Федеральное агентство по образованию
Уральский государственный технический университет – УПИ
имени первого Президента России Б. Н. Ельцина

ЯЗЫК ПРОГРАММИРОВАНИЯ С++ В ЗАДАЧАХ

Методические указания к лабораторному практикуму
по дисциплине «Системное программирование»
для студентов направления
230100 – Информатика и вычислительная техника

Екатеринбург
УГТУ – УПИ
2009

УДК 658.1 (078)

Составитель О. Л. Чагаева

Научный редактор С. И. Тимошенко, канд. техн. наук

ЯЗЫК ПРОГРАММИРОВАНИЯ С++ В ЗАДАЧАХ : методические указания к лабораторному практикуму по дисциплине «Системное программирование» / сост. О. Л. Чагаева. – Екатеринбург : УГТУ – УПИ, 2009. – 40 с.

Методические указания содержат задания на лабораторные работы по дисциплине «Системное программирование». В работе последовательно разбираются **вопросы, возникающие в ходе выполнения** заданий. Задания расположены **по нарастающей сложности, приведены примеры** реализации задач **на языке программирования С++, что соответствует** задачам подготовки программистов. **Приведены необходимые** справочные материалы и примеры программ. **Рекомендуется студентам** всех специальностей направления **230100 «Информатика и вычислительная** техника».

Библиогр.: 11 назв. Рис. 2.

Подготовлены кафедрой
«Программные средства и системы»
факультета ускоренного обучения

ВВЕДЕНИЕ

Язык программирования С++ был развит из языка программирования С и за очень немногим исключением сохраняет его как подмножество. Это означает, что программы, написанные на С, могут быть легко перекомпилированы как программы С++. Однако, следует иметь в виду, что некоторые конструкции С и С++ обрабатываются по-разному. Поэтому, может возникнуть ситуация, когда существующей программе на языке С потребуются небольшие изменения. С другой стороны, С++ предоставляет ряд дополнительных возможностей.

Первоначально язык программирования С задумывался как язык, который должен заменить программирование на ассемблере в самых насущных задачах системного программирования. Язык С++ унаследовал эти свойства. Различие между ними состоит в степени внимания, уделяемого типам и структурам. В отличие от С, С++ требует от программиста большего внимания к типам объектов.

В большинстве разработок имеются понятия, которые трудно представить в программе в виде одного из основных типов или как функцию без ассоциированных в ней статических данных. С точки зрения С++, подобное понятие разумно описать как класс, представляющий его в программе. Класс – это тип, то есть он задает поведение объектов его класса, определяет способы их создания и удаления, приемы работы с ними.

Класс является фундаментальным механизмом, с введением которого язык С приобретает объектно-ориентированные черты и становится С++. Класс есть расширение понятия "структура" языка С. Он позволяет создавать типы и определять функции, которые задают поведение типа. Каждый представитель класса называется объектом.

Предназначение понятия "класс" состоит в том, чтобы предоставить программисту инструмент для создания новых типов, столь же удобных в обращении, сколь и встроенные типы. В идеале тип, определяемый пользователем, должен отличаться от встроенных типов не способом использования, а способом создания.

Тип есть конкретное представление некоторого понятия. Новый тип создается для того, чтобы дать специальное и конкретное определение понятия, которому ничто прямо и очевидно среди встроенных типов не отвечает.

В определении нового типа основная идея - отделить несущественные подробности реализации от тех качеств, которые существенны для его правильного использования.

Обычно, для преодоления сложности программ, используется прием иерархического упорядочения, т.е. организация связанных между

собой понятий в древовидную структуру с самым общим понятием в корне. В C++ такие структуры представляются производными классами. Часто можно организовать программу как множество деревьев, т.е. программист задает множество базовых классов, каждый из которых имеет свое собственное представление производных классов. Для определения набора действий для самой общей интерпретации понятия часто можно использовать виртуальные функции. Для представления множеств взаимозависимых классов в C++ можно использовать идею friend-классов.

1. ЛАБОРАТОРНАЯ РАБОТА 1

Цель: написание первой программы на языке C++;
знакомство с операциями над указателями;
знакомство с правилами передачи параметров в функцию

1. Ознакомиться со структурой программы на языке C++. Для этого выполнить программу возведения числа 2 в степень. Модифицировать программу так, чтобы вычислялась степень с любым основанием.

```
#include <iostream.h>
void main(void)
{
  int i, k;
  int power(int, int);
  for(i=0; i<10; i++)
  {
    k=power(2, i);
    cout<<"i = "<<i<<", k = "<<k<<endl;
  }
}

int power(int x, int n)
{
  int i, p=1;
  for(i=1; i<n; i++) p*=x;
  return p;
}
```


2. Выполнить программу, иллюстрирующую разные операции с указателями. Повторить эксперимент для разных типов данных.

```
#define PR(x) printf("x=%u, *x=%d, &x=%u\n",x,*x,&x)
#include <stdio.h>
void main(void)
{
  int mas[]={100, 200, 300};
  int *ptr1, *ptr2;
  ptr1=mas;
  ptr2=&mas[2];
  PR(ptr1);
  ptr1++;
  PR(ptr1);
  PR(ptr2);
  ++ptr2;
  printf("ptr2-ptr1=%u\n",ptr2-ptr1);
}
```

3. Написать программу, которая меняет местами значения двух переменных. В качестве параметра использовать int, int* и int&.

Структура программы следующая: функция main() вводит переменные и вызывает другую функцию, которая и выполняет обмен:

```
#include <iostream.h>
void main(void)
{
  int a=2, b=5;
  void obmen1(int, int);
  void obmen2(int*, int*);
  void obmen3(int&, int&);
  cout << "до обмена:   a= "<< a <<"b="<<b<< endl;
  //вызов obmen1(int, int);
  cout << "после обмена 1:  a= "<< a <<"b="<<b<< endl;
  //вызов obmen2(int*, int*);
  cout << "после обмена 2:  a= "<< a <<"b="<<b<< endl;
  //вызов obmen3(int&, int&);
  cout << "после обмена 3:  a= "<< a <<"b="<<b<< endl;
}
```

Написать 3 варианта функции обмена. В качестве параметров эта функция в первый раз получает два параметра типа int, второй раз – типа int*, третий раз – типа int&. Придумайте способ достижения результата в каждом из трех вариантов.

Отчет по лабораторной работе 1 должен включать:

- блок-схемы всех программ по пунктам;
- тексты программ с подробными комментариями;
- выводы по проделанным экспериментам.

