << setprecision(3)
               << cout.precision() << endl
               << sqrt(2.0) << endl;
            return 0;
    }
```

4. Использование функции-члена cout.fill и манипулятора setw ()

```
    #include <iostream.h>
    #include <iomanip.h>
    void main(void)
    {
      cout << "Таблица информации" << endl;
      cout.fill (' . ');
      cout << "Профиль компании" << setw(20) << 10 << endl;
      cout << "Доходы и убытки компании" << setw(12) << 11 << endl;
      cout << "Члены правления компании" << setw(14) << 13 << endl;
    }
```

5. Пример использования исключительной ситуации при работе с файлами

```
    #include <iostream.h>
    class file_open_error {};
    class file_read_error {};
    class file_write_error {};
    void file_copy(char *source, char *target)
    {
char line[256];
ifstream input_file(source);
ofstream output_file(target);
if (input_file.fail())
   throw file_open_error();
else
   if (output_file.fail()) throw file_open_error();
else
      {
  while ((! input_file.eof()) && (! input_file.fail()))
        {
     input_file.getline(line, sizeof(line)) ;
     if (! input_file.fail()) output_file << line << endl;
```

```
        else throw file_read_error();
        if (output_file.fail()) throw file_write_error() ;
      }
    }
  }
      void main()
      {
      try
      {
   file_copy("SOURCE.TXT", "TARGET.TXT") ;
 };
      catch (file_open_error)
      {
   cerr << "Ошибка открытия исходного или целевого файла" << endl;
   exit(1);
 }
      catch (file_read_error)
      {
   cerr << "Ошибка чтения исходного файла" << endl;
   exit(1);
 }
      catch (file_write_error)
      {
   cerr << "Ошибка записи целевого файла" << endl;
   exit(1);
 }
      }
```

6. Завершение выполнения программы стандартным обработчиком:

```
#include <iostream.h>
class some_exception { };
void main(void)
{
 cout << "Перед генерацией исключительной ситуации" << endl;
 throw some_exception();
 cout << "Исключительная ситуация сгенерирована" << endl;
}
```

Лабораторная работа №6

Тема: Функции и методы высших порядков.

Цель: Получить практические навыки создания функциональных типов и функций объектов. Освоить технологию создания и использования функций и методов высшего порядка.

I. Краткие теоретические сведения.

Функцией высшего порядка называется такая функция (метод) класса, у которой один или несколько аргументов принадлежат к функциональному типу.

Использование указателей на функции

Рассмотрим функцию высшего порядка на примере задачи решения уравнения f(x)=0 в промежутке [a,b] для произвольной функции методом дихотомии. С этой целью создадим класс, в котором определим метод, решающий задачу. По сути самой задачи этот метод представляет собой функцию высшего порядка. Рассмотрим программный код, описывающий класс:

```
class FunctionZero
{
public:
static double dihotom(double a, double b, double eps, double(* f)(double x))
};
double FunctionZero::dihotom(double a, double b, double eps, double(*
f)(double x))
{
 float x, x1=a, x2=b;
 while (x2-x1)>eps
{x=(x1+x2)\2;
if (f(x)==0) return x;
if (f(x)>0) x1=x; else x2=x;
};
return x1;
}
```

- Класс **FunctionZero** предназначен для работы с функциями.
- Метод **dihotom** - основной метод класса позволяет решить задачу. Этот метод есть функция высшего порядка, поскольку одним из его аргументов является указатель на функцию, с одним аргументом типа double и возвращающих значение этого же типа.
- Впоследствии класс может быть расширен, и помимо метода дихотомии он может включать другие методы.

Использование шаблонов

При использовании шаблонов необходимо для каждой функции создать класс в котором имеется метод для вычисления значения функции.

Объекты-функции – это экземпляры класса, в котором определена операция «круглые скобки» (). В ряде случаев удобно заменить функцию на объект-функцию. Когда объект-функция используется в качестве функции, то для ее вызова используется operator ().

Пример:

```
class kub{
public:
double operator()(double x)
{return x*x*x;}
};
```

Рассмотрим программный код, описывающий шаблон класса с методом дихотомии:

```
template <class T>
class FunctionZero
{
public:
static double dihotom(double a, double b, double eps, T f)
};

template <class T>
double FunctionZero<T>::dihotom(double a, double b, double eps, T f)
{
 float x, x1=a, x2=b;
 while (x2-x1)>eps
{x=(x1+x2)\2;
if (f(x)==0) return x;
if (f(x)>0) x1=x; else x2=x;
};
return x1;
}
```

Наследование и полиморфизм - альтернатива обратному вызову

Программная система в объектно-ориентированном стиле представляет собой одно или несколько семейств интерфейсов и классов, связанных отношением наследования. Классы-потомки наследуют методы своих родителей, могут их переопределять и добавлять новые методы. Переопределив метод родителя, потомки без труда могут вызывать как собственный метод, так и метод родителя; все незакрытые методы родителя им известны и доступны.

Вначале построим интерфейс то есть абстрактный класс функции имеющий чисто виртуальный метод вычисления значения.

```
class function
{ public:
virtual double value(double )=0;
};
```

Теперь построим класс, метод которого будет решать задачу для некоторой функции, заданной виртуальным методом класса.

```
class FunctionZero
{
public:
static double dihotom(double a, double b, double eps, function& f)
};

double FunctionZero::dihotom(double a, double b, double eps, function& f)
{
 float x, x1=a, x2=b;
 while (x2-x1)>eps
{x=(x1+x2)\2;
if (f.value(x)==0) return x;
if (f.value(x)>0) x1=x; else x2=x;
};
return x1;
}
```

Для вычисления интеграла от реальной функции единственное, что теперь нужно сделать - это задать класс-потомок, переопределяющий виртуальный метод. Вот пример такого класса:

```
class func1:public function
{
  public:
 double value(double x)
   {
     int k = 1; int b = 2;
     return (double)(k*x +b);
   };

 class func2:public function
 public:
 double value(double x)
   {
     double a = 1.0; double b = 2.0; double c= 3.0;
     return (double)(a*x*x +b*x +c);
   }
 }
```

II. Постановка задачи

1. Написать и реализовать программу для решения задачи связанной с применением методов для произвольной функции с использованием указателей на функции.

a) Создать класс в котором имеется метод высшего порядка для решения указанной задачи.

b) Реализовать несколько функций для которых известно точное решение задачи. Применить к ним функцию высшего порядка.

c) Сравнить полученные решения с точными для нескольких значений критерия точности и заданного количество шагов.

2. Написать и реализовать программу для решения задачи связанной с применением методов для произвольной функции с использованием шаблонов и объектов функций.

3. Написать и реализовать программу для решения задачи интегрирования для произвольной функции с использованием наследования и интерфейсов.

III. Варианты

	Методы	**Интегрирование**
	Методы решения нелинейных уравнений	**Квадратурные формулы**
1	Метод пошагового спуска	Формула прямоугольников
2	Метод простой итерации	Формула Ньютона
3	Метод хорд	Формула Симпсона
4	Метод пошагового спуска	Формула средних
5	Метод секущих	Формула Ньютона - Котеса
6	Разностный метод Ньютона	Формула Чебышева
	Методы решения дифференциальных уравнений	
7	Метод Эйлера	Формула Гаусса
8	Усовершенствованный метод Эйлера	Формула Ньютона - Котеса
9	Усовершенствованный метод Эйлера-Коши	Формула Чебышева
	Методы минимизации функции от одной переменной	
10	Метод пошагового спуска	Формула прямоугольников
11	Метод дихотомии	Формула Ньютона
12	Метод золотого сечения	Формула Симпсона
13	Метод Фибоначчи	Формула средних
14	Разностный метод Ньютона	Формула Гаусса
15	Метод парабол	Формула Ньютона - Котеса

	Методы решения нелинейных уравнений	
16	Метод Ньютона	Формула Уэддла
17	Метод Ньютона – Рафсона	Метод Монте Карло
	Методы решения дифференциальных уравнений	
18	Метод Рунге Кутта третьего порядка	Четырехполосная формула
19	Метод Рунге Кутта четвертого порядка	Пятиполосная формула
20	Метод Адамса третьего порядка	Шестиполосная формула
21	Метод Адамса четвертого порядка	Метод Монте Карло
	Методы минимизации функции от одной переменной	
22	Метод Ньютона	Четырехполосная формула
23	Метод касательных	Пятиполосная формула
24	Метод секущих	Шестиполосная формула
25	Метод Ньютона – Рафсона	Формула Уэддла

IV. Содержание отчета

1. Титульный лист: название дисциплины, номер и наименование работы, фамилия, имя, отчество студента, дата выполнения.
2. Постановка задачи.
3. Следует дать конкретную постановку, т.е указать шаблон какого класса должен быть создан, какие должны быть в нем конструкторы, компоненты-функции, перегруженные операции и т.д.
4. То же самое следует указать для пользовательского класса.
5. Определение шаблона класса с комментариями.
6. Определение пользовательского класса с комментариями.
7. Реализация конструкторов, деструктора, операции присваивания и операций, которые заданы в варианте задания.
8. То же самое для пользовательского класса.
9. Результаты тестирования. Следует указать для каких типов и какие операции проверены и какие выявлены ошибки (или не выявлены)

V. Методические указания

1. Если в методе используются производные то при использовании указателей на функции необходимо ввести указатели на производные как дополнительные параметры, при использовании шаблонов или интерфейсов их надо представлять как дополнительных методов класса представляющего функцию.
2. При решении нелинейных уравнений вычисления надо проводить пока не выполнится условие: $|x_i - x_{i-1}| < \varepsilon$.

3. При решении дифференциальных уравнений вычисления надо проводить пока не выполнится условие: $\max(|y1_i-y0_i|)<\varepsilon$. Здесь $y0_i$, результаты, полученные до дробления шага, а $y1_i$, результаты, полученные после дробления шага.

4. При вычислении интегралов вычисления надо проводить пока не выполнится условие: $|S1-S0|<\varepsilon$. Здесь S0, результаты, полученные до дробления шага, а S1, результаты, полученные после дробления шага.

4. Функции алгоритмы реализовать как статические методы высшего порядка классов.

5. Рассмотрим задачу вычисления определенного интеграла с заданной точностью с помощью метода трапеций на основе абстрактных классов. С этой целью создадим класс, в котором определим метод, вычисляющий интеграл. Рассмотрим программный код, описывающий класс:

