“C/C++” — кратко обо всём

Роман Савоченко
rom_as@diyaorg.dp.ua

19 июля 2005 г.

Оглавление

1 Язык "C/C++" 4

1.1 Ключевые слова C/C++ . . . 4
1.2 Постоянные . . . 4
1.3 Переменные . . . 5
1.3.1 Основные типы переменных . . . 5
1.3.2 Перечислимый тип (enum) . . . 5
1.3.3 Структура (struct) . . . 5
1.3.4 Тип объединение (union) . . . 5
1.3.5 Пустой тип (void) . . . 5
1.3.6 Сокращенный тип (typedef) . . . 5
1.3.7 Тип класс (class) . . . 5
1.4 Синтаксис . . . 7
1.4.1 Полезные обороты . . . 7
1.4.2 Указатели . . . 8
1.4.3 Ссылки . . . 8
1.4.4 Массивы . . . 8
1.4.5 Перегрузка функций . . . 9
1.4.6 Перегрузка операций . . . 9
1.4.7 Шаблоны . . . 9
1.4.8 Обработка исключительных ситуаций . . . 9
1.5 Операции . . . 10
1.6 Операторы . . . 10
1.6.1 C . . . 10
1.6.2 C++ . . . 11
1.7 Спецификаторы класса памяти . . . 11
1.8 Препроцессор . . . 12
1.8.1 Директивы Препроцессора . . . 12
1.9 Стандартные заголовочные файлы . . . 13

2 Основные функции языка "C/C++" 15

2.1 Математические функции (math.h|cmath) . . . 15
2.2 Функции для работы с дисками, директориями и файлами . . . 15
2.2.1 Функции потокового ввода-вывода . . . 15
2.2.2 Работа с директориями . . . 15
2.2.3 Доступ к файлам . . . 16
2.2.4 Функции работы с временными файлами . . . 17
2.3 Функции поддержки переменного числа параметров <stdarg.h> . . . 17
2.4 Функции/переменные работы со временем и временными интервалами . . . 17
2.5 Функции проверки и преобразования символов <ctype.h, cctype> . . . 18
2.6 Функции работы со строками . . . 18
2.7 Потоковая обработка строк (string) <string> . . . 19
2.7.1 Функции . . . 19
2.8 Стандартная библиотека шаблонов (STL) . . . 20
2.8.1 Контейнеры . . . 20
2.8.2 Итераторы . . . 22
2.8.3 Алгоритмы . . . 22
2.8.4 Класс <bitset> . . . 25
2.8.5 Объекты-функции . . . 25
2.9 функции работы с памятью . . . 26
2.10 Специальные функции . . . 26
2.10.1 Работа с терминалом . . . 27
2.10.2 Работа с динамическими библиотеками . . . 27
2.10.3 Лимитирование . . . 27
2.10.4 Документирование и ведение логов . . . 27
2.10.5 Функции управления безопасностью . . . 27
2.11 Потоковые функции языка C++ . . . 28
2.11.1 Манипуляторы потока . . . 28
2.11.2 Компонентные функции класса ios <iostream> . . . 28
2.11.3 Компонентные функции класса ostream <iostream> . . . 29
2.11.4 Компонентные функции класса istream <iostream> . . . 29
2.12 Процессы. . . . 30
2.13 Сигналы. . . . 30
2.14 Потоки . . . 31

2.14.1 Отмена потоков . . . 32
2.14.2 Глобальные данные потоков . . . 32
2.14.3 Обычные потоковые семафоры . . . 32
2.14.4 Исключающие семафоры . . . 32
2.14.5 Сигнальные переменные . . . 32
2.15 IPC . . . 32
2.15.1 Сообщения . . . 32
2.15.2 Семафоры . . . 33
2.15.3 Разделяемая память . . . 33
2.16 Каналы . . . 33
2.17 Сокеты . . . 33

Глава 1

Язык “C/C++”

1.1 Ключевые слова C/C++

Ключевые слова для C/C++			
auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while
Ключевые слова только для C++			
asm	bool	catch	class
const_cast	delete	dynamic_cast	explicit
false	friend	inline	mutable
namespace	new	operator	private
protected	public	reintepret_cast	static_cast
template	this	throw	true
try	typeid	typename	using
virtual	wchar_t		