2. ЛАБОРАТОРНАЯ РАБОТА 2

Цель: ***приобретение навыков работы с матричным представлением данных;***

обработка матриц.

Используя заготовку для ввода матрицы, написать следующие функции:

- минимум матрицы;
- максимум матрицы;
- максимум нижнетреугольной части матрицы;
- максимум верхнетреугольной части матрицы;
- минимум нижнетреугольной части матрицы;
- минимум верхнетреугольной части матрицы;
- минимум главной диагонали матрицы;
- максимум главной диагонали матрицы;
- минимум второстепенной диагонали матрицы;
- максимум второстепенной диагонали матрицы;
- среднеарифметическое значение элементов матрицы;
- среднеарифметическое значение элементов нижнетреугольной части матрицы;
- среднеарифметическое значение элементов верхнетреугольной части матрицы;
- суммы строк матрицы;
- суммы столбцов матрицы;
- минимальные значения строк;
- минимальные значения столбцов;

- максимальные значения строк;
- максимальные значения столбцов;
- среднеарифметические значения строк;
- среднеарифметические значения столбцов;
- суммы нижне- и верхнетреугольных частей матрицы;
- элемент, наиболее близкий по значению к среднеарифметическому.

Заготовочная функция для матричных операций:

```
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <conio.h>
#define N 5
void main()
{
  float m[N][N];
  int i,j;
  for(i=0;i<N;i++)
      for(j=0;j<N;j++) m[i][j]=rand()/10.;
  for(i=0; i<N; i++)
  {
      for(j=0 ; j<N;j++)
           cout<<setw(8)<<setprecision(5)<<m[i][j];
      cout << endl;
  }
  getch();
}
```

В "заготовочной" функции используются некоторые функции управления выводом в стандартный поток *cout* (объявлены в файле *iomanip.h*):

- *setw(int)* – задание общей длины поля (позиций). Аргумент – целое значение;
- *setprecision(int)* – задание точности вывода (количество выводимых символов). Для данных вещественных типов точность определяется суммарным количеством цифр в целой и дробной частях. Если точность превосходит длину поля, длина поля игнорируется и устанавливается как увеличенная на единицу (для разделительной точки) точность. Если точность не позволяет вывести всю дробную часть числа, она округляется до заданного количества символов. В

случае, когда в отведенных позициях не удается разместить даже целую часть числа, значение выводится в экспоненциальной форме.

2. Отчет по лабораторной работе 2 должен включать:
   - блок-схемы алгоритмов;
   - тексты всех функций с подробными комментариями;
   - выводы о проделанных экспериментах.

3. ЛАБОРАТОРНАЯ РАБОТА 3

Цель: освоение основных методов упорядочения числовых данных;

знакомство с реализацией различных алгоритмов сортировки средствами языка С++

1. Разобраться, как работает предложенная программа сортировки массива методом «мини-макса».

2. Написать программы, сортирующие одномерный массив методами «пузырька» и «быстрой сортировки».

Можно ли такие функции использовать для работы с многомерным массивом?

3. Для массива целых значений выполнить сортировку по возрастанию четных и по убыванию нечетных значений.

4. Отсортировать массив по возрастанию на интервале индексов от N1 до N2.

5. Отсортировать массив по убыванию на интервале индексов от N1 до N2.

Функция сортировки массива методом "мини-макса":

```
#include <iostream.h>
#include <conio.h>
void main(void)
{
  int mas[]={2,5,-8,1,-4,6,3,-5,-9,13,0,4,9};
  //текущие индексы мин. и макс. элементов
  int imin, imax;
  //вычисление n - количества элементов
  int n=sizeof(mas)/sizeof(int);
  int i;
  //установка начальных индексов для мин и макс
  imin=i=0; imax=i=0;
```



```
  clrscr();
  for(i=0; i<n; i++)
    cout << mas[i] << ' ';
  cout << endl;
  for(i=0; i<n-1;i++)          //нахождение мин и макс
  {
     imin=i;
     for(int j=i+1; j<n; j++)
       if(mas[j] < mas[imin]) imin = j;
     int t=mas[i];
     mas[i]=mas[imin];
     mas[imin]=t;
  }
  for(i=0; i<n; i++)          //вывод на экран
    cout << mas[i] << ' ';
  cout << endl;
}
```

Отчет по лабораторной работе 3 должен включать:

- блок-схемы алгоритмов сортировки методами «мини-макса», «пузырька» и «быстрой сортировки»;
- тексты программ с подробными комментариями;
- выводы по проделанным экспериментам;
- ответы на вопросы.

4. ЛАБОРАТОРНАЯ РАБОТА 4

Цель: приобретение навыков обработки строковых данных

1. Познакомиться с библиотечными функциями работы со строками. Написать программу, которая использует эти функции (для работы с ними подключить файл string.h):
   - strlen() – определение дины строки;
   - strcat() – конкатенация строк;
   - strcpy() – копирование строк;
   - strcmp() – сравнение строк.

2. Написать свои варианты функций:
   - определения длины строки (выполнить работу тремя способами);
   - копирования строк;
   - сравнения строк;
   - конкатенации строк.

Структура программ следующая: функция main() объявляет строки и вызывает другую функцию, которая и выполняет работу с переданными ей параметрами:

```
#include <iostream.h>
void main(void)
{
  char str1[]="qwerty", str2[]="1234567890";
  int dlina1(char*);
  int dlina2(char*);
  int dlina3(char*);
  void kopir(char*, char*);
  void sravn(char*, char*);
  void konkat(char*, char*);
  // использование функций длины строки
  cout<<"длина str1="<<dlina1(str1)<<",str2=";
  cout<< dlina1(str2)<< endl;
  cout<<"длина str1="<<dlina2(str1)<<",  str2=";
  cout<< dlina2(str2)<< endl;
  cout<<"длина str1= "<<dlina3(str1)<<",  str2=";
  cout<< dlina3(str2)<< endl;
  cout<< "после обмена 1:   a= "<< a <<"b="<<b<< endl;
  //вызов kopir(char*, char*);
  cout<< "результат копирования:str1= "<< str1;
  cout<<" , str2="<<str2<< endl;
  //вызов sravn(char*, char*);
  cout<< "результат сравнения:str1= "<< str1;
  cout<<" , str2="<<str2<< endl;
  //вызов konkat(char*, char*);
  cout<<"результат сцепления:str1= "<< str1;
  cout<<" , str2="<<str2<< endl;
}
```


3. Переписать функции так, чтобы они использовали динамическую память при задании строк:

```
#include <iostream.h>
#include <alloc.h>
void main(void)
{
  char *str1, *str2;
  str1=(char*)malloc(100);
  str2=(char*)malloc(100);
......
  free(str1);
  free(str2);
}
```

4. Изменить программу так, чтобы вместо malloc() использовалась функция calloc(). В чем сходство и различие этих функций?
5. В следующей программе организовать массив строк. Применить функции, написанные в предыдущем пункте к строкам, составляющим этот массив.