```
class HighOrderIntegral
{
double Trap(double a, double b, int n, functions& sif)
public:
  static double EvalIntegral(double a, double b, double eps, SubIntegralFun
sif);
};
double HighOrderIntegral:: EvalIntegral(double a, double b, double eps,
functions& sif)
 {
  int n=4;
  double I0=0, I1 = Trap( a, b, n,sif);
  for( n=8; n < Pow(2,15); n*=2)
  {
    I0 =I1; I1=Trap(a,b,n,sif);
    if(Abs(I1-I0)<eps) break;
  }
  if(Abs(I1-I0)< eps) {
    cout<< "Требуемая точность достигнута! "<< eps;
    cout<< "достигнутая точность ="<<Abs(I1-I0);
     cout<< "количество шагов ="<<n<<endl;}
  else
   {
    cout<< "Требуемая точность не достигнута! "<< eps;
    cout<< "достигнутая точность ="<<Abs(I1-I0);
     cout<< "количество шагов ="<<n<<endl;}
  return(I1);
};
double HighOrderIntegral:: Trap(double a, double b, int n,
SubIntegralFun sif)
{
  //Вычисляет частную сумму по методу трапеций
```

```
    double x = a, sum = sif(x)/2, dx = (b-a)/n;
    for (int i= 2; i <= n; i++)
    {
      x += dx;       sum += sif(x);
    }
    x = b; sum += sif(x)/2;
    return(sum*dx);
  }
}//class HighOrderIntegral
```

• Класс HighOrderIntegral предназначен для работы с функциями.

• Для вычисления интеграла применяется классическая схема. Интервал интегрирования разбивается на n частей, и вычисляется частичная сумма по методу трапеций, представляющая приближенное значение интеграла. Затем n удваивается, и вычисляется новая сумма. Если разность двух приближений по модулю меньше заданной точности eps, то вычисление интеграла заканчивается, иначе процесс повторяется в цикле. Цикл завершается либо по достижении заданной точности, либо когда n достигнет некоторого предельного значения (в нашем случае - 215).

• Вычисление частичной суммы интеграла по методу трапеций реализовано закрытой процедурой Trap.

6. Пример, выполняющий создание нужных объектов и тестирующий их работу:

```
void main ()
{
  func1 sif1;
  func2 sif2;
  HighOrderIntegral<sif1>::EvalIntegral(2,3,0.1e-5, sif1);
  HighOrderIntegral::EvalIntegral(2,3,0.1e-5, sif2);
}
```

Лабораторная работа №7

Тема: Стандартная библиотека шаблонов STL.

Цель: Освоить технологию программирования с использованием библиотеки стандартных шаблонов (STL) языка C++.

I. Краткие теоретические сведения.

Состав STL.

Ядро библиотеки образуют три элемента: **контейнеры, алгоритмы** и **итераторы.**

Контейнеры (containers) – это объекты, предназначенные для хранения других элементов. Например, вектор, линейный список, множество.

Ассоциативные контейнеры (associative containers) позволяют с помощью ключей получить быстрый доступ к хранящимся в них значениям.

В каждом классе-контейнере определен набор функций для работы с ними. Например, список содержит функции для вставки, удаления и слияния элементов.

Алгоритмы (algorithms) выполняют операции над содержимым контейнера. Существуют алгоритмы для инициализации, сортировки, поиска, замены содержимого контейнеров. Многие алгоритмы предназначены для работы с последовательностью (sequence), которая представляет собой линейный список элементов внутри контейнера.

Итераторы (iterators) – это объекты, которые по отношению к контейнеру играют роль указателей. Они позволяют получить доступ к содержимому контейнера примерно так же, как указатели используются для доступа к элементам массива.

Классы-контейнеры.

В STL определены следующие классы-контейнеры (в угловых скобках указаны заголовочные файлы, где определены эти классы):

Основные контейнеры

vector динамический массив<vector.h>

list линейный список<list.h>

deque двусторонняя очередь<deque.h>

set множество<set.h>

multiset множество, в котором каждый элемент не обязательно уникален <set.h>

map ассоциативный список для хранения пар ключ / значение, где с каждым ключом связано одно значение <map.h>

multimap с каждым ключом связано два или более значений <map.h>

Производные контейнеры

stack	стек<stack.h>
queue	очередь <queue.h>
priority_queue	очередь с приоритетом <queue.h>

Конструкторы

Любой контейнерный класс содержит конструктор по умолчанию, копирующий конструктор и деструктор.

Например конструкторы и деструктор контейнерного класса вектор:

vector<elem> c	Создает пустой вектор не содержащий ни одного элемента
vector<elem> c1(c2)	Создает копию другого вектора того же типа (с копированием всех элементов)
vector<elem> c(n)	Создает вектор из n элементов, создаваемых конструктором по умолчанию
vector<elem> c(n,x)	Создает вектор, инициализируемый n копиями элемента x
~vector<elem>()	Уничтожает все элементы и освобождает память

Для любого объекта, который будет храниться в контейнере, должен быть определен конструктор по умолчанию. Кроме того, для объекта должны быть определены операторы < и ==.

Итераторы

С итераторами можно работать так же, как с указателями. К ним можно применить операции *, инкремента, декремента. Типом итератора объявляется тип iterator, который определен в различных контейнерах.

Существует пять типов итераторов:

1. Итераторы ввода (input_iterator) поддерживают операции равенства, разыменования и инкремента.

==, !=, *i, ++i, i++, *i++

Специальным случаем итератора ввода является istream_iterator.

2. Итераторы вывода (output_iterator) поддерживают операции разыменования, допустимые только с левой стороны присваивания, и инкремента.

++i, i++, *i=t, *i++=t

Специальным случаем итератора вывода является ostream_iterator.

3. Однонаправленные итераторы (forward_iterator) поддерживают все операции итераторов ввода/вывода и, кроме того, позволяют без ограничения применять присваивание.

==, !=, =, *i, ++i, i++, *i++

4. Двунаправленные итераторы (biderectional_iterator) обладают всеми свойствами forward-итераторов, а также имеют дополнительную операцию

декремента (--i, i--, *i--), что позволяет им проходить контейнер в обоих направлениях.

5. **Итераторы произвольного доступа** (random_access_iterator) обладают всеми свойствами biderectional-итераторов, а также поддерживают операции сравнения и адресной арифметики, то есть непосредственный доступ по индексу.

i+=n, i+n, i-=n, i-n, i1-i2, i[n], i1<i2, i1<=i2, i1>i2, i1>=i2

В STL также поддерживаются **обратные итераторы** (reverse iterators). Обратными итераторами могут быть либо двунаправленные итераторы, либо итераторы произвольного доступа, но проходящие последовательность в обратном направлении.

Распределители памяти, предикаты и функции сравнения.

Вдобавок к контейнерам, алгоритмам и итераторам в STL поддерживается ещё несколько стандартных компонентов. Главными среди них являются **распределители памяти, предикаты и функции сравнения.**

У каждого контейнера имеется определенный для него распределитель памяти (**allocator**), который управляет процессом выделения памяти для контейнера.

По умолчанию распределителем памяти является объект класса **allocator**. Можно определить собственный распределитель.

В некоторых алгоритмах и контейнерах используется функция особого типа, называемая **предикатом**. Предикат может быть унарным и бинарным. Возвращаемое значение: истина либо ложь. Точные условия получения того или иного значения определяются программистом. Тип унарных предикатов **UnPred**, бинарных – **BinPred**. Тип аргументов соответствует типу хранящихся в контейнере объектов.

Определен специальный тип бинарного предиката для сравнения двух элементов. Он называется **функцией сравнения** (comparison function). Функция возвращает истину, если первый элемент меньше второго. Типом функции является тип **Comp**.

Особую роль в STL играют объекты-функции.

Объекты-функции – это экземпляры класса, в котором определена операция «круглые скобки» (). В ряде случаев удобно заменить функцию на объект-функцию. Когда объект-функция используется в качестве функции, то для ее вызова используется operator ().

Контейнера *vector*-вектор.

Вектор vector в STL определен как динамический массив с доступом к его элементам по индексу.

template<class T,class Allocator=allocator<T>>class std::vector{...};

где *T* – тип предназначенных для хранения данных.

Allocator задает распределитель памяти, который по умолчанию является стандартным.

В классе vector определены следующие конструкторы:

explicit vector(const Allocator& a=Allocator());

explicit vector(size_type число, const T&значение= T(), const Allocator&a= =Allocator());

vector(const vector<T,Allocator>&объект);

template<class InIter>**vector**(InIter начало, InIter конец, const Allocator&a= =Allocator());

Первая форма представляет собой конструктор пустого вектора.

Во второй форме конструктора вектора число элементов – это число, а каждый элемент равен значению значение. Параметр значение может быть значением по умолчанию.

Третья форма конструктора вектор – это конструктор копирования.

Четвертая форма – это конструктор вектора, содержащего диапазон элементов, заданный итераторами начало и конец.

Конструкторы для вектора.

