Лабораторная работа №3

Массивы структур и строки. Хранение данных на внешних носителях

1. Цель работы:

1. Получить практические навыки работы со строковыми данными.
2. Получить практические навыки работы со структурами.
3. Получить практические навыки организации динамических массивов с элементами сложной структуры.
4. Получение практических навыков записи структурированной информации в файлы в стиле C;
5. Получение практических навыков записи структурированной информации в файлы в стиле C++;

2. Теоретические сведения

2.1.1. Структуры

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

```
struct Date          //определение структуры
{
     int day;
     int month;
     int year;
};

Date birthday;       //переменная типа Date
```

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

Доступ к элементам структур обеспечивается с помощью уточненных имен:
`имя_структуры.имя_элемента`

```
     //присваивание значений полям переменной birthday
     birthday.day=11; birthday.month=3; birthday.year=1993;
     Date Data;
     // присваивание значения переменной birthday переменной Data
     Data=birthday;
```

Из элементов структурного типа можно организовывать массивы также как из элементов стандартных типов.

```
     Date mas[15];  //массив структур

     //ввод значений массива
     for(int i=0;i<15;i++)
     {
          cout<<"\nEnter day:";cin>>mas[i].day;
          cout<<"\nEnter month:";cin>>mas[i].month;
          cout<<"\nEnter year:";cin>>mas[i].year;
     }
```

2.1.2. Строки

Строка в C++ – это массив символов, заканчивающийся нуль-символом – `'\0'` (нуль-терминатором). По положению нуль-терминатора определяется фактическая длина строки. Количество элементов в таком массиве на 1 больше, чем изображение строки.

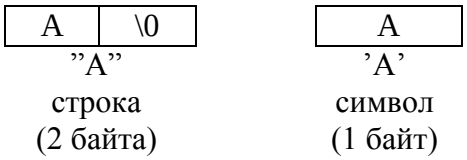


Рис. 1. Представление строки и символа

Присвоить значение строке с помощью оператора присваивания нельзя. Поместить строку в массив можно либо при вводе, либо с помощью инициализации.

```
char s1[10]="string1";//инициализация
char s2[]="string2";//инициализация
char str2[]={'a', 'b','s','w','\0'};//инициализация
char s3[10];
cin>>s3;//ввод до первого разделителя
//выделение памяти под динамическую строку
char *s4=new char[strlen(s3)+1];
strcpy(s4,s3);//копирование строки s3 в строку s4

char str3[100];
gets(str3);// чтение строки до символа \n
puts(str3);//вывод строки в стандартный поток

char str4[100];
scanf('%s',str4);// чтение строки до первого разделителя(пробел, табуляция и т.д.)
printf("%s",str4);//вывод строки в стандартный поток
```

Для работы со строками существуют специальные библиотечные функции, которые содержатся в заголовочном файле `string.h` (или `cstring.h`).

Прототип функции	Краткое описание	Примечание
`unsigned` **`strlen`**`(const char* s);`	Вычисляет длину строки s.	
`int` **`strcmp`**`(const char* s1, const char* s2);`	Сравнивает строки s1 и s2.	Если s1<s2, то результат отрицательный, если s1==s2, то результат равен 0, если s2>s1 – результат положительный.
`int` **`strcnmp`**`(const char* s1, const char* s2);`	Сравнивает первые n символов строк s1 и s2.	Если s1<s2, то результат отрицательный, если s1==s2, то результат равен 0, если s2>s1 – результат положительный.
`char*` **`strcpy`**`(char* s1, const char* s2);`	Копирует символы строки s1 в строку s2.	
`char*` **`strncpy`**`(char* s1, const char* s2, int n);`	Копирует n символов строки s1 в строку s2.	Конец строки отбрасывается или дополняется пробелами.

`char* strcat(char* s1, const char* s2);`	Приписывает строку s2 к строке s1	
`char* strncat(char* s1, const char* s2);`	Приписывает первые n символов строки s2 к строке s1	
`char* strdup(const char* s);`	Выделяет память и переносит в нее копию строки s	При выделении памяти используются функции

2.1.3. Передача строк в качестве параметров функций

Строки при передаче в функции могут передаваться как одномерные массивы типа char или как указатели типа char*. В отличие от обычных массивов в функции не указывается длина строки, т. к. в конце строки есть признак конца строки /0.

```
//Функция поиска заданного символа в строке
int  find(char *s,char c)
{
for (int I=0;I<strlen(s);I++)
if(s[I]==c) return I;
return -1
}
```

2.1.4. Потоковый ввод-вывод в стиле C

2.1.4.1. Режимы открытия файлов

Особенностью C является отсутствие в этом языке структурированных файлов. Все файлы рассматриваются как не структурированная последовательность байтов. При таком подходе понятие файла распространяется и на различные устройства.