Отчет по лабораторной работе 4 должен включать:

- блок-схемы алгоритмов всех функций;
- тексты программ с подробными комментариями;
- выводы по проделанным экспериментам;
- ответы на вопросы.

5. ЛАБОРАТОРНАЯ РАБОТА 5

Цель: организация ввода-вывода;

освоение навыков работы с файлами средствами библиотеки функций ввода-вывода;

использование аргументов функции main.

1. Разобрать программу, выводящую содержимое текстового файла на экран. Как можно сократить текст программы?

```
#include <stdio.h>
void main(void)
{
   char ch, name[50];
   FILE *in;
   printf("Введите имя файла для просмотра: ");
   scanf("%s", name);
   if((in=fopen(name, "r"))==NULL)
    printf("Файл %s не открыт", name);
   else
      while(!feof(in))
      {
         ch=getc(in);
         putchar(ch);
      }
}
```

2. Дополнить программу предыдущего пункта функцией подсчета «пустых» и «непустых» символов в текстовом файле. Под «пустыми» символами понимаются символы, не отображающиеся на экране: управляющие символы, пробел, звуковой сигнал и т. д.
3. Дополнить программу первого пункта функцией поиска в текстовом файле заданной подстроки. Искомую подстроку необходимо передавать в качестве параметра функции. Результаты работы сформировать в отдельном файле.
4. Написать программу, которая обрабатывает текстовый файл следующим образом.

В исходном тексте номера страниц проставлены в первой строке страницы. Требуется перенести эти номера в последнюю строку страницы и убрать знаки переноса. Например, номер страницы: - 34 -. Необходимо получить 34.

Признаком перехода на следующую страницу является управляющий символ '\f'.

5. Написать программу-шифратор файлов. Она читает из файла и пишет в другой файл закодированные символы.

Схема шифровки: c = c^key[i], где c – символ, считанный из файла; key – ключ шифрования, строка, которая передается как параметр командной строки. Программа используется символы из key циклически, пока не будет считан весь ввод.

Для проверки правильности программы зашифровать зашифрованный файл еще раз с тем же ключом. Должен получиться файл – ТОЧНАЯ копия исходного файла.

6. Дополнить программу п. 5 возможностью вводить имя файла и ключ из командной строки.

Отчет по лабораторной работе 5 должен включать:

- блок-схемы алгоритмов всех функций;
- тексты программ с подробными комментариями;
- выводы по проделанным экспериментам;
- ответы на вопросы.

6. ЛАБОРАТОРНАЯ РАБОТА 6

Цель: знакомство с механизмом описания классов C++;

создание собственных типов данных.

1. Описать класс для работы со строкой, как со стандартным типом данных.
   - В разделе описания элементов-данных класс должен иметь переменную для хранения символов строки.
   - В разделе описания элементов-функций необходимо предусмотреть:
   - конструкторы в нескольких вариантах;
   - деструкторы, если необходимо;
   - перегруженные операции для выполнения присваивания (=); сцепления (+), сравнения (==);
   - методы для определения длины строки;
   - методы ввода и вывода строки.
2. Описание класса выделить в заголовочный файл, определение его методов – в файл с расширением *.cpp*, текст главной функции написать в отдельном файле.

Прототип программы:

```
class Stroka {
    char str[80];
  public:
    Stroka(char*);
    Stroka(){}
    Stroka(const Stroka&);
    Stroka& operator=(const Stroka&);
    Stroka& operator+(const Stroka&);
    int operator==(const Stroka&);
    int dlina();
    void vvod();
    void vyvod();
};

Stroka::Stroka(char *string)
{
    strcpy(str,string);
}
Stroka:: Stroka(const Stroka& s)
{
    strcpy(str, s.str);
}
Stroka& Stroka::operator=(const Stroka& s)
{
    strcpy(str, s.str);
    return *this;
}
Stroka& Stroka::operator+(const Stroka& s)
{
    strcat(str, s.str);
    return *this;
}
int Stroka::operator==(const Stroka& s)
{
    if ((strcmp(str, s.str))==0)
        return 1;
    else
        return 0;
}
int Stroka::dlina()
{
    return strlen(str);
```



```
}
void Stroka::vvod()
{
    cin >>  str;
}
void Stroka::vyvod()
{
    cout <<  str;
}
void main(void)
{
    Stroka s1("qwert"),  s3, s4(s1), s5;
    s3.vvod();
    s3="asdfg";
    s3.vyvod();
    s5= s1+s3+s4;
    cout<<"длина s5 = "<<s5.dlina();
    s5.vyvod();
    if(s1==s5)
        cout << "строки s1 и s5 равны";
    else
        if (s1==s4)
            cout << "строки s1 и s4 равны";
}
```

3. Изменить программу так, чтобы вместо библиотечных функций, выполняющих действия со строкой, использовались аналогичные функции, написанные при выполнении заданий лабораторной работы 4.
4. Дополнить описание класса конструктором, который создает экземпляр класса, используя заданный размер динамической памяти. Понадобится ли при наличии такого конструктора деструктор?

Отчет по лабораторной работе 6 должен включать:

- блок-схемы алгоритмов всех функций;
- тексты программ с подробными комментариями;
- выводы по проделанным экспериментам;
- ответы на вопросы.