```
vector<int> a;
vector<double> x(5);
vector<char> c(5,'*');
vector<int> b(a); //b=a
```

Для любого объекта, который будет храниться в векторе, должен быть определен конструктор по умолчанию. Кроме того, для объекта должны быть определены операторы < и ==.

Для класса вектор определены следующие операторы сравнения:
==, <, <=, !=, >, >=.

Кроме этого, для класса vector определяется оператор индекса [].

- Новые элементы могут включаться с помощью функций *insert(), push_back(), resize(), assign().*
- Существующие элементы могут удаляться с помощью функций *erase(), pop_back(), resize(), clear().*
- Доступ к отдельным элементам осуществляется с помощью итераторов *begin(), end(), rbegin(), rend(),*
- Манипулирование контейнером, сортировка, поиск в нем и тому подобное возможно с помощью глобальных функций файла – заголовка <algorithm.h>.

Двусторонняя очередь (Deque)

deque - вид последовательности, которая, подобно вектору, поддерживает итераторы произвольного доступа. Кроме того она поддерживает операции вставки и стирания в начале или в конце за постоянное время; вставка и стирание в середине занимают линейное время. Как с векторами, управление памятью обрабатывается автоматически.

Спецификация шаблона для класса Deque:

template <class T, template <class U> class Allocator = allocator> class deque {…}

Список (List)

Список - вид последовательности, которая поддерживает двунаправленные итераторы и позволяет операции вставки и стирания с постоянным временем в любом месте последовательности, с управлением памятью, обрабатываемым автоматически. В отличие от векторов и двусторонних очередей, быстрый произвольный доступ к элементам списка не поддерживается, но многим алгоритмам, во всяком случае, только и нужен последовательный доступ.

Спецификация шаблона для класса list:

template <class T, template <class U> class Allocator = allocator> class list { ... }

Ассоциативные контейнеры (массивы).

Ассоциативный массив содержит пары значений. Зная одно значение, называемое **ключом** (key), мы можем получить доступ к другому, называемому **отображенным значением** (mapped value).

Ассоциативный массив можно представить как массив, для которого индекс не обязательно должен иметь целочисленный тип:

V& operator[](const K&) возвращает ссылку на V, соответствующий K.

Ассоциативные контейнеры – это обобщение понятия ассоциативного массива.

Ассоциативный контейнер **map** – это последовательность пар (ключ, значение), которая обеспечивает быстрое получение значения по ключу. Контейнер map предоставляет двунаправленные итераторы.

Ассоциативный контейнер map требует, чтобы для типов ключа существовала операция "<". Он хранит свои элементы отсортированными по ключу так, что перебор происходит по порядку.

Спецификация шаблона для класса map:

template<class Key,class T,class Comp=less<Key>,class Allocator=allocator<pair> >

class std::map

В классе map определены следующие конструкторы:

explicit **map**(const Comp& c=Comp(),const Allocator& a=Allocator());

map(const map<Key,T,Comp,Allocator>& ob);

template<class InIter> **map**(InIter first,InIter last,const Comp& c=Comp(),const Allocator& a=Allocator());

Первая форма представляет собой конструктор пустого ассоциативного контейнера, вторая – конструктор копии, третья – конструктор ассоциативного контейнера, содержащего диапазон элементов.

Определена операция присваивания:

map& **operator=**(const map&);

Определены следующие операции: ==, <, <=, !=, >, >=.

В map хранятся пары ключ/значение в виде объектов типа **pair**.

Создавать пары ключ/значение можно не только с помощью конструкторов класса pair, но и с помощью функции **make_pair**, которая создает объекты типа pair, используя типы данных в качестве параметров.

Типичная операция для ассоциативного контейнера – это ассоциативный поиск при помощи операции индексации ([]).

mapped_type& **operator[]**(const key_type& K);

Множества **set** можно рассматривать как ассоциативные массивы. в которых значения не играют роли, так что мы отслеживаем только ключи.

template<classT,class Cmp=less<T>,class Allocator=allocator<T>>class std::set{…};

Множество, как и ассоциативный массив, требует, чтобы для типа T существовала операция "меньше" (<). Оно хранит свои элементы отсортированными, так что перебор происходит по порядку.

Методы контейнеров

Типы

value_type тип элемента

allocator_type тип распределителя памяти

size_type тип индексов, счетчика элементов и т.д.

iterator ведет себя как value_type*

reverse_iterator просматривает контейнер в обратном порядке

reference ведет себя как value_type&

key_type тип ключа (только для ассоциативных контейнеров)

key_compare тип критерия сравнения (только для ассоциативных контейнеров)

mapped_type тип отображенного значения

Методы получения итераторов

begin() указывает на первый элемент

end() указывает на элемент, следующий за последним

rbegin() указывает на первый элемент в обратной последовательности

rend() указывает на элемент, следующий за последним в обратной последовательности

Доступ к элементам

front() ссылка на первый элемент

back() ссылка на последний элемент

operator[](i) доступ по индексу без проверки

at(i) доступ по индексу с проверкой

Методы включения элементов

insert(p,x) добавление x перед элементом, на который указывает p

insert(p,n,x) добавление n копий x перед p

insert(p,first,last) добавление элементов из [first:last] перед p

push_back(x) добавление x в конец

push_front(x) добавление нового первого элемента (только для списков и очередей с двумя концами)

Методы удаления элементов

erase(p) удаление элемента в позиции p

erase(first,last) удаление элементов из [first:last]

pop_back() удаление последнего элемента

pop_front() удаление первого элемента (только для списков и очередей с двумя концами)

Метод присваивания

operator=(x) контейнеру присваиваются элементы контейнера x

assign(n.x) присваивание контейнеру n копий элементов x (не для ассоциативных контейнеров)

assign(first,last) присваивание элементов из диапазона [first:last]

Ассоциативные методы

find(elem) находит позицию первого элемента с значением elem

lower_bound(elem) находит первую позицию куда может быть вставлен элемент

upper_bound(elem) находит последнюю позицию куда может быть вставлен элемент

equal_range(elem) находит первую и последнюю позицию куда может быть вставлен элемент

Ассоциативные методы

operator[](k) доступ к элементу с ключом k

find(k) находит позицию элемента с ключом k

lower_bound(k) находит позицию первого элемента с ключом k

upper_bound(k) находит первый элемент с ключом, большим k

equal_range(k) находит lower_bound (нижнюю границу) и upper_bound (верхнюю границу) элементов с ключом k

Другие методы

size() число элементов

empty() контейнер пуст?

capacity() память, выделенная под вектор (только для векторов)

reserve(n) выделяет память для контейнера под n элементов

resize(n) изменяет размер контейнера (только для векторов, списков и очередей с двумя концами)

swap(x) обмен местами двух контейнеров

==, !=, < операции сравнения

II. Постановка задачи

1. Написать и отладить программу, которая демонстрирует использование контейнерного класса стек библиотеки STL для решения задачи указанной в варианте задания.

В программе выполнить следующее:

a) Создать объект-контейнер класса стек.

b) Реализовать функции необходимые для решения задачи.

c) Протестировать программу.

2. Написать и отладить программу, которая демонстрирует использование контейнерных классов библиотеки STL для решения задачи указанной в варианте задания.

В программе выполнить следующее:

a) Создать объект-контейнер класса необходимого для решения указанного в варианте задачи.

b) Реализовать функции необходимые для решения задачи.

c) Протестировать программу.

3. Написать и реализовать программу, которая демонстрирует использование контейнерных классов библиотеки STL , итераторов и связанных с ними методов класса для типа данных указанных в варианте.

В программе выполнить следующее:

a) Создать объект-контейнер класса необходимого для решения указанного в варианте задачи.

b) Просмотреть контейнер используя итераторы.

c) Изменить контейнер, удалив из него одни элементы и добавив другие.

d) Просмотреть контейнер используя итераторы.

III. Варианты

1.1. С помощью стека решить задачу проверки правильности записи выражения состоящего из круглых скобок.

1.2. Перенести символы из выражения в дек таким образом, чтобы сначала шли цифры затем буквы.

1.3. Сформировать словарь слов с ключами. Удалить из него K элементов, начиная с заданного номера, добавить элемент перед элементом с заданным ключом.

2.1. С помощью стека решить задачу проверки правильности записи выражения состоящего из круглых и прямоугольных скобок.

2.2. Перенести символы из выражения в дек таким образом, чтобы сначала шли согласные затем гласные буквы.

2.3. Сформировать словарь слов с ключами. Удалить из него элемент с заданным ключом, добавить элемент перед элементом с заданным ключом;

3.1. С помощью стека решить задачу проверки правильности записи выражения состоящего из круглых, прямоугольных и фигурных скобок.

3.2. В произвольном порядке вводятся числа. С помощью дека распечатать сначала четные затем нечетные числа.

3.3. Сформировать словарь слов с ключами. Удалить из него элемент перед элементом с заданным ключом, добавить K элементов в начало словаря.

4.1. С помощью стека проверить правильность записи выражения в виде 0^n1^n.

4.2. С помощью приоритетной очереди найти заданное количество наибольших чисел из чисел, хранящихся в файле.

4.3. Сформировать словарь слов с ключами. Удалить элемент с заданным ключом. Добавить по К элементов в начало и в конец списка.