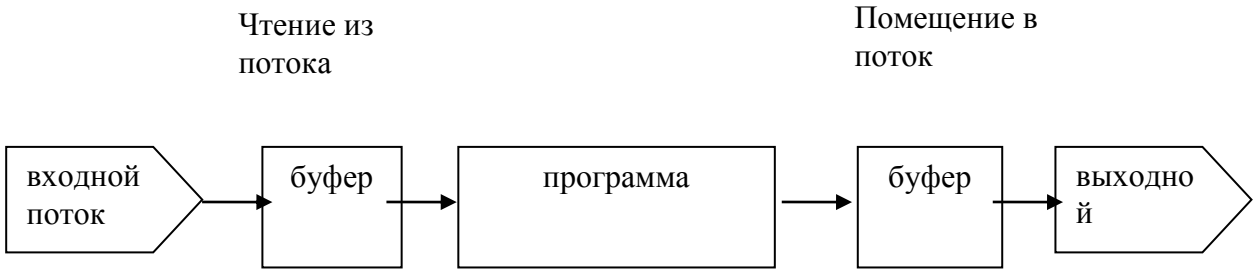
В C существуют средства ввода-вывода. Все операции ввода-вывода реализуются с помощью функций, которые находятся в библиотеке C. Библиотека C поддерживает три уровня ввода-вывода:

- потоковый ввод-вывод;
- ввод-вывод нижнего уровня;
- ввод-вывод для консоли и портов (зависит от ОС).

Поток – это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику.

Чтение данных из потока называется извлечением, вывод в поток – помещением, или включением.

Поток определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен (оперативная память, файл на диске, клавиатура или принтер). Обмен с потоком для увеличения скорости передачи данных производится, как правило, через специальную область оперативной памяти — буфер. Буфер накапливает байты, и фактическая передача данных выполняется после заполнения буфера. При вводе это дает возможность исправить ошибки, если данные из буфера еще не отправлены в программу.



При работе с потоком можно:

- Открывать и закрывать потоки (связывать указатели на поток с конкретными файлами);
- вводить и выводить строку, символ, форматированные данные, порцию данных произвольной длины;
- анализировать ошибки ввода-вывода и достижения конца файла;
- управлять буферизацией потока и размером буфера;
- получать и устанавливать указатель текущей позиции в файле.

Функции библиотеки ввода-вывода находятся в заголовочном файле <stdio.h>.

Прежде чем начать работать с потоком, его надо инициировать, т. е. открыть. При этом поток связывается со структурой предопределенного типа FILE, определение которой находится в библиотечном файле <stdio.h>. В структуре находится указатель на буфер, указатель на текущую позицию файла и т. п. При открытии потока, возвращается указатель на поток, т. е. на объект типа FILE.

```
#include <stdio.h>;
. . . . . .
FILE *fp;
. . . . . ..
fp= fopen( ”t.txt”, ”r”);
```

где `fopen(<имя_файла>,<режим_открытия>)` - функция для инициализации файла.
Существуют следующие режимы для открытия файла:

Режим	Описание режима открытия файла
r	Файл открывается для чтения, если файл не существует , то выдается ошибка при исполнении программы.
w	Файл открывается для записи, если файл не существует, то он будет создан, если файл уже существует, то вся информация из него стирается.
a	Файл открывается для добавления, если файл не существует, то он будет создан, если существует, то информация из него не стирается, можно выполнять запись в конец файла
r+	Файл открывается для чтения и записи, изменить размер файла нельзя, если файл не существует , то выдается ошибка при исполнении программы.
w+	Файл открывается для чтения и записи, если файл не существует, то он будет создан, если файл уже существует, то вся информация из него стирается.
a+	Файл открывается для чтения и записи, если файл не существует, то он будет создан, если существует, то информация из него не стирается, можно выполнять запись в конец файла

Поток можно открыть в текстовом (t) или двоичном (b) режиме. По умолчанию используется текстовый режим. В явном виде режим указывается следующим образом:

- ”`r+b`”или ”`rb`” - двоичный (бинарный) режим;
- ”`r+t`” или ”`rt`” – текстовый режим.

В файле stdio.h определена константа EOF, которая сообщает об окончании файла (отрицательное целое число).

При открытии потока могут возникать следующие ошибки:

- файл, связанный с потоком не найден (при чтении из файла);
- диск заполнен (при записи);
- диск защищен от записи (при записи) и т. п.