7. ЛАБОРАТОРНАЯ РАБОТА 7

Цель: ***знакомство с механизмом наследования, виртуальными функциями, абстрактными классами.***

1. Предлагаемая программа выводит на экран 2 объекта, которые перемещаются по экрану при нажатии клавиши. Необходимо перевести эту программу с Паскаля на С++. Графические библиотеки языка сходны. При написании программы необходимо помнить, что все ключевые слова, названия библиотечных функций необходимо писать строчными буквами.

```
Uses Graph, Crt;
Type Point = Object
    X,Y:Integer;
    Cvet:Word;
    Procedure Init (XN, YN:Integer; Color: Word);
    Procedure Show; virtual;
    Procedure Hide; virtual;
    Procedure Locat (Var XL,YL:Integer);
    Procedure Fly (Cost:Integer);
  End;
Krug = Object(Point)
   Radius: Word;
   Constructor Init (XN, YN: Integer; R, Color: Word);
   Procedure Show; virtual;
   Procedure Hide; virtual;
End;
Ring = Object(Krug)
 Width: Word;
 Constructor CRing;
 Procedure Init (XN, YN:Integer; R, Color, Wid: Word);
 Procedure Show; virtual;
End;
  Procedure Point.Init;
  Begin
    X:= XN; Y:= YN; Cvet:= Color
  End;
  Procedure Point.Show;Begin  End;
  Procedure Point.Hide;Begin  End;
  Procedure Point.Locat;
  Begin
    XL:= X; YL:= Y
  End;
  Procedure Point.Fly;
```



```
Var XX, YY:Integer;
Begin
   Show;
   Randomize;
   Repeat
       Locat(XX,YY);
       Repeat {перемещение1}
           XX:= XX + Round((Random-0.5)*Cost)
       Until (XX > 0) And (XX < GetMaxX);
       Repeat {перемещение2}
           YY:= YY + Round((Random-0.5)*Cost)
       Until (YY > 0) And (YY < GetMaxY);
       Hide;
       X:= XX; Y:= YY;
       Show;
       Delay (300)    { мелькание кругов }
   Until Keypressed;      Readln;
End;
Constructor Krug.Init;
Begin
    Point.Init (XN, YN, Color);
    Radius:= R;
    Show
End;
Procedure Krug.Show;
Begin
     SetColor (Cvet);
     SetFillStyle (1, Cvet);
     Pieslice (X, Y, 0, 320, Radius)
End;
Procedure Krug.Hide;
Begin
    SetColor (GetBkColor);
    SetFillStyle (1,GetBKColor);
    Pieslice (X, Y, 0, 320, Radius)
End;
Procedure Ring.Init;
Begin
    Width:= Wid;
    Krug.Init (XN, YN, R, Color)
End;
Constructor Ring.CRing;  Begin  End;
Procedure Ring.Show;
```

```
Begin
     Krug.Show;
     SetFillStyle (10, GetBkColor);
     Pieslice (X, Y, 90 , 320, Radius - Width)
End;

Var
   d,r: Integer;
   TestKrug: Krug;
   TestRing: Ring;
BEGIN
   d:= detect; InitGraph(d,r,'c:\bp\bgi');
   SetBKColor (80);
   TestKrug.Init (150, 40, 50, 1);
   TestRing.CRing;
   TestRing.Init (450, 80, 90, 50, 10);
   TestKrug.Fly (100);
   TestRing.Fly (60);
   TestKrug. Fly (60);
   Readln;
   TestKrug.Hide;
   TestRing.Hide;
     Readln;
END.
```

2. Дополнить набор методов одного из классов функциями, которые выполняют сужение-расширение вырезанного сектора на фигурах и смену цветов фигур одновременно с перемещением.

Отчет по лабораторной работе 7 должен включать:

- блок-схемы алгоритмов всех функций;
- тексты программ с подробными комментариями;
- выводы по проделанным экспериментам;
- ответы на вопросы.

8. ЛАБОРАТОРНАЯ РАБОТА 8

Цель: знакомство с классами потоков языка C++;

организация работы с файлами с помощью объектов-потоков;

изучение свойств потоков, применение методов, позволяющих вмешиваться в состояние потока.

1. Для класса Stroka, написанного на лабораторной работе 6, предусмотреть возможность ввода – вывода его объектов при помощи операций помещения и извлечения (аналогично тому, как это делается для объектов стандартных типов данных).
2. Выполнить программы на стр. 22, 23, 24 метод. указаний «Классы потоков языка программирования C++» [11].
3. Вспомнить программу-шифратор, написанную на лабораторной работе 5. Переписать ее с использованием классов потоков.
4. Изучить методы, позволяющие изменять состояние потоков. В одну из программ, написанных на предыдущих шагах, вставить методы, иллюстрирующую эти возможности. Написать собственную программу, иллюстрирующую эти возможности.

Отчет по лабораторной работе 8 должен включать:

- блок-схемы алгоритмов всех функций;
- тексты программ с подробными комментариями;
- выводы по проделанным экспериментам;
- ответы на вопросы.

9. ЛАБОРАТОРНАЯ РАБОТА 9

Цель: знакомство с механизмом шаблонов C++;

создание и использование шаблонов функций и шаблонов классов в C++.

Шаблоны позволяют давать обобщенное, с точки зрения производительности используемых типов, определение классов и функций. Эти определения могут служить компилятору основой для классов и функций, создаваемых для конкретного типа данных. Обычно шаблоны применяются там, где используются процедуры, которые переписаны много раз для разных типов данных.

Шаблоны часто называют параметризованными типами. Они позволяют компилировать новые классы или функции, задавая типы в качестве параметров.

Различают шаблоны функций и шаблоны классов.

9.1. Шаблоны функций

Это обобщенное определение, из которого компилятор может автоматически генерировать код (создавать представителя) функции.

Синтаксис:

```
template <class T1, …>   //список аргументов шаблона
возвр_тип имя_функции (параметры)
{
    тело функции
}
```

Определение функции похоже на обычное, но один или несколько параметров должны использовать типы, определенные в списке аргументов шаблона.

Например,

```
template <class T>//шаблон функции func воспринимает
void func(T t)    //один параметр произвольного типа
{
    тело функции
}

template <class T>
void Swap(T t[], int i1, int i2)//Воспринимает массив
{                               //произвольного типа и
    T tmp = t[i1];              //два целых числа.
    t[i1]  = t[i2];             //Обменивает содержимое
    t[i2]  = tmp;               //элементов массива с
                                //указанными индексами
}
template <class T>              //шаблон функции func1
void func1(T t1, T t2)          //воспринимает два
{                               //параметра одного и
    тело функции                //того же произвольного
                                //типа
}

template <class T1, class T2>  //шаблон функции func2
void func2(T1 t1, T2 t2)       //воспринимает два
{                              //параметра различного
                               //или одного и того же)
    тело функции               //произвольного типа
}
```


Например, часто приходится сортировать элементы массива. Можно использовать шаблон функции и написать обобщенное определение для сортировки массивов любых типов.