5.1. С помощью стека проверить правильность записи выражения в котором должно быть равное количество букв a и b.

5.2. С помощью приоритетной очереди из текста заданного в файле выделить заданное количество наиболее длинных слов.

5.3. Сформировать словарь слов с ключами. Удалить из него элемент с заданным ключом, добавить элемент с указанным номером.

6.1. С помощью стека проверить правильность записи выражения в котором после каждой подчепочки состоящей из буквы a должна идти подчепочка такой же длины из буквы b (например aabbabaaaabbbb).

6.2. С помощью приоритетной очереди выполнить слияние большого количества небольших упорядоченных файлов в один упорядоченный файл.

6.3. Сформировать мультисловарь слов с ключами. Удалить элемент с заданным номером. Добавить по К элементов перед и после элемента с заданным ключом

7.1. С помощью стека проверить правильность парных тегов форматирования в HTML странице (например <A>, <B>,<I> и.т.д). Область действия тегов не является пересекающимся.

7.2. С помощью приоритетной очереди решить задачу упаковки максимального количество грузов в корзину с ограниченной емкостью.

7.3. Сформировать мультисловарь слов с ключами. Удалить из него первый максимальный элемент, добавить К элементов перед элементом с заданным ключом

8.1. С помощью стека проверить правильность записи выражения в виде прямой польской записи. Операндами являются буквы латинского алфавита.

8.2. С помощью приоритетной очереди решить задачу упаковки максимального количество слов в файл с ограниченной емкостью.

8.3. Сформировать мультисловарь слов с ключами. Удалить из него К элементов, начиная с заданного номера, добавить К элементов перед элементом с заданным ключом.

9.1. С помощью стека проверить правильность записи выражения в виде обратной польской записи. Операндами являются буквы латинского алфавита.

9.2. Сформировать множество служебных слов языка Паскаль. В заданном тексте программы напечатать все слова, которые не являются служебными.

9.3. Сформировать мультисловарь слов с ключами. Удалить из него последний максимальный элемент, добавить К элементов перед элементом с заданным ключом

10.1. С помощью стека проверить правильность арифметического выражения в бесскобочной записи. Операндами являются буквы латинского алфавита.

10.2. Сформировать множество служебных слов языка С++. В заданном тексте программы напечатать все слова которые не являются служебными.

10.3. Сформировать мультисловарь слов с ключами. Добавить в него элемент с заданным значением, удалить К элементов с заданным ключом.

11.1. С помощью стека проверить правильность логического выражения в бесскобочной записи. Операндами являются буквы латинского алфавита.

11.2. Для программы языка Паскаль составить множество переменных объявленных в разделе объявлений. С помощью множества проверить имеются ли необъявленные переменные в теле программы, состоящей из операторов присвоения.

11.3. Сформировать дек целых чисел. Удалить первый элемент равный 0. Добавить после каждого четного элемента массива элемент со значением M[I-1]+2.

12.1. С помощью стека преобразовать в правильную запись выражение. состоящее из круглых скобок путем вставки символов. Вывести сообщение если это невозможно.

12.2. С помощью мультимножества определить состоят два текста из одинаковых слов.

12.3. Сформировать дек целых чисел. Удалить все элементы равные 0.

Добавить после первого четного элемента массива элемент со значением M[I-1]+2.

13.1. С помощью стека преобразовать в правильную запись выражение. состоящее из круглых, прямоугольных скобок и фигурных скобок путем вставки символов. Вывести сообщение если это невозможно.

13.3. С помощью словаря для текста заданного в файле сформировать и вывести частотный словарь, в котором слова отсортированы, по количеству обращений.

13.3. Сформировать дек целых чисел. Удалить элементы, индексы которых кратны 3. Добавить после каждого отрицательного элемента массива элемент со значением | M[I-1]+1|.

14.1. С помощью стека реализовать перевод выражения в обратной польской записи в прямую польскую запись. Операндами являются буквы латинского алфавита.

14.2. С помощью мультисловаря реализовать формирование англо-русского словаря в котором слова отсортированы в лексикографическом порядке. Реализовать подстрочный перевод текста заданного в файле.

14.3. Сформировать дек целых чисел. Удалить элемент с заданным номером. Добавить после первого четного элемента массива элемент со значением M[I-1]+2.

15.1. С помощью стека реализовать перевод выражения в прямой польской записи в обратную польскую запись. Операндами являются буквы латинского алфавита.

15.2. Построить хеш таблицу переменных с помощью мультисловаря. В переменных значимыми считать первые три символа.

15.3. Сформировать дек целых чисел. Удалить 5 последних элементов массива. Добавить в начало массива 3 элемента с значением M[I+1]+2.

16.1. С помощью стека реализовать перевод выражения в обратной польской записи в представления в виде триад. Операндами являются буквы латинского алфавита.

16.2. Сформировать словарь переменных и их типов, а также типов и их приоритетов в языке C++. С помощью словарей проверить совместимость типов в заданном выражении

16.3. Сформировать мультимножество вещественных чисел. Удалить из него все минимальные элементы, добавить К элементов в начало.

17.1. С помощью стека реализовать перевод выражения в прямой польской записи в представления в виде триад. Операндами являются буквы латинского алфавита.

17.2. С помощью очереди найти все вершины, до которых можно дойти от заданной вершины в ненаправленном графе.

17.3. Сформировать мультимножество вещественных чисел. Удалить все элементы с заданным значением, добавить К элементов в конец.

18.1. С помощью стека реализовать перевод логического выражения в бесскобочной записи в обратную польскую запись. Операндами являются буквы латинского алфавита.

18.2. С помощью очереди найти все вершины, до которых можно дойти от заданной вершины в ненаправленном графе за М шагов.

18.3. Сформировать мультимножество вещественных чисел. Удалить последний элемент равный 0. Добавить элемент с заданным значением.

19.1. С помощью стека реализовать перевод арифметического выражения в бесскобочной записи в прямую польскую запись. Операндами являются буквы латинского алфавита.

19.2. С помощью очереди определит, является ли ненаправленный граф связным.

19.3. Сформировать мультимножество вещественных чисел. Удалить из него последний минимальный элемент, добавить К элементов.

20.1. С помощью стека реализовать перевод логического выражения в бесскобочной записи в форму триад. Операндами являются буквы латинского алфавита.

20.2. С помощью очереди найти компоненты связности в ненаправленном графе.

20.3. Сформировать мультимножество вещественных чисел. Удалить из него все максимальные элементы, добавить минимальный элемент.

21.1. С помощью стека реализовать перевод арифметического выражения в бесскобочной записи в форму триад. Операндами являются буквы латинского алфавита.

21.2. С помощью очереди определить существует ли цикл в направленном графе.

21.3. Сформировать двунаправленный список. Удалить из него все минимальные элементы, добавить в начало максимальный элемент

22.1. С помощью стека реализовать вычисление значения выражения в обратной польской записи, в котором операндами являются буквы латинского алфавита.

22.2. С помощью очереди определить существует ли путь от первой вершины до последней в направленном графе.

22.3. Сформировать двунаправленный список. Удалить из него все повторяющиеся элементы, добавить в конец первый неповторяющийся элемент

23.1. С помощью стека реализовать вычисление значения выражения в прямой польской записи, в котором операндами являются буквы латинского алфавита.

23.2. С помощью очереди решить следующие задачу. Даны обозначения двух полей шахматной доски (например, A_5 и C_2). Найти минимальное число ходов, которые нужны шахматному коню для перехода с первого поля на второе.

23.3. Сформировать двунаправленный список. Удалить элементы кратные 7. Добавить после каждого нечетного элемента списка элемент со значением 0.

24.1. С помощью стека реализовать вычисление арифметического выражения в бесскобочной записи. Операндами являются буквы латинского алфавита.

24.2. С помощью очереди решить следующие задачу. Дано обозначение поля шахматной доски (например, A_5). Найти все поля до которых может дойти конь с заданного поля.

24.3. Сформировать двунаправленный список. Удалить последний элемент равный 0. Добавить после элемента списка с заданным индексом элемент со значением 100.

25.1. С помощью стека реализовать вычисление логического выражения в бесскобочной записи. Операндами являются буквы латинского алфавита.

25.2. С помощью очереди решить следующие задачу. Дано обозначение поля шахматной доски (например, A_5 и C_2). Найти все поля до которых может дойти конь с заданного поля за M ходов.

25.3. Сформировать двунаправленный список. Удалить все элементы с заданным значением. Добавить перед каждым четным элементом списка элемент со значением 0.

IV. Содержание отчета

1. Титульный лист: название дисциплины, номер и наименование работы, фамилия, имя, отчество студента, дата выполнения.
2. Постановка задачи.
3. То же самое следует указать для пользовательского класса.
4. Определение шаблона класса с комментариями.
5. Определение пользовательского класса с комментариями.
6. Реализация конструкторов, деструктора, операции присваивания и операций, которые заданы в варианте задания.
7. То же самое для пользовательского класса.

V. Методические указания

1. Доступ к вектору через итератор.