В этих случаях указатель на поток приобретет значение NULL (0). Указатель на поток, отличный от аварийного не равен 0.

Для вывода об ошибке при открытии потока используется стандартная библиотечная функция из файла <stdio.h>

```
void perror (const char*s);
```

```
if ((fp=fopen(”t.txt”, ”w”)==NULL)
{
// выводит строку символов с сообщением об ошибке
perror(”\nошибка при открытии файла”);
exit(0);
}
```

После работы с файлом, его надо закрыть

```
fclose(<указатель_на_поток>);
```

Когда программа начинает выполняться, автоматически открываются несколько потоков, из которых основными являются:

- стандартный поток ввода (stdin);
- стандартный поток вывода (stdout);
- стандартный поток вывода об ошибках (stderr).

По умолчанию stdin ставится в соответствие клавиатура, а потокам stdout и stderr - монитор. Для ввода-вывода с помощью стандартных потоков используются функции:

- getchar()/putchar() – ввод-вывод отдельного символа;
- gets()/puts() – ввод-вывод строки;
- scanf()/printf() – форматированный ввод/вывод.

Аналогично работе со стандартными потоками выполняется ввод-вывод в потоки, связанные с файлами.

Для символьного ввода-вывода используются функции:

- int fgetc(FILE*fp), где fp – указатель на поток, из которого выполняется считывание. Функция возвращает очередной символ в форме int из потока fp. Если символ не может быть прочитан, то возвращается значение EOF.
- int fputc(int c, FILE*fp), где fp – указатель на поток, в который выполняется запись, c – переменная типа int, в которой содержится записываемый в поток символ. Функция возвращает записанный в поток fp символ в форме int . Если символ не может быть записан, то возвращается значение EOF.

Для построчного ввода-вывода используются следующие функции:

- char* fgets(char* s,int n,FILE* f), где char*s – адрес, по которому размещаются считанные байты, int n – количество считанных байтов, FILE* f – указатель на файл, из которого производится считывание.

Прием байтов заканчивается после передачи n-1 байтов или при получении управляющего символа ‘\n’. Управляющий символ тоже передается в принимающую строку. Строка в любом случае заканчивается ‘\0’. При успешном завершении работы функция возвращает указатель на прочитанную строку, при неуспешном – 0.

- int puts(char* s, FILE* f), где char*s – адрес, из которого берутся записываемые в файл байты, FILE* f – указатель на файл, в который производится запись.

Символ конца строки (‘\0’) в файл не записывается. Функция возвращает EOF, если при записи в файл произошла ошибка, при успешной записи возвращает неотрицательное число.

Для блокового ввода-вывода используются функции:

- int fread(void*ptr,int size, int n, FILE*f), где void*ptr – указатель на область памяти, в которой размещаются считанные из файла данные, int size – размер одного считываемого элемента, int n – количество считываемых элементов, FILE*f – указатель на файл, из которого производится считывание.

В случае успешного считывания функция возвращает количество считанных элементов, иначе – EOF.

- `int fwrite(void*ptr,int size, int n, FILE*f)`, где `void*ptr` – указатель на область памяти, в которой размещаются считанные из файла данные, `int size` – размер одного записываемого элемента, `int n` – количество записываемых элементов, `FILE*f` – указатель на файл, в который производится запись.

В случае успешной записи функция возвращает количество записанных элементов, иначе – EOF.

В некоторых случаях информацию удобно записывать в файл без преобразования, т. е. в символьном виде пригодном для непосредственного отображения на экран. Для этого можно использовать функции форматированного ввода-вывода:

- `int fprintf(FILE *f, const char*fmt,. . .)` , где `FILE*f` – указатель на файл, в который производится запись, `const char*fmt` – форматная строка, . . . – список переменных, которые записываются в файл.

Функция возвращает число записанных символов.

- `int fscanf(FILE *f, const char*fmt, par1,par2, . . .)` , где `FILE*f` – указатель на файл, из которого производится чтение, `const char*fmt` – форматная строка, `par1,par2,. . .` – список переменных, в которые заносится информация из файла.

Функция возвращает число переменных, которым присвоено значение.

Средства прямого доступа дают возможность перемещать указатель текущей позиции в потоке на нужный байт. Для этого используется функция

- `int fseek(FILE *f, long off, int org)`, где `FILE *f` - – указатель на файл, `long off` – позиция смещения, `int org` – начало отсчета.

Смещение задается выражение или переменной и может быть отрицательным, т. е. возможно перемещение как в прямом, так и в обратном направлениях. Начало отсчета задается одной из определенных в файле <stdio.h> констант:

`SEEK_SET ==0` – начало файла;
`SEEK_CUR==1` – текущая позиция;
`SEEK_END ==2` – конец файла.