Например, шаблон функции для сортировки массива методом «пузырьковой» сортировки.

```
template <class T>
void Sort(T array[], size_t size)
{
    int i, j;
    T tmp;
for (i = 0; i < size-1; i++)
for( j = size-1; i < j; j--)
    if(array[i] > array[j])
    {
        tmp = array[j];
        array[j] = array[j-1];
        array[j-1] = tmp;
    }
}
```

9.1.1. Использование шаблонов функций

После определения шаблона функции, можно сразу ее вызывать. Компилятор автоматически создаст представителя тела функции для указанных в вызове типов.

Например, использование шаблона функции Sort.

```
#include <iostream.h>
void main(void)
{
    int i;
    int mas[] = {10, 20, 30, 11, 25, 32, 0};
    Sort(mas, sizeof(mas)/sizeof(mas[0]));
    for(i = 0; i < sizeof(mas)/sizeof(mas[0]); i++)
      cout << "mas[" << i << "] = " << mas[i] << endl;
}
```

Шаблон Sort можно применять и для классов, определенных пользователем, но класс должен обязательно перегрузить операцию '>'.

Например, применение шаблона Sort к типу, определенному пользователем.

```
#include <iostream.h>
class Money
{
     long dollars;
     int cents;
  public:
     Money(){}
     Money(long d, int c)
     {
          dollars = d;
          cents = c;
     }
int operator>(const Money&) const;
friend ostream& operator <<(ostream& , Money&);
};

int Money::operator>(const Money& amt) const
{
return
(dollars>amt.dollars)||
  ((dollars==amt.dollars)&&(cents>amt.cents));
}

ostream& operator<<(ostream& os, Money &amt)
{
     os << "$" << amt.dollars << '.' << amt.cents;
     return os;
}

void main(void)
{
     Money mas[]={Money(19,10),
                  Money(99,99),
                  Money(99,95),
                  Money(19,95)
                 };
     Sort(mas, sizeof(mas)/sizeof(mas[0]));
     for(i = 0; i < sizeof(mas)/sizeof(mas[0]); i++)
       cout << "mas[" << i << "] = " << mas[i] << endl;
}
```


9.1.2. Перегрузка шаблонов функций

Как и обычная функция, шаблон функций может быть перегружен, т. е. можно предусмотреть несколько шаблонов функций с одним именем, но различными параметрами.

Например,

```
#include <iostream.h>
template <class T>
T getmax(T t1, T t2)  //Возвращает больший из двух
{                     //параметров
     return t1 > t2?t1:t2;
}

template <class T>
T getmax(T t[], size_t size)
{
     T retval = t[0];
     int i;
     for(i = 0; i < size; i++)
          if(t[i] > retval) retval = t[i];
     return retval;
}

void main(void)
{
     int i1 = 3, i2 = 5;
     int mas[] = {3, 9, 5, 8};
     cout << "max int = " << getmax(i1, i2) << endl;
     cout << "max int = ";
     cout << getmax(sizeof(mas)/sizeof(mas[0]))<< endl;
}
```

9.1.3. Специализация шаблонов функции

Специализированная функция шаблона – это обычная функция, имя которой совпадает с именем функции в шаблоне, но которая описывается для параметров специфических типов. Такие функции объявляют, если обобщенный шаблон не годится.

Например, функцию getmax нельзя использовать для строк. Поэтому шаблон можно дополнить специализированной функцией.

```
#include <iostream.h>
#include <string.h>
template <class T>
T getmax(T t1, T t2)        //Возвращает больший из двух
{                           //параметров
     return t1 > t2?t1:t2;
}
char* getmax(char s1, char s2){
     return strcmp(s1, s2) >0? s1: s2;
}
template <class T>
T getmax(T t[], size_t size)
{
     T retval = t[0];
     int i;
     for(i = 0; i < size; i++)
          if(t[i] > retval) retval = t[i];
     return retval;
}

void main(void)
{
     int i1 = 3, i2 = 5;
     char *s1 = "строка1";
     char *s2 = "строка2";
     cout << "max int = " << getmax(i1, i2) << endl;
     cout << "max str = " << getmax(s1, s2) << endl;
}
```

9.1.4. Разрешение ссылки на функцию

Когда компилятор встречает обращение к функции, для разрешения ссылки применяется следующий алгоритм.

- Найти функцию (не шаблонную), параметры которой соответствуют указанным в вызове.
- Если функция не найдена, найти шаблон, из которого можно генерировать функцию с точным соответствием параметров.
- Если никакой шаблон функции не обеспечивает точного соответствия, снова рассмотреть обычные функции на предмет возможного преобразования типов.

9.2. Шаблоны классов

Шаблоны классов дают обобщенное определение семейства классов, использующее произвольные типы или константы. После определения шаблона класса можно поручить компилятору генерировать на его основе новый класс для конкретного типа или константы.

Синтаксис:

```
template < список аргументов шаблона >
class имя_класса
{
     тело класса
};
```

Список аргументов шаблона представляет собой один или несколько аргументов, перечисленных через запятую. Каждый аргумент является:

- либо именем типа, за которым следует идентификатор;
- либо class T.

Тело класса должно использовать список аргументов шаблона.

Например, шаблон класса для простых стековых операций.

```
template <class T>
class Tstack{
    protected:
        int numItem;
        T *item;
    public:
        Tstack(size_t size = 10)
        {
            numItem = 0;
            item = new T[size];
        }
        ~Tstack()
        {
            delete[] item;
        }
        void push(T t);
        T pop();
};
```

```
template <class T>
void Tstack <T>::push(T t)
{
    item[numItem++] = t;
}
template <class T>
T Tstack <T>::pop()
{
    return item[--numItem];
}
```

Параметры шаблона могут быть либо типами, либо константами.

Например, шаблон класса для обобщенного блока памяти получает константу как параметр.

```
template <int size>
class MemBlock
{
    protected:
        char *p;
    public:
        MemBlock() { p = new char[size];}
        ~MemBlock(){ delete[]  p;       }
        operator char*(){ return p;     }
};
```

Например, шаблон класса, содержащего блок памяти для определенного типа, получает как параметры тип и константу.

```
template <class T, int num>
class TypeBlock{
    protected:
        T* p;
    public:
        TypeBlock();
        ~TypeBlock();
        operator T*();
};
template <class T, int num>
TypeBlock <T, num>::TypeBlock(){
    p = new T[num];
}
```



```
template <class T, int num>
TypeBlock <T, num>::~TypeBlock()
{
    delete[] p ;
}
template <class T, int num>
TypeBlock <T, num>::operator T*(){
    return p;
}
```

9.2.1. Использование шаблонов классов

Чтобы создать представителя шаблонного класса, можно указать имя шаблона со списком аргументов в <>.