```
#include<iostream>
#include<vector >
using namespace std;
void main()
{vector<int> v;
int i;
for(i=0;i<10;i++)v.push_back(i);
cout<<"size="<<v.size()<<"\n";
vector<int>::iterator p=v.begin();
while(p!=v.end())
{cout<<*p<<" ";p++;
}
}
```

2. Пример использования деков.

```
#include<iostream>
#include<deque>
#include<string>
```

```
using namespace std;
void main()
{
//Создание пустого дека для строк
deque<int> coll;
//вставить несколько элементов
coll.assign (3,string("string"));
coll.push_back("last string");
coll.push_front("first string");
//Вывод элементов
deque<int>::iterator pos;
for(pos=coll.begin();pos!=coll.end();++pos) cout<<*pos<<" ";
cout<<endl;
//Удаление первого и последнего элемента
coll.pop_front();
coll.pop_back ();
//Вставка во все элементы кроме первого
for(unsigned i=0; i<coll.size; ++i ) coll[i]="another"+coll[i];
//Увеличение размера до 4 элементов
coll.resize(4,"resized string");
//Вывод элементов
for(pos=coll.begin();pos!=coll.end();++pos) cout<<*pos<<" ";
cout<<endl;
}
```

3. Пример работы с двумя списками.

```
#include<iostream>
#include<list>
using namespace std;
void printsList(const list1&<int>, const list2&<int>)
{
set<int>::iterator pos;
cout<<"list1:";
for(pos=l1.begin();pos!=l1.end();++pos) cout<<*pos<<" ";
cout<<endl<<"list1:";
for(pos=l2.begin();pos!=l2.end();++pos) cout<<*pos<<" ";  }
void main()
{list<int> list1,list2; // Создание двух пустых списков
// Вставка элементов
for(int i=0;i<6; i++) { list1.push_back(i); list2.push_front(i);}
// Вывод элементов
printsList(l1,l2);
// Сортировка, присваивание и удаление дубликатов
list2.sort();
list1=list2;
list2.unique();
```

```
// Вывод элементов
printsList(l1,l2);
// Слияние двух отсортированных списков в первом списке
list1.merge(list2);
// Вывод элементов
printsList(l1,l2);
}
```

4. В отображении хранятся данные биржевых котировок. Элемент представляет пару, в которой ключом является название акции, значением ее цена.