Функция возвращает 0, если перемещение в потоке выполнено успешно, иначе возвращает ненулевое значение.

2.1.4.2 Обработка элементов файла

Для того чтобы удалить элемент из файла нужно использовать вспомогательный файл. Во вспомогательный файл переписываются все элементы исходного файла за исключением тех, которые требуется удалить. После этого исходный файл удаляется из памяти, а вспомогательному файлу присваивается имя исходного файла.

```
void del(char *filename)
{//удаление записи с номером x
FILE *f;//исходный файл
FILE*temp;//вспомогательный файл
//открыть исходный файл для чтения
f=fopen(filename,”rb”);
//открыть вспомогательный файл для записи
temp=fopen(”temp”,”wb”)
student a;//буфер для чтения данных из файла
//считываем данные из исходного файла в буфер
for(long i=0; fread(&a,sizeof(student),1,f);i++)
     if(i!=x)//если номер записи не равен x
     {
     //записываем данные из буфера во временный файл
     fwrite(&a,sizeof(student),1,temp);
```

```
        }
        else
        {
        cout<<a<<" - is deleting...";
        }
    fclose(f);//закрываем исходный файл
    fclose(temp); //закрываем временный файл
    remove(filename);//удаляем исходный файл
    rename("temp", filename);//переименовываем временный файл
    }
```

Для корректировки элементов файла используется аналогичный алгоритм. Данные из исходного файла переписываются во вспомогательный файл, но записи, которые нужно изменить записываются в откорректированном виде.

Для добавления элементов в начало или в середину файла также используется вспомогательный файл, в который в нужное место добавляются новые данные.

Для добавления элементов конец файла достаточно открыть его в режиме “a” или “a+” (для добавления) и записать новые данные в конец файла.

```
    f=fopen(filename,"ab");//открыть файл для добавления
    cout<<"\nHow many records would you add to file?";
    cin>>n;
    for(int i=0;i<n;i++)
    {
    //прочитать объект
    fwrite(&a,sizeof(student),1,f);//записать в файл
    }
    fclose(f);//закрыть файл
```

2.1.5. Потоковый ввод-вывод в стиле C++

C++ предоставляет возможность ввода/вывода как на низком уровне – неформатированный ввод-вывод, так и на высоком – форматированный ввод-вывод. При неформатированном вводе/выводе передача информации осуществляется блоками байтов данных без какого-либо преобразования. При форматированном - байты группируются таким образом, чтобы их можно было воспринимать как типизированные данные (целые числа, строки символов, числа с плавающей запятой и т. п.)

По направлению обмена потоки можно разделить на

- входные (данные вводятся в память),
- выходные (данные выводятся из памяти),
- двунаправленные (допускающие как извлечение, так и включение).

По виду устройств, с которыми работает поток, потоки можно разделить на стандартные, файловые и строковые:

- стандартные потоки предназначены для передачи данных от клавиатуры и на экран дисплея,
- файловые потоки — для обмена информацией с файлами на внешних носителях данных (например, на магнитном диске),
- строковые потоки — для работы с массивами символов в оперативной памяти.

Для работы со стандартными потоками библиотека C++ содержит библиотеку `<iostream.h>`. При этом в программе автоматически становятся доступными объекты:

- `cin` - объект, соответствует стандартному потоку ввода,
- `cout` - объект, соответствует стандартному потоку вывода.

Оба эти потока являются буферизированными.

Форматированный ввод/вывод реализуется через две операции:

- << - вывод в поток;
- >> - чтение из потока.

Использование файлов в программе предполагает следующие операции:

- создание потока;
- открытие потока и связывание его с файлом;
- обмен (ввод/вывод);
- уничтожение потока;
- закрытие файла.

Для работы со файловыми потоками библиотека C++ содержит библиотеки:

- `<ifstream.h>` - для работы с входными потоками,
- `<ofstream.h>` - для работы с выходными потоками
- `<fstream.h>` - для работы с двунаправленными потоками.

В библиотечных файлах потоки описаны как классы, т. е. представляют собой пользовательские типы данных (аналогично структурам данных). В описание класса, кроме данных, добавляются описания функций, обрабатывающих эти данные (соответственно компонентные данные и компонентные функции (методы)). Обращаться к компонентным функциям можно также как и к компонентным данным с помощью . или ->.