Например, использование шаблона класса Tstack.

```
//стек значений int, размер по умолчанию
Tstack <int> st_int1;
//стек для int на 40 элементов
Tstack <int> st_int2(40);
Tstack <long> g *ptr;      //указатель на стек для long
Tstack <double> dbl[10];   //массив стеков для double
Tstack <char*> cSt;        //стек указателей на char
//указатель на стек для указателей на char
Tstack <char*> *ptrcSt;
void Init()  //выделение памяти для переменной ptrcSt
{
    ptrcSt = new Tstack <char*>(20);
}
```

Создав представителя шаблонного класса, можно обращаться с ним как с принадлежащим обычному классу. Например,

```
Tstack <int> st_int(10);
st_int.push(33);
st_int.push(45);
cout << "Значения в стеке = ";
cout << st_int.pop() << ',' << st_int.pop() << endl;
```

Например, применение шаблона класса MemBlock:

```
MemBlock<512> Block;
strcpy(Block, "String1");
cout << (char*)Block << endl;
```

Для упрощения использования шаблонов классов можно использовать ключевое слово typedef. Объявления, сделанные в левом и правом столбиках, означают одно и то же.

```
Tstack <long> lstack;
Tstack <long> *plstack;
```

```
typedef Tstack <long> Lst;
Lst lstack;
Lst *plstack;
```

9.2.2. Специализация шаблонов класса

Можно специализировать шаблон класса, предусмотрев для специфических типов реализацию некоторых методов. Например:

```
template <class T>
class Tarray
{
    protected:
        // элементы данных
    public:
        Tarray(int size = 100)
        {
            //тело конструктора
        }
        ~Tarray()
        {
            //тело деструктора
        }
        void add(T item);
};
void Tarray <char*>::add(char *str)
{
    //специализированный вариант метода add для char*
}
template <class T>
void Tarray <T>::add(T t)
{
    //обобщенное определение метода add для типа T
}
```


Использование специализированного шаблона:

```
int i1 = 10;
Tarray <int> iarray;
iarray.add(i1);
char *msg = "строка символов";
//использует обобщенный вариант шаблона
Tarray <char*> strArray;
//вызов add передается специализированному варианту add
strArray.add(msg);
```

9.3. Задание на лабораторную работу 9

1. Отладить программу, иллюстрирующую применение шаблона Sort к типу, определенному пользователем (см. п. 9.1.1).
2. Выполнить примеры из п.п. 9.1.2 и 9.1.3.
3. Необходимо написать программу, выполняющую сортировку железнодорожного состава с использованием тупика (рис. 1). Работу выполнить с использованием шаблона класса TStack (см. п. 9.2).

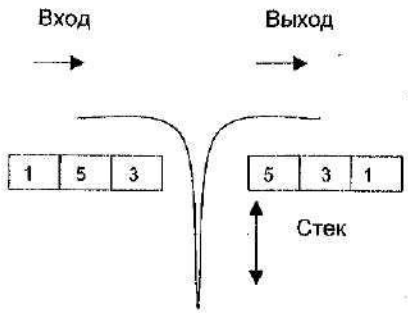

Рис. 1. Схема железнодорожного разъезда для решения задачи

Отчет по лабораторной работе 9 должен включать:

- блок-схемы алгоритмов всех функций;
- тексты программ с подробными комментариями;
- выводы по проделанным экспериментам;
- ответы на вопросы.

10. ЛАБОРАТОРНАЯ РАБОТА 10

Цель: знакомство со средствами создания линейных списков в C++; использование шаблонов для работы с динамическими структурами.

10.1. Динамические структуры данных

Любая программа предназначена для обработки данных, от способа организации которых зависят алгоритмы работы, поэтому выбор структур данных должен предшествовать созданию алгоритмов.

Память под данные выделяется либо на этапе компиляции (в этом случае необходимый объем должен быть известен до начала выполнения программы, то есть задан в виде константы), либо во время выполнения программы с помощью операции new или функции malloc (необходимый объем должен быть известен до распределения памяти). В обоих случаях выделяется непрерывный участок памяти.

Если до начала работы с данными невозможно определить, сколько памяти потребуется для их хранения, память выделяется по мере необходимости отдельными блоками, связанными друг с другом с помощью указателей. Такой способ организации данных называется *динамическими структурами данных,* поскольку их размер изменяется во время выполнения программы. Как известно, представителем динамических структур данных являются линейные списки. Динамическая структура может занимать несмежные участки оперативной памяти.

Динамические структуры широко применяют и для более эффективной работы с данными, размер которых известен, особенно для решения задач сортировки, поскольку упорядочивание динамических структур не требует перестановки элементов, а сводится к изменению указателей на эти элементы. Например, если в процессе выполнения программы требуется многократно упорядочивать большой массив данных, имеет смысл организовать его в виде линейного списка. При решении задач поиска элемента в тех случаях, когда важна скорость, данные лучше всего представить в виде бинарного дерева.

Элемент любой динамической структуры данных представляет собой структуру (в смысле struct), содержащую, по крайней мере, два поля: для хранения данных и для указателя. Полей данных и указателей может быть несколько. Поля данных могут быть любого типа: основного, составного или типа указатель. Описание простейшего элемента (компоненты, узла) может выглядеть следующим образом:


```
struct Node {
 Data d; //тип данных Data должен быть определен ранее
 Node *p;
};
```

10.2. Линейные списки

Самый простой способ связать множество элементов - сделать так, чтобы каждый элемент содержал ссылку на следующий. Такой список называется *однонаправленным (односвязным).* Если добавить в каждый элемент вторую ссылку - на предыдущий элемент, получится *двунаправленный список (двусвязный),* если последний элемент связать указателем с первым, получится *кольцевой список.*

Каждый элемент списка содержит *ключ,* идентифицирующий этот элемент. Ключ обычно бывает либо целым числом, либо строкой и является частью поля данных. В качестве ключа в процессе работы со списком могут выступать разные части поля данных. Например, если создается линейный список из записей, содержащих фамилию, год рождения, стаж работы и пол, любая часть записи может выступать в качестве ключа: при упорядочивании списка по алфавиту ключом будет фамилия, а при поиске, к примеру, ветеранов труда ключом будет стаж. Ключи разных элементов списка могут совпадать.