```
# include <iostream.h>
# include<map>
# include<string>
using namespace std:
int main()
typedef map<string,float> StringFloatMap;
StringFloatMap Stocks; //Создание пустого контейнера
//Вставка нескольких элементов
Stocks["BASF"]=369.50;
Stocks["VW"]=412.50;
Stocks["BMW"]=834.00;
//Вывод всех элементов
StringFloatMap::iterator pos;
for(pos=stocks.begin();pos!=stocs.end();++pos) {
cout<<"stock"<<pos->first<<"\t";
cout<<"prise"<<pos->second<<endl;} }
//Переименование ключа
Stocs["Volkswagen"]=Stocs["BMW"];
Stocs.erase("BMW");
//Вывод всех элементов
for(pos=stocks.begin();pos!=stocs.end();++pos) {
cout<<"stock"<<pos->first<<"\t";
cout<<"prise"<<pos->second<<endl;}
}
}
```

Лабораторная работа №8

Тема: Алгоритмы STL.

Цель: Освоить технологию обобщенного программирования использованием библиотеки стандартных шаблонов (STL) языка C++.

I. Краткие теоретические сведения.

Алгоритмы

Каждый алгоритм выражается шаблоном функции или набором шаблонов функций. Таким образом, алгоритм может работать с очень разными контейнерами, содержащими значения разнообразных типов. Алгоритмы, которые возвращают итератор, как правило, для сообщения о неудаче используют конец входной последовательности. Алгоритмы не выполняют проверки диапазона на их входе и выходе. Когда алгоритм возвращает итератор, это будет итератор того же типа, что и был на входе. Алгоритмы реализуют большинство распространенных универсальных операций с контейнерами, такие как просмотр, сортировка, поиск, вставка и удаление элементов. Аргументами всех алгоритмов являются полуинтервалы [beg, end).

Алгоритмы определены в заголовочном файле <algorithm.h>.

Ниже приведены имена некоторых наиболее часто используемых функций алгоритмов STL.

1. Алгоритмы не изменяющие элементы интервала.

for_earch() выполняет операции для каждого элемента интервала
find() находит первое вхождение значения в интервал
find_if() находит первое соответствие предикату в интервале
count() подсчитывает количество вхождений значения в последовательность
count_if() подсчитывает количество выполнений предиката в интервале
min_element() наименьшее значение в интервале
max_element() наибольшее значение в интервале

2. Алгоритмы замены элементов в интервале

fill() заменяет все элементы данным значением
replace() заменяет элементы с указанным значением
replace_if() заменяет элементы при выполнении предиката

3. Алгоритмы удаления элементов интервала

remove() удаляет элементы с данным значением
remove_if() удаляет элементы при выполнении предиката
unique() удаляет равные соседние элементы

4. Алгоритмы перестановки элементов интервала

reverse() меняет порядок следования элементов на обратный

random_shuffle() перемещает элементы согласно случайному равномерному распределению ("тасует" последовательность)

rotate() производит циклический сдвиг элементов

partition() изменяет порядок элементов так что элементы отвечающие критерии оказываются впереди

partition_stable() тоже что и partition(), но с сохранением порядка элементов отвечающих и не отвечающих критерию

next_permutation() следующая перестановка в лексикографическом порядке

prev_permutation() предыдущая перестановка в лексикографическом порядке

5. Алгоритмы сортировки элементов интервала.

sort() сортирует интервал

partial_sort() сортирует часть интервала

stable_sort() сортирует интервал, сохраняя порядок следования равных элементов

6. Алгоритмы копирования элементов интервала в другой интервал.

copy() копирует интервал, начиная с первого элемента

copy_bacwards() копирует интервал, начиная с последнего элемента

replace_copy() копирует интервал, заменяя элементы с указанным значением

replace_copy_if() копирует интервал, заменяя элементы при выполнении предиката

remove_copy() копирует интервал, удаляя элементы с указанным значением

remove_copy_if() копирует интервал, удаляя элементы при выполнении предиката

unique_copy() копирует интервал, удаляя равные соседние элементы

rotate_copy() производит циклический сдвиг элементов при копировании

7. Алгоритмы сравнения двух интервалов

search() находит первое вхождение интервала

find_end() находит последнее вхождение интервала

equal проверяет равны ли два интервала

mismatch возвращает первый различающийся элемент в двух интервалах

lexicographical_compare() Лексикографическое сравнение двух интервалов

8. Модифицирующие алгоритмы для двух интервалов

transform() модифицирует (и копирует) элементы, объединяет элементы двух интервалов

swap_ranges меняет местами два интервала

9. Алгоритмы поиска в упорядоченном интервале

lower_bound() находит первое вхождение значения в отсортированном интервале

upper_bound() находит первый элемент, больший чем заданное значение

binary_search() определяет, есть ли данный элемент в отсортированном интервале

10. Алгоритм сравнения двух упорядоченных интервалов

includes() проверка на вхождение одного интервала в другое

11. Алгоритмы над двумя упорядоченными интервалами.

set_union() объединение интервалов

set_intersection() пересечение интервалов

set_difference() разность интервалов

set_symmetric_difference() симметрическая разность интервалов

merge() слияние двух интервалов

12. Численные алгоритмы.

accumulate() объединяет все значения элементов(вычисляет сумму, произведение и.т.д.)

inner_product() объединяет все элементы двух интервалов(вычисляет скалярное произведение и.т.д.)

adjacent_difference() объединяет каждый элемент с его предшественником (вычисляет разность и.т.д.)

partial_sum() объединяет каждый элемент со всеми предшественниками (вычисляет все частичные суммы и.т.д.)

II. Постановка задачи

1. Написать и реализовать программу, которая демонстрирует использование контейнерных классов и немодифицирующих алгоритмов STL.

В программе выполнить следующее:

a) Создать контейнер, содержащий объекты пользовательского типа.

b) Просмотреть контейнер используя алгоритм **for_each()**.

c) Найти в контейнере элемент, удовлетворяющий заданному условию.

d) Подсчитать, количество элементов удовлетворяющих заданному условию.

e) Создать второй контейнер этого же класса содержащий объекты пользовательского типа.

f) Просмотреть контейнер используя алгоритм **for_each()**.

g) Найти в контейнере максимальный элемент, удовлетворяющий заданному условию сравнения.

h) Сравнить два интервала и если они не равны найти первый различающийся элемент

2. Написать и реализовать программу, которая демонстрирует использование контейнерных классов и модифицирующих алгоритмов STL.

В программе выполнить следующее:

a) Создать контейнер, содержащий объекты пользовательского типа. Использовать контейнер второго типа указанного в варианте.

b) Просмотреть контейнер.

c) Переместить элементы, удовлетворяющие заданному условию в другой (предварительно пустой) контейнер.

d) Изменить второй контейнер, удалив из него одни элементы и заменив другие отвечающие заданным условиям.

e) Просмотреть второй контейнер.

f) Отсортировать обе контейнера и удалить равные соседние элементы.

g) Просмотреть оба контейнера.

h) Проверить вхождение первого контейнера во второй.

i) Создать третий контейнер путем объединения первых двух.

j) Просмотреть третий контейнер.

3. Создать собственную версию указанных в варианте алгоритмов и использовать их в программе. Также в программе использовать один из численных алгоритмов или алгоритм лексикографического сравнения связанных с пользовательским типом данных.

III. Варианты

	1	2	Класс	Алгоритм
1	priority_queue	list	РАБОЧИЙ	find() find_if()
2	Map	stack	ЖУРНАЛ	count() count_if()
3	multimap	queue	ТОВАР	min_element() max_element()
4	Set	vector	КВИТАНЦИЯ	copy() copy_bacwards()
5	multiset	deque	ЦЕХ	lower_bound() upper_bound()
6	priority_queue	list	ПЕРСОНА	unique() unique_copy()
7	map	stack	КОРАБЛЬ	rotate() rotate_copy()
8	multimap	queue	ГОРОД	remove_copy() remove_copy_if()

9	set	vector	КАРТИНА	replace_copy() replace_copy_if()
10	multiset	deque	ПОСТОЯЛЕЦ	replace() replace_if()
11	priority_queue	list	ТЕЛЕФОН	remove() remove_if()
12	map	stack	ФУТБОЛИСТ	partial_sort() stable_sort()
13	multimap	queue	ДИСК	partition() partition_stable()
14	set	vector	УЧЕНИК	random_shuffle() reverse()
15	multiset	deque	ДИСЦИПЛИНА	equal() mismatch
16	priority_queue	list	БЛЮДО	search() find_end()
17	map	stack	МАРШРУТ	fill() sort()
18	multimap	queue	ЭКЗАМЕН	binary_search() lexicographical_ compare()
19	set	vector	АДРЕС	next_permutation() prev_permutation()
20	multiset	deque	СЛУЖАЩИЙ	transform() swap_ranges
21	priority_queue	list	СТРАНА	includes() merge()
22	map	stack	ПРЕПОДАВА-ТЕЛЬ	set_difference() set_symmetric_ difference()
23	multimap	queue	ВОЕННЫЙ	set_union() set_intersection()
24	set	vector	ЖИВОТНОЕ	accumulate() inner_product()
25	multiset	deque	ДЕТАЛЬ	adjacent_ difference() partial_sum()

IV. Содержание отчета

1. Титульный лист: название дисциплины, номер и наименование работы, фамилия, имя, отчество студента, дата выполнения.

2. Постановка задачи.

3. Определение пользовательского класса.

4. Определения используемых в программах компонентных функций для работы с контейнером, включая конструкторы.

5. Определения и объяснения используемых в программах алгоритмов STL.

6. Определения и объяснения, используемых предикатов и функций сравнения.

V. Методические указания

1. Для сравнения элементов при сортировке следует написать функцию **comp** и использовать вторую версию алгоритма **sort.**
2. Условия поиска и замены элементов выбираются самостоятельно и для них пишется функция-предикат.
3. Для ввода-вывода объектов пользовательского класса следует перегрузить операции “>>” и “<<”.
4. Некоторые алгоритмы могут не поддерживать используемые в программе контейнеры. В этом случае следует написать свой алгоритм.
5. Примеры применения некоторых наиболее часто используемых немодифицирующих функций-алгоритмов.

Использование алгоритма **for_each()**:

```
# include <iostream.h>
# include<vector>
# include<algorithm>
using namespace std;
class IntPrint
{public:
void operator() (int elem){
 cout<<elem<<endl; }
};
main()
{
  vector<int> coll(5,4);
   IntPrint cmp;
   for_each(coll.begin(), coll.end(), cmp);
   cout<<num;
}
```

Использование алгоритма **find_if()**:

```
# include <iostream.h>
# include<vector>
# include<algorithm>
using namespace std;
class Greater
```

```
{int k;
public:
Greater(int m){k=m};
bool operator() (int elem){
 return elem>k;
 }
};
main()
{
  vector<int> coll(5,4);
   vector<int>::iterator p;
   Greater cmp(5);
   p=find(coll.begin(), coll.end(), cmp);
   if (p=coll.end) cout<<"элемент не найден"; else cout<<"элемент
найден";
}
```

Использование алгоритма **count_if()**:

```
# include <iostream.h>
# include<vector>
# include<algorithm>
using namespace std;
class IsEven
{public:
bool operator() (int elem){
 return elem %2==0;
 }
};
main()
{
  vector<int> coll(5,4);
   IsEven cmp;
   int num=count(coll.begin(), coll.end(), cmp);
   cout<<num;
}
```

Использование алгоритма **max_element()**:

```
# include <iostream.h>
# include<vector>
# include<algorithm>
using namespace std;
class CompInt
{public:
bool lessthen (int elem1,int elem2){
```

```
  return abs(elem1)<abs(elem2);
  }
};
main()
{
  vector<int> coll1(5,4);
  CompInt cmp;
   vector<int>::iterator p;
   p=max_element(coll.begin(), coll.end(),cmp);
   p.print();
}
```

6. Примеры применения некоторых наиболее часто используемых модифицирующих функций-алгоритмов.

Использование алгоритма **copy ()**:

```
# include <iostream.h>
# include<vector>
# include<list>
# include<algorithm>
using namespace std;

main()
{
  vector<int> coll1(5,4);
  list <int> coll2;
  copy (coll11.begin(), coll2.end(), back_inserter(coll2));
  copy (coll11.begin(), coll2.end(), ostream_iterator<int>(cout," "));
}
```

Использование алгоритма **replace** и **replace_if()**:

```
# include <iostream.h>
# include<vector>
# include<algorithm>
using namespace std;
class IsEven
{public:
bool operator() (int elem){
 return elem %2==0;
 }
};
main()
{
  vector<int> coll;
 for(int i=0;i<6; i++) coll1.push_back(i);
 IsEven cmp;
```

```
replace (coll.begin(), coll.end(), 5, 42);
replace_if (coll.begin(), coll.end(), cmp,5);
  for(pos=coll.begin();pos!=coll.end();++pos) cout<<*pos<<" ";
}
```

Использование алгоритма **fill ()**:

```
# include <iostream.h>
# include<vector>
# include<algorithm>
using namespace std;

main()
{
  vector<int> coll;
  fill_n (back_inserter(coll),9,"hello");
  fill (coll1.begin(), coll.end(),"again");
  vector<int>::iterator pos;
  for(pos=coll.begin();pos!=coll.end();++pos) cout<<*pos<<" ";
}
```

7. Примерная реализация алгоритма For_each