Для создания файлового потока используются специальные методы –конструкторы, которые создают поток соответствующего класса, открывают файл с указанным именем и связывают файл с потоком:

- `ifstream(const char *name, int mode = ios::in);//входной поток`
- `ofstream(const char *name, int mode = ios::out | ios::trunc);//выходной поток`
- `fstreamCconst char *name, int mode = ios::in | ios::out);//двунаправленный поток`

Вторым параметром является режим открытия файла. Вместо значения по умолчанию можно указать одно из следующих значений, определенных в классе ios.

`ios::in`	открыть файл для чтения;
`ios::out`	открыть файл для записи;
`ios::ate`	установить указатель на конец файла, читать нельзя, можно только записывать данные в конец файла;
`ios::app`	открыть файл для добавления;
`ios::trunc`	если файл существует, то создать новый;
`ios::binary`	открыть в двоичном режиме;
`ios::nocreate`	если файл не существует, выдать ошибку, новый файл не открывать
`ios::noreplace`	если файл существует, выдать ошибку, существующий файл не открывать;

Открыть файл в программе можно с использованием либо конструкторов, либо метода `open`, имеющего такие же параметры, как и в соответствующем конструкторе.

```
fstream f; //создает  файловый поток f
//открывается файл, который связывается с потоком
f.open("..\\f.dat",ios::in);
```

```
// создает и открывает для чтения файловый поток f
fstream f ("..\\f.dat",ios::in);
```

После того как файловый поток открыт, работать с ним можно также как и со стандартными потоками `cin` и `cout`. При чтении данных из входного файла надо контролировать, был ли достигнут конец файла после очередной операции вывода. Это можно делать с помощью метода `eof()`.

Если в процессе работы возникнет ошибочная ситуация, то потоковый объект принимает значение равное 0.

Когда программа покидает область видимости потокового объекта, то он уничтожается, при этом перестает существовать связь между потоковым объектом и физическим файлом, а сам файл закрывается. Если файл требуется закрыть раньше, то используется метод `close()`.

```
//создание файла из элементов типа person
struct person
{
char name[20];
int age;
};
person *mas;//динамический массив
fstream f("f.dat",ios::out);//двунаправленный файловый поток
int n;
cout<<"N?";
cin>>n;
mas=new person[n];//создаем динамический массив
for(int i=0;i<n;i++)
{
     cout<<"?";
//ввод одного элемента типа person из стандартного потока cin
     cin>>mas[i].name;
     cin>>mas[i].age;
}
//запись элементов массива в файловый поток
for(i=0;i<n;i++)
{

     f<<mas[i].name;f<<"\n";
     f<<mas[i].age;f<<"\n";

}
 f.close();//закрытие потока

//чтение элементов из файла
person p;
f.open("f.dat",ios::in);//открываем поток для чтения
do
{
/*читаем элементы типа person из файлового потока f в переменную p*/
     f>>p.name;
     f>>p.age;
     // если достигнут конец файла, выходим из цикла
     if (f.eof())break;
     //вывод на экран
     cout<<p.name<<" "<<p.age<<"\n";

}while(!f.eof());
f.close();//закрытие потока
```

Часть 2. Функции в C/ C++.

2.2.1. Функции с начальными значениями параметров (по-умолчанию)

В определении функции может содержаться начальное (умалчиваемое) значение параметра. Это значение используется, если при вызове функции соответствующий параметр опущен. Все параметры, описанные справа от такого параметра, также должны быть умалчиваемыми.

```
const int N=20;//количество элементов массива
char mas1[N][10];//массив имен
int mas2[N]//массив возрастов
```

```
void init(int i, char* name="Вася ", int age=17)
{
strcpy(mas1[i],name);
mas2[i]=age;
}
```

Примеры использования функции `init()`:

1.
```
for (int i=1;i<N;i++)
      init(i) ;
```
Всем элементам массива `mas1` присваивается значение «Вася», всем элементам массива `mas2` присваивается значение 17.

2.
```
for (int i=1;i<N;i++)
      {
      char Name[10];
      cout<<"Введите имя:"; cin>>Name;
      init(I,Name) ;
      }
```
Всем элементам массива `mas1` присваивается значение переменной Name, всем элементам массива `mas2` присваивается значение 17.