Над списками можно выполнять следующие операции:

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

Рассмотрим двунаправленный линейный список. Для формирования списка и работы с ним требуется иметь, по крайней мере, один указатель - на начало списка. Удобно завести еще один указатель - на конец списка. Для простоты допустим, что список состоит из целых чисел, то есть описание элемента списка выглядит следующим образом:

```
struct Node{
     int d;
     Node *next;
     Node *prev;
};
```

Ниже приведена программа, которая формирует список из 5 чисел, добавляет число в список, удаляет число из списка и выводит список на экран. Указатель на начало списка обозначен pbeg, на конец списка - pend, вспомогательные указатели - pv и pkey.

```
#include <iostream.h>
struct Node{
     int d;
     Node *next;
     Node *prev;
};
//-----------------------------------------------------
Node *first(int d);
void add(Node **pend, int d);
Node *find(Node * const pbeg, int i);
int remove(Node **pbeg, Node **pend, int key);
Node *insert(Node* const pbeg,Node **pend,int key,int d);
//-----------------------------------------------------
int main()
{
 // Формирование первого элемента списка
 Node *pbeg = first(1);
 // Список заканчивается, едва начавшись
 Node *pend = pbeg;
 //Добавление в конец списка четырех элементов 2,3,4,5
 for (int i = 2; i<6; i++) add(&pend, i);
 // Вставка элемента 200 после элемента 2
 insert(pbeg, &pend, 2, 200);
 // Удаление элемента 5
 if(!remove (&pbeg, &pend, 5)) cout << "не найден";
 Node *pv = pbeg;
 while (pv)                    // вывод списка на экран
 {
     cout << pv->d << ' ';
     pv = pv->next;
 }
 return 0;
}
```



```
// Формирование первого элемента
Node * first(int d)
{
     Node *pv = new Node;
     pv->d = d;
     pv->next = 0;
     pv->prev = 0;
     return pv;
}
// Добавление в конец списка
void add(Node **pend, int d)
{
     Node *pv = new Node:
     pv->d = d;
     pv->next = 0;
     pv->prev = *pend;
     (*pend)->next = pv;
     *pend = pv;
}
//-----------------------------------------------------
// Поиск элемента по ключу
Node * find(Node * const pbeg, int d)
{
     Node *pv = pbeg;
     while (pv)
     {
          if(pv->d = = d) break;
          pv = pv->next;
     }
     return pv;
}
//-----------------------------------------------------
// Удаление элемента
int remove(Node **pbeg, Node **pend, int key)
{
     Node *pkey = find(*pbeg, key);
     if (pkey)                                   // 1
     {
          if (pkey == *pbeg)                     // 2
          {
               *pbeg = (*pbeg)->next;
               (*pbeg)->prev =0;
          }
```

```
        else
            if (pkey == *pend)                    // 3
            {
                *pend = (*pend)->prev;
                (*pend)->next =0;
            }
            else                                  // 4
            {
                (pkey->prev)->next = pkey->next;
                (pkey->next)->prev = pkey->prev;
            }
        delete pkey;
        return 1;                                 // 5
    }
    return 0;                                     // 6
}
//-------------------------------------------------------
// Вставка элемента
Node* insert(Node* const pbeg, Node **pend,int key, int d)
{
  Node *pkey = find(pbeg, key);
  if(pkey)
  {
      Node *pv = new Node;
      pv->d = d;
  // 1 - установление связи нового узла с последующим:
      pv->next = pkey->next;
  // 2 - установление связи нового узла с предыдущим:
      pv->prev = pkey;
  // 3 - установление связи предыдущего узла с новым:
      pkey->next = pv;
  // 4 - установление связи последующего узла с новым:
      if( pkey != *pend)
          (pv->next)->prev = pv;
  // Обновление указателя на конец списка,
  // если узел вставляется в конец:
      else
          *pend = pv;
      return pv;
  }
  return 0;
}
```


Результат работы программы:1 2 200 3 4

Все параметры, не изменяемые внутри функций, должны передаваться с модификатором const. Указатели, которые могут измениться (например, при удалении из списка последнего элемента указатель на конец списка требуется скорректировать), передаются по адресу.

Рассмотрим подробнее *функцию удаления элемента из списка* remove. Ее параметрами являются указатели на начало и конец списка, и ключ элемента, подлежащего удалению. *В строке 1* выделяется память под локальный указатель pkey, которому присваивается результат выполнения функции нахождения элемента по ключу find. Эта функция возвращает указатель на элемент в случае успешного поиска и 0, если элемента с таким ключом в списке нет. Если pkey получает ненулевое значение, условие в операторе if становится истинным, (элемент существует), и управление передается оператору 2, если нет - выполняется возврат из функции со значением false (оператор 6).

Удаление из списка происходит по-разному в зависимости от того, находится элемент в начале списка, в середине или в конце. *В операторе 2* проверяется, находится ли удаляемый элемент в начале списка - в этом случае следует скорректировать указатель pbeg на начало списка так, чтобы он указывал на следующий элемент в списке, адрес которого находится в поле next первого элемента. Новый начальный элемент списка должен иметь в своем поле указателя на предыдущий элемент значение 0.

Если удаляемый элемент находится в конце списка *(оператор 3)*, требуется сместить указатель pend конца списка на предыдущий элемент, адрес которого можно получить из поля prev последнего элемента. Кроме того, нужно обнулить для нового последнего элемента указатель на следующий элемент. Если удаление происходит из середины списка, то единственное, что надо сделать, - обеспечить двустороннюю связь предыдущего и последующего элементов. После корректировки указателей память из-под элемента освобождается, и функция возвращает значение true.

Работа *функции вставки элемента* в список проиллюстрирована на рис.2. Номера около стрелок соответствуют номерам операторов в комментариях.

Сортировка связанного списка заключается в изменении связей между элементами. Алгоритм состоит в том, что исходный список просматривается, и каждый элемент вставляется в новый список на место, определяемое значением его ключа.