```
Namespace std {
Template<class Iterator, class Operation>
Operation For_each(Iterator beg, Iterator end, Operation op)
{
  while (beg!=end) {
   op(*beg);
   ++beg;
   }
   return op;
 }
 }
```

СПИСОК ЛИТЕРАТУРЫ

Основная литература:

1. Гради Буч. Объектно –ориентированной анализ и проектирование с примерами приложений на С++. Невский диалект, 2001 г., 560 с.,
2. Грехем И. Объектно ориентированные методы. Принципы и практика. Вильямс., 2004, 879 с.,
3. Иванова Г.С. Объектно ориентированное программирование. Учебник., МГТУ им Баумана, 2003, 320 с.
4. Ашарина Н.А. Основы программирования на языках Си, С++. Учебный курс., М.: 2002
5. Шмидский Я.К. Прораммирование на языке С++: Самоучитель. Учебное пособие., Диалектика, 2004 г., 361 с.
6. Страуструп Б. Язык программирования С++. Третье издание, М.: Бином, 1999
7. Пол Айра. Объектно-ориентированное программирование на С++. Второе издание. – М.: Бином, 1999.
8. Аммераль Л. STL для программистов на С++., М: ДМК, 1999.

Дополнительная литература:

1. Крупник А.Б. Изучаем С++. Питер. 2003, 251 с.
2. Мейерс С. Наиболее эффективное использование С++. 35 новых рекомендаций. ДМК-Пресс, 2000, 304 с.
3. Николаенко Д.В. Самоучитель по Visual С++., Спб, 2001.
4. Элджер Дж. С++: библиотека программиста, СПб: Питер, 1999.
5. Либерти Д. Освой самостоятельно С++: 10 минут на урок. Пер с англ. Вильямс, 2004, 374 с.
6. Шилдт Г. Самоучитель С++. Второе издание, СПб.: BHV, 1998.
7. Луис Д. С и С++. Справочник., М: Бином, 1997.
8. Подбельский В.В. Язык С++ – М.: Финансы и статистика, 1996.
9. Фейсон Т. Объектно-ориентированное программирование на С++ 4.5. – Киев: Диалектика, 1996.
10. Шилдт Г. Теория и практика С++., СПб.: BHV, 1996.
11. Керниган Б., Ритчи Д. Язык программирования Си. М. Финансы и статистика. 1985.

Приложение 1

1) В двух ключевых методах к буквам расположенным на четных и нечетных позициях применяются два ключа K1 и K2.

3) Подстановка P означает перестановку алфавита, при котором ни одна буква не остаётся на своём месте. Подстановка представляется в программе виде кодов букв.

4) В много ключевых методах применяется набор ключей. В методе Вижинера ключи представляются в виде случайного массива чисел.

5) В методе Вернама слово переводится в двоичный код. Ключи представляются в виде случайного массива битов.

6) В рекуррентных методах ключ вычисляется на каждом шаге. В методе Фон Неймана на каждом шаге предыдущее значение возводится в квадрат и цифры самого старшего и младшего разряда отбрасываются.

7) В методах перестановки буквы не меняются, а переставляются. В четной перестановке используются два четных ключа. Длина строки должна быть четной, иначе добавляется специальный символ.

Одноключевые методы **I код исходной буквы J код шифр буквы N объем алфавита**		
1	Цезаря	J=(I+K) mod N
2	Аффинный	J=(A*I+K) mod N; A не должен иметь общих делителей с N
3	С XOR	J=I^K, K<256, N=256
4	Подстановки	J=P[I]
Многоключевые методы		
5	Двух ключевая	Если НОМЕР(I) четно то J=(I+K1) mod N, иначе J=(I+K2) mod N
6	Вижинера	J=(I+M[L]) mod N; L=НОМЕР(I) mod ДЛИНА(M)
7	Вернама	J=I^M[L]; L=НОМЕР(I) mod ДЛИНА(M)
8	Паскаля	J=(I+C[K,L]) mod N; L=НОМЕР(I) mod K
Рекуррентные методы J=(I+ K_L) mod N		
9	Стандартный	RND(K_0); K_L = 100*RND();
10	Линейный	K_L = (A*K_{L-1}+B) mod N;
11	Квадратичный	K_L = (A*K_{L-1}*K_{L-1}+B *K_{L-1}+C) mod N;
12	Фон Неймана	K_L=УСЕЧЕНИЕ(K_{L-1} *K_{L-1});
Методы перестановки **i номер исходной буквы в тексте j номер той же буквы в шифр тексте, N длина текста**		
13	Простая	j=(i+K) mod N
14	Зеркальная	j=(N-i+K) mod N
15	Четная	Если i четно то j=(i+K1) mod N , иначе j=(i+K2) mod N

Приложение 2

Квадратурные формулы	
Прямо-угольников	$S=h*(f_0+ f_1+\ldots+ f_{n-1})$; $h=(b-a)/n$
Симпсона	$S=(h/3)*(f_0+ f_{2*m}+4*(f_1+f_3+\ldots+f_{2*m-1})+ 2*(f_2+f_4+\ldots+f_{2*m-2}))$; $h=(b-a)/2*m$
Ньютона	$A=(3*h/8)*(f_0+ f_3+3*f_1 +3*f_2)$; $h=(b-a)/(2*m+1)$;
Квадратурные формулы высших порядков	
Четырех полосная	$A=(2*h/45)*(7*f_0+ 32*f_1+12*f_2+32*f_3+7*f_4)$
Пяти полосная	$A=(5*h/288)*(19*f_0+ 75*f_1+50*f_2 +50*f_3 +75*f_4+19*f_5)$
Шести полосная	$A=(6*h/840)*(41*f_0+216*f_1+27*f_2+272*f_3+27*f_4+216*f_5+27*f_6)$
Уэддла	$A=(3*h/10)*(f_0+5*f_1+f_2+6*f_3+f_4+5*f_5+f_6)$
Квадратурные формулы без аппроксимации области	
Гаусса	$S= (0.5*(b-a))*(A_1*f(x_1)+ A_2*f(x_2)+\ldots+ A_n*f(x_n))$ $x_i=0.5*(b+a)+0.5*(b-a)*t_i$
Ньютона - Котеса	$S=((b-a)/n)*(H_0*f_0+ H_1*f_1+\ldots+ H_n*f_n)$
Чебышева	$S=((b-a)/n)* (f(x_1)+ f(x_2)+\ldots+ f(x_n))$; $x_i=0.5*(b+a)+0.5*(b-a)*t_i$
Монте Карло	$S=(1/n)* (\Sigma f(x_i))$; x_i случайные точки лежащие в области; n общее количество точек
Методы решения нелинейных уравнений	
Метод простой итерации	Уравнение $f(x)=0$ приводится к виду $x=\varphi(x)$. Рекуррентное соотношение $x_{i+1}=\varphi(x_i)$
Метод хорд	$x_{i+1}= x_i-((f(x_i)*(x_i-c))/(f(x_i)- f(c))$; Если $f(a)*f''(a)>0$ то $c=a$; Если $f(b)*f''(b)>0$ то $c=b$;
Метод пошагового спуска	Если $f(a)>0$ то $x_0=a$ и $h=(b-a)/2$; Если $f(b)>0$ то $x_0=b$ и $h= -(b-a)/2$; $x_{i+1}=x_i+ h$; Если не выполнены условия $f(x_{i+1}) < f(x_i)$ и $f(x_{i+1}) >0$ то $h=h/2$ и вычисление повторяется
Метод секущих	$x_{i+1}=x_i- (x_i- x_{i-1})*f(x_i) / (f(x_i)- f(x_{i-1}))$
Разностный метод Ньютона	$x_{i+1}=x_i- h*(f(x_i))/ (f(x_i) -f(x_i-h))$
Метод Ньютона	$x_{i+1}=x_i- f(x_i) / f'(x_i)$
Метод Ньютона – Рафсона	$x_{i+1}=x_i- \beta_k*f'(x_i) / f''(x_i)$ Если $f(x_{i+1})> f(x_i)$ то $\beta_k= \beta_k/2$ и вычисление повторяется
Методы решения дифференциальных уравнений	
Метод Эйлера	$y_{k+1}=y_k+h*f(x_k,y_k)$
Усовершенствованный метод Эйлера	Сначала вычисляется $y_{k+1/2}=y_k+h*f(x_k,y_k)$; Затем $y_{k+1}=y_k+h*f(x_{k+1/2},y_{k+1/2})$

Усовершенствованный метод Эйлера-Коши	Сначала вычисляется $z_{k+1}=y_k+h*f(x_k,y_k)$; Затем $y_{k+1}=y_k+h*(f(x_{k+1},z_{k+1})+f(x_k,y_k))/2$
Метод Рунге Кутта третьего порядка	Сначала вычисляются числа $k_1=h*f(x_i,y_i,z_i)$; $k_2=h*f(x_i+h/2, y_i+h*k_1/2)$; $k_3=h*f(x_i+h/2, y_i-h*k_1+2*h*k_2)$; Затем $y_{i+1}=y_i+(h/6)*(k_1+4*k_2+k_3)$;
Метод Рунге Кутта четвертого порядка	Сначала вычисляются числа $k_1=h*f(x_i,y_i,z_i)$; $k_2=h*f(x_i+h/2, y_i+h*k_1/2)$; $k_3=h*f(x_i+h/2, y_i+h*k_2/2)$; $k_4=h*f(x_i+h, y_i+h*k_3)$; Затем $y_{i+1}=y_i+h*(k_1+2*k_2+2*k_3+k_4)$;
Адамса третьего порядка	$y_n=y_{n-1}+(h/12)*(23*f_{n-1}-16*f_{n-2}+5*f_{n-3})$
Адамса четвертого порядка	$y_n=y_{n-1}+(h/24)*(55*f_{n-1}-59*f_{n-2}+37*f_{n-3}-9*f_{n-4})$
Методы минимизации функции от одной переменной	
Метод пошагового спуска	$f(x_i) > f(x_i+h)$ $x_{i+1}=x_i+h$; $f(x_i) > f(x_i-h)$ $x_{i-1}=x_i-h$; Если ни одно из этих условий не выполнено то $h=h/2$ и вычисление повторяется
Метод дихотомии	$c=(a_i+b_i)/2$; $x_1=c-h$; $x_2=c+h$; $f(x_1) > f(x_2)$ $a_{i+1}=x_1$; $b_{i+1}=b_{i+1}$; $f(x_1) < f(x_2)$ $a_{i+1}=a_i$; $b_{i+1}=x_2$; $f(x_1) = f(x_2)$ $a_{i+1}=x_1$; $b_{i+1}=x_2$;
Метод золотого сечения	$x_1=a_i+(3-\sqrt{5})*(b_i-a_i)/2$; $x_2=a_i+(\sqrt{5}-1)*(b_i-a_i)/2$; $f(x_1) > f(x_2)$ $a_{i+1}=x_1$; $b_{i+1}=b_{i+1}$; $f(x_1) < f(x_2)$ $a_{i+1}=a_i$; $b_{i+1}=x_2$; $f(x_1) = f(x_2)$ $a_{i+1}=x_1$; $b_{i+1}=x_2$;
Метод Фибоначчи	$x_1=a_i+(b_i-a_i)*F_{n-1}/F_n$; $x_2=a_i+(b_i-a_i)*F_n/F_{n+1}$; $f(x_1) > f(x_2)$ $a_{i+1}=x_1$; $b_{i+1}=b_{i+1}$; $f(x_1) < f(x_2)$ $a_{i+1}=a_i$; $b_{i+1}=x_2$; $f(x_1) = f(x_2)$ $a_{i+1}=x_1$; $b_{i+1}=x_2$;
Разностный метод Ньютона	$x_{i+1}=x_i-0.5*h*(f(x_i+h)-f(x_i-h))/(f(x_i+h)-2*f(x_i)+f(x_i-h))$
Метод парабол	$x^*=x_2-0.5*((x_2-x_1)^2*(f(x_2)-f(x_3)-(x_3-x_2)^2*(f(x_2)-f(x_1)))/((x_2-x_1)*(f(x_2)-f(x_3)-(x_3-x_2)*(f(x_2)-f(x_1)))$ из точек x_1,x_2,x_3 ,x^*- выбираем удачную тройку
Метод Ньютона	$x_{i+1}=x_i-f'(x_i)/f''(x_i)$
Метод касательных	$a_0=a$; $b_0=b$; $x^*=(f(a_i)-f(b_i)+b_i*f'(a_i)-a_i*f'(b_i))/(f'(b_i)-f'(a_i)$ Если $f'(x^*)>0$ то $a_{i+1}=a_i$; $b_{i+1}=x^*$; Если $f'(x^*)<0$ то $a_{i+1}=x^*$; $b_{i+1}=b_i$;
Метод секущих	$x_{i+1}=x_i-(x_i-x_{i-1})*f'(x_i)/(f'(x_i)-f'(x_{i-1}))$
Метод Ньютона - Рафсона	$x_{i+1}=x_i-\beta_k*f(x_i)/f''(x_i)$ Если $f(x_{i+1})>f(x_i)$ то $\beta_k=\beta_k/2$ и вычисление повторяется

ОГЛАВЛЕНИЕ

Методическое пособие по лабораторным занятиям дисциплины «Объектно – **ориентированные языки программирования»**

Разработка рассмотрена на заседании кафедры «ТП»
и рекомендована к печати
(протокол № от 2008 года)

Авторы:

Зав. каф. Назиров Ш.А.
Доц.каф. Кабулов Р.В.
Старший преп. Арипова Н.А.

Ответственный редактор — Проректор по учебной работе ТУИТ, д.т.н., проф. Каримов М.М.

Корректор — Т.М.Яковлева

Бумага офсетная. Заказ № 188
Тираж. 50
Отпечатано в типографии ТУИТ
Ташкент 700084, ул.А.Тимура – 108