2.2.2. Функции с переменным числом параметров

В C++ допустимы функции, у которых при компиляции не фиксируется число параметров, и, кроме того, может быть неизвестен тип этих параметров. Количество и тип параметров становится известным только в момент вызова, когда явно задан список фактических параметров. Каждая функция с переменным числом параметров должна иметь хотя бы один обязательный параметр. Определение функции с переменным числом параметров:

```
тип имя (явные параметры,. . . )
{
      тело функции
}
```

После списка обязательных параметров ставится запятая, а затем многоточие, которое показывает, что дальнейший контроль соответствия количества и типов параметров при обработке вызова функции производить не нужно. При обращении к функции все параметры и обязательные, и необязательные будут размещаться в памяти друг за другом. Следовательно, определив адрес обязательного параметра как `p=&k`, где `p` - указатель, а `k` – обязательный параметр, можно получить адреса и всех остальных параметров: оператор `k++`; выполняет переход к следующему параметру списка. Еще одна сложность заключается в определении конца списка параметров, поэтому каждая функция с переменным числом параметров должна иметь механизм определения количества и типов параметров. Существует два подхода:

1) известно количество параметров, которое передается как обязательный параметр;
2) известен признак конца списка параметров.

```
//Найти сумму последовательности
//чисел, если известно количество чисел
      #include <iostream>
      using namespace std;
      int sum(int k, ...)
      //явный параметр k задает количество чисел
      {
      int *p=&k;//настроили указатель на параметр k
      int s=0;
      for(;k!=0;k--)
      s+=*(++p);
      return s;
      }
      int main()
      {
```

```
      //среднее арифметическое 4+6
      cout<<"\n 4+6="<<sum(2,4,6);
      //среднее арифметическое 1+2+3+4
      cout<<"\n 1+2+3+4="<<sum(4,1,2,3,4);
      return 0;
      }
```

Для доступа к списку параметров используется указатель `*p` типа `int`. Он устанавливается на начало списка параметров в памяти, а затем перемещается по адресам фактических параметров (`++p`).

```
/*Найти сумму последовательности чисел, если известен признак конца списка
параметров */
#include <iostream>
using namespace std;
int sum(int k, ...)
{
int *p = &k;     //настроили указатель на параметр k
int s = 0;       //значение первого параметра присвоили s
for(int i=1;*p!=0;p++)     //пока нет конца списка
s += *p;
return s;
}

int main()
{
//находит среднее арифметическое 4+6
cout<<"\n 4+6="<<sum(4,6,0);
//находит среднее арифметическое 1+2+3+4
cout<<"\n 1+2+3+4="<<sum(1,2,3,4,0);
}
```

2.2.3. Перегрузка функций

Цель перегрузки состоит в том, чтобы функция с одним именем по-разному выполнялась и возвращала разные значения при обращении к ней с различными типами и различным числом фактических параметров. Для обеспечения перегрузки необходимо для каждой перегруженной функции определить возвращаемые значения и передаваемые параметры так, чтобы каждая перегруженная функция отличалась от другой функции с тем же именем. Компилятор определяет, какую функцию выбрать по типу фактических параметров.

```
#include <iostream.h>
#include <string.h>

//сравнение двух целых чисел
int max(int a, int b)
{
     if (a>b) return a;
     else return b;
}

//сравнение двух вещественных чисел
float max(float a, float b)
{
     if(a>b)return a;
     else return b;
}

//сравнение двух строк
char* max(char* a, char* b)
{
     if (strcmp(a,b)>0) return a;
     else return b;
```

```
}

void main()
{
      int a1,b1;
      float a2, b2;
      char s1[20];
      char s2[20];

      cout<<"\nfor int:\n";
      cout<<"a=?";cin>>a1;
      cout<<"b=?";cin>>b1;
      cout<<"\nMAX="<<max(a1,b1)<<"\n";

      cout<<"\nfor float:\n";
      cout<<"a=?";cin>>a2;
      cout<<"b=?";cin>>b2;
      cout<<"\nMAX="<<max(a2,b2)<<"\n";

      cout<<"\nfor char*:\n";
      cout<<"a=?";cin>>s1;
      cout<<"b=?";cin>>s2;
      cout<<"\nMAX="<<max(s1,s2)<<"\n";
}
```

Правила описания перегруженных функций:

- Перегруженные функции должны находиться в одной области видимости.
- Перегруженные функции могут иметь параметры по умолчанию, при этом значения одного и того же параметра в разных функциях должны совпадать. В разных вариантах перегруженных функций может быть разное количество умалчиваемых параметров.
- Функции не могут быть перегружены, если описание их параметров отличается только модификатором `const` или наличием ссылки: функции `int& f1(int&, const int&){…}` и `int f1(int, int){…}` – не являются перегруженными, т. к. компилятор не сможет узнать какая из функций вызывается, потому что нет синтаксических отличий между вызовом функции, которая передает параметр по значению и функции, которая передает параметр по ссылке.