Ниже приведена *функция формирования упорядоченного списка* (предполагается, что первый элемент существует):

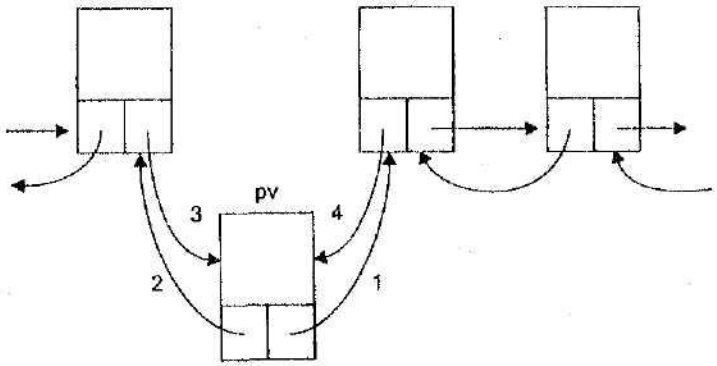

Рис. 2. Вставка элемента в список

```
void add_sort(Node **pbeg, Node **pend, int d)
{
  Node *pv = new Node;      // добавляемый элемент
  pv->d = d;
  Node *pt = *pbeg;
  while (pt)                //просмотр списка
  {
    if(d<pt->d) //занести перед текущим элементом (pt)
    {
      pv->next = pt;
      if (pt == *pbeg) // в начало списка
      {
         pv->prev = 0;
         *pbeg = pv;
      }
      else              // в середину списка
      {
         (pt->prev)->next = pv;
         pv->prev = pt->prev;
      }
      pt->prev = pv;
      return;
    }
    pt = pt->next;
  }
  pv->next = 0;                  // в конец списка
  pv->prev = *pend;
  (*pend)->next = pv;
  *pend = pv;
}
```


10.3. Задание на лабораторную работу 10

1. На основании приведенной программы для двусвязного списка написать аналогичную по выполняемым действиям программу для односвязного списка.

2. Создать шаблон класса «односвязный линейный список».

3. Использовать его для решения следующей задачи:

Составить программу, которая содержит динамическую информацию о наличии автобусов в автобусном парке.

Сведения о каждом автобусе содержат:

- номер автобуса;
- фамилию и инициалы водителя;
- номер маршрута.

Программа должна обеспечивать:

- начальное формирование данных обо всех автобусах в парке в виде списка;
- при выезде каждого автобуса из парка вводится номер автобуса, и программа удаляет данные об этом автобусе из списка автобусов, находящихся в парке, и записывает эти данные в список автобусов, находящихся на маршруте;
- при въезде каждого автобуса в парк вводится номер автобуса, и программа удаляет данные об этом автобусе из списка автобусов, находящихся на маршруте, и записывает эти данные в список автобусов, находящихся в парке;
- по запросу выдаются сведения об автобусах, находящихся в парке, или об автобусах, находящихся на маршруте.

Отчет по лабораторной работе 10 должен включать:

- блок-схемы алгоритмов всех функций;
- тексты программ с подробными комментариями
- выводы по проделанным экспериментам;
- ответы на вопросы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Елманова Н.З. Введение в Borland C++ Builder [Текст] / Н. З. Елманова, С. П. Кошель. - М.: Диалог–МИФИ, 1997. - 272 с.
2. Керниган Б. Язык программирования Си [Текст] / Б. Керниган, Д. Ритчи; пер. с англ. – 3-е изд., испр. - СПб.: Невский Диалект, 2001. - 352 с.; ил.
3. Подбельский В.В. Программирование на языке Си [Текст]: учеб. пособие / В.В. Подбельский, С.С. Фомин – 2-е изд., доп. - М.: Финансы и статистика, 2000. – 600 с.; ил.
4. Скляров В.А. Программирование на языках Си и Си++ [Текст]: учеб. пособие. / В.А. Скляров М.: Высшая школа, 1999. – 288 с.
5. Крячков А.В. Программирование на С и С++. Практикум [Текст]: учеб. пособие для вузов / А.В. Крячков, И.В. Сухинина, В.К. Томшин; под ред. В.К. Томшина. – 2-е изд., испр. - М.: Горячая линия – Телеком, 2000. – 344 с.; ил.
6. Страуструп Б. Язык программирования Си++ [Текст] / Б. Страуструп; пер. с англ. – М.: Радио и связь, 1991. – 352 с.
7. Бабэ Бруно. Просто и ясно о Borland C++ [Текст] / Бруно Бабэ; пер. с англ. – М.: БИНОМ, 1995. – 400 с.
8. Павловская Т. А. С/С++. Программирование на языке высокого уровня [Текст] / Т. А. Павловская. – СПб.: Питер, 2001. – 464 с.
9. Чагаева О.Л. Основные конструкции языка программирования С++ [Текст]: Методические указания по дисциплине "Системное программирование" / О.Л. Чагаева. Екатеринбург: изд. ИПК УГТУ, 2001. - 42 с.
10. Чагаева О.Л. Объектно-ориентированное программирование на языке С++ [Текст]: Методические указания по дисциплине "Системное программирование" / О.Л. Чагаева. Екатеринбург: изд. ИПК УГТУ, 2001. - 44 с.
11. Чагаева О.Л. Классы потоков в языке программирования С++ [Текст]: Методические указания по дисциплине "Системное программирование" / О.Л. Чагаева. Екатеринбург: изд. ИПК УГТУ, 2001. - 36 с.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 3

1. ЛАБОРАТОРНАЯ РАБОТА 1 4

2. ЛАБОРАТОРНАЯ РАБОТА 2 6

3. ЛАБОРАТОРНАЯ РАБОТА 3 8

4. ЛАБОРАТОРНАЯ РАБОТА 4 9

5. ЛАБОРАТОРНАЯ РАБОТА 5 12

6. ЛАБОРАТОРНАЯ РАБОТА 6 13

7. ЛАБОРАТОРНАЯ РАБОТА 7 16

8. ЛАБОРАТОРНАЯ РАБОТА 8 19

9. ЛАБОРАТОРНАЯ РАБОТА 9 19

9.1. Шаблоны функций 20

9.1.1. Использование шаблонов функций *21*

9.1.2. Перегрузка шаблонов функций *23*

9.1.3. Специализация шаблонов функции *23*

9.1.4. Разрешение ссылки на функцию *24*

9.2. Шаблоны классов 25

9.2.1. Использование шаблонов классов *27*

9.2.2. Специализация шаблонов класса *28*

9.3. Задание на лабораторную работу 9 29

10. ЛАБОРАТОРНАЯ РАБОТА 10 30

10.1. Динамические структуры данных 30

10.2. Линейные списки 31

10.3. Задание на лабораторную работу 10 37

БИБЛИОГРАФИЧЕСКИЙ СПИСОК 38