2.2.3. Шаблоны функций

Шаблоны вводятся для того, чтобы автоматизировать создание функций, обрабатывающих разнотипные данные. Например, алгоритм сортировки можно использовать для массивов различных типов. При перегрузке функции для каждого используемого типа определяется своя функция. Шаблон функции определяется один раз, но определение параметризируется, т. е. тип данных передается как параметр шаблона.

```
template <class имя_типа [,class имя_типа]>
заголовок_функции
{
      тело функции
}
```

Таким образом, шаблон семейства функций состоит из 2 частей – заголовка шаблона: `template<список параметров шаблона>` и обыкновенного определения функции, в котором вместо типа возвращаемого значения и/или типа параметров, записывается имя типа, определенное в заголовке шаблона.

```
//сравнение двух чисел любого типа
template<class T>
T max(T a, T b)
{
      if (a>b) return a;
      else return b;
```

```
}
```

Шаблон служит для автоматического формирования конкретных описаний функций по тем вызовам, которые компилятор обнаруживает в программе. Например, если в программе вызов функции осуществляется как `max(1,5)`, то компилятор сформирует определение функции `int max(int a, int b){…}`.

2.2.4. Указатель на функцию

Каждая функция характеризуется типом возвращаемого значения, именем и списком типов ее параметров. Если имя функции использовать без последующих скобок и параметров, то он будет выступать в качестве указателя на эту функцию, и его значением будет выступать адрес размещения функции в памяти. Это значение можно будет присвоить другому указателю. Тогда этот новый указатель можно будет использовать для вызова функции.

Указатель на функцию определяется следующим образом:

```
тип_функции(*имя_указателя)(спецификация параметров)
```

В определении указателя количество и тип параметров должны совпадать с соответствующими типами в определении функции, на которую ставится указатель.

Вызов функции с помощью указателя имеет вид:

```
(*имя_указателя)(список фактических параметров);
```

```
#include <iostream.h>
int f1(char* S)
{
cout<<S<<"\n";
return 1;
}

void main()
{
      char s[20]="\nfunction1";
      int(*ptr1)(char*);    //указатель на функцию
      ptr1=f1;         //указателю присваивается адрес функции f1
      cout<<((*ptr1)(s));   //вызов функции f1 с помощью указателя
}
```

Указатели на функции удобно использовать в тех случаях, когда функцию надо передать в другую функцию как параметр.

```
#include <iostream.h>
#include <math.h>

typedef float(*fptr)(float);//тип-указатель на функцию уравнения

/*решение уравнения методом половинного деления, уравнение передается с
помощью указателя на функцию */
float root(fptr f, float a, float b, float e)
{
      float x;
      do
      {
            x=(a+b)/2;            //находим середину отрезка
            if ((*f)(a)*f(x)<0)         //выбираем отрезок
            b=x;
            else a=x;
      }
      while((*f)(x)>e && fabs(a-b)>e);
      return x;
}

//функция, для которой ищется корень
float testf(float x)
```

```
{
      return x*x-1;
}

void main()
{
      /*в функцию root передается указатель на функцию, координаты отрезка и
      точность */
      float res=root(testf,0,2,0.0001);
      cout<<”\nX=”<<res;
}
```

3. Постановка задачи

1. Используя ввод-вывод в стиле C/C++ создать файл и записать в него структурированные данные.
2. Вывести созданный файл на экран.
3. Написать функцию, которая позволяет удалить из файла данные.
4. Написать функцию, которая позволяет добавить в файл данные.
5. Вывести содержимое измененного файла на экран.
6. Решить Задачу 1 согласно варианту.
7. Считать значение строки с клавиатуры.
8. Записать строку в файл (синтаксис C), организовать с помощью функций.
9. Обработать строку согласно варианту (Задача 2) Для обработки строк использовать стандартные функции из библиотечного файла <string.h>.
10. Вывести измененную строку на экран, организовать с помощью функций
11. Записать измененную строку в файл (синтаксис C).
12. Написать функцию с умалчиваемыми параметрами в соответствии с вариантом, продемонстрировать различные способы вызова функции:
    a. с параметрами заданными явно,
    b. с опущенными параметрами
    c. часть параметров задана явно, а часть опущена.
13. Написать функцию с переменным числом параметров в соответствии с вариантом, продемонстрировать вызов функции с различным числом параметров.
14. Написать перегруженные функции в соответствии с вариантом. Написать демонстрационную программу для вызова этих функций (продемонстрировать на массивах с различными типами данных).
15. Написать шаблон функций вместо перегруженных функций из задания 5. Написать демонстрационную программу для вызова этих функций.
16. Решить уравнение указанным в варианте методом. Уравнение передать в функцию как параметр с помощью указателя.

4 Варианты

Таблица 1

№ варианта	Задача 1. массив структур (2 способами: стиль С и С++)		Задача 2. Строки
	Структура	Действие	
1	Структура "Автомобиль": - марка; - год выпуска; - цена; - цвет.	Подобрать автомобиль для покупателя, согласно заданных критериев.	Отсортировать слова в строке в лексикографическом порядке (по алфавиту).
2	Структура "Спортивная команда": - название; - город; - количество игроков; - количество набранных очков.	Отсортировать команды согласно рейтинга.	Перевернуть каждое четное слово в строке.
3	Структура "Студент": - фамилия, имя, отчество; - номер телефона; - группа;	Отсортировать студентов по фамилии.	Перевернуть предложение (последнее с первым словом поменять местами и т.д.).
4	Структура "Студент": - фамилия, имя, отчество; - дата рождения; - домашний адрес; - рейтинг	Организовать поиск по заданным пользователем критериям	Описать функцию, которая будет вычислять сумму кодов слова, отсортировать слова, в порядке возрастания этих сумм.

5	struct employee
	{
	char*name;
	float salary;
	int stage
	};
		Организовать поиск сотрудников со стажем меньше 3 лет	Удалить из строки все слова, заканчивающиеся на гласную букву.

6	Структура "Камера хранения": -фамилия, имя; -дата сдачи; -срок хранения; -инвентарный номер и название предмета	Отсортировать по фамилии, дате, просмотреть клиентов за последний месяц	Перевернуть каждое нечетное слово в строке.

Таблица 2

№ варианта	Задача 3. ***Функция с умалчиваемыми параметрами***	Задача 4. ***Функция с переменным числом параметров***	Задача 5. ***Перегруженные функции и шаблон функции***	Задача 6. ***Передача функции как параметра другой функции с помощью указателя***
1	Печать марки, год выпуска, цвет	Минимальный элемент в списке параметров	Среднее арифметическое массива	Метод итераций $3\sin\sqrt{x} + 0{,}35x - 3{,}8 = 0$ Отрезок, содержащий корень: [2;3] Точное значение: 2,2985
2	Печать название, город	Максимальный элемент в списке параметров	Количество отрицательных элементов в массиве	Метод Ньютона $3\sin\sqrt{x} + 0{,}35x - 3{,}8 = 0$ Отрезок, содержащий корень: [2;3] Точное значение: 2,2985
3	Печать фамилии, курса и группы	Количество четных элементов в списке параметров	Максимальный элемент в массиве	Метод итераций $3\sin\sqrt{x} + 0{,}35x - 3{,}8 = 0$ Отрезок, содержащий корень: [2;3] Точное значение: 2,2985
4	Печать фамилии, имени и рейтинга	Среднее арифметическое элементов в списке параметров	Минимальный элемент в массиве	Метод хорд $0{,}1x^2 - x \ln x = 0$ Отрезок, содержащий корень: [1;2] Точное значение: 1,1183
5	Печать фамилии, стаж и зарплата	Максимальный из элементов в списке параметров, стоящих на четных местах	Сортировка массива методом простого обмена	Комбинированный метод $0{,}25x^3 + x - 1{,}2502 = 0$ Отрезок, содержащий корень: [0;2] Точное значение: 1,0001
6	Печать фамилии, срок хранения, название	Максимальный из элементов в списке параметров, стоящих на нечетных местах	Сортировка массива методом выбора	Метод Ньютона $\sqrt{1 - 0{,}4x^2} - \arcsin x = 0$ Отрезок, содержащий корень: [0;1] Точное значение: 0,7672

5. Методические указания

1. Для выделения памяти под массивы использовать операцию new, для удаления массивов из памяти – операцию delete.
2. Для формирования и печати структур (клавиатура/экран) написать отдельные функции:

```
person make_person()
{
      int Age; char Name[20];
      cout<<"Name?";
      cin>>Name;
      cout<<"Age?";
      cin>>Age;
      person p;
      p.name=new char[strlen(Name)+1];
      strcpy(p.name,Name);
      p.age=Age;
```

```
    return p;
}
void print_person(person p)
{
    cout<<"\nName:  "<<p.name<<"\t"<<"Age:  "<<p.age;
}
```

3. Для выделения памяти, заполнения массивов, поиска заданных элементов, заполнения, просмотра, добавления и удаления данных из файлов написать отдельные функции. В функции main() должны быть размещены только описания переменных и обращения к соответствующим функциям.
4. Если отсутствуют элементы, соответствующие критерию поиска, то должно быть выведено сообщение о том, что требуемые элементы не найдены.

6. Содержание отчета

1. Постановка задачи (общая и для конкретного варианта).
2. Определения функций для реализации поставленных задач.
3. Определение функции main().
4. Содержимое исходного файла
5. Содержимое модифицированного файла.