1.2 Постоянные

Целые константы	
Десятичные	цифры 0-9;(12, 111, 956)
Восьмеричные	цифры 0-7;(012=10, 011=73, 076=62)
Шестнадцатеричные	цифры 0-9,буквы A-F или a-f (0x12=18, 0x2f=47, 0XA3=163)
Длинные целые константы	
Десятичная	12l=12, 956L=956;
Восьмеричные	012l=10, 076L=62;
Шестнадцатеричные	0x12l=18, 0XA3L=163.
Беззнаковые целые константы	
Десятичная	12u=12, 956U=956;
Восьмеричные	012u=10, 076U=62;
Шестнадцатеричные	0x12u=18, 0XA3U=163.
Константы с плавающей точкой	
Всегда представляются типами float, double и long double: 345. = 345; 2.1e5 = 210000; .123E3fL = 123; 4037e-5l = .04037	
Символьные константы	
Состоит из одного символа кода ASCII, заключеного в апострофы: 'A'; 'a'; '7'; '$'. Многобайтовые символы: L'ab'. Специальные символы:	
`\a`	звонок
`\b`	возврат на один символ назад
`\f`	перевод страницы
`\n`	новая строка
`\r`	перевод каретки
`\t`	горизонтальная табуляция
`\v`	вертикальная табуляция
`\'`	апостроф
`\"`	двойные кавычки
`\\`	обратная дробная черта
`\?`	вопросительный знак
`\23`	символ задан десятичным числом
`\0x23`	символ задан шестнадцатеричным числом
`\023`	символ задан восьмеричным числом
Строковые константы (литералы)	
Представляет последовательность символов кода ASCII, заключённой в кавычки: "строка".	

1.3 Переменные

1.3.1 Основные типы переменных

Тип (байт_сист)	Диапазон значений
bool (1)	false=0; true=1
char (1)	-128...127
int (2_16)	-32.768...32.767
int (4_32)	$-2^{31}\ldots2^{31}$
short	-32.768...32.767
long (4_32)	$-2^{31}\ldots2^{31}$
long (8_64)	$-2^{63}\ldots2^{63}$
long long (8)	$-2^{63}\ldots2^{63}$
unsigned char (1)	$0\ldots255$
unsigned (2_16)	$0\ldots65535$
unsigned (4_32)	$0\ldots2^{32}$
unsigned short (2)	$0\ldots65535$
unsigned long (4_32)	$0\ldots2^{32}$
unsigned long (8_64)	$0\ldots2^{64}$
unsigned long long (8)	$0\ldots2^{64}$
enum (2_16)	32.768...32.767
enum (4_32)	$-2^{31}\ldots2^{31}$
float (4)	$3.4*10^{38}$ (7 знаков)
double (8)	$3.4*10^{308}$ (15 знаков)
long double (12)	$3.4*10^{4932}$ (19 знаков)

1.3.2 Перечислимый тип (enum)

Определяет тип enum и-или enum переменную. Если фигурные скобки заданы то ключевое слово enum объявляет тип enum, состоящий из набора именованных целочисленных констант. Переменная типа enum хранит одно из значений, определенных типом enum. Таким образом, enum переменная всегда имеет тип int. Enum может использоваться, для объявления констант, которые могут использовать имена без объявления переменных для них, как показано в следующем примере:

```
enum DAYS
{
  saturday,
  sunday = 10,
  monday,
  tuesday,
  wednesday,
  thursday,
  friday
} today;
enum DAYS yesterday = monday;
DAYS yesterday = monday;     // C++
int tomorrow = wednesday;
```

1.3.3 Структура (struct)

Объявляет тип структуры и-или переменную структуры. Если заданы фигурные скобки то определяется структурный тип. Безымянные разрядные поля могут использоваться для выравнивания.

Если фигурные скобки не заданы то ключевое слово *struct* используется для определения переменной структуры. В языке C++ добавлена возможность использования в структуре ключевых слов public, private, protect:

```
struct my_str;  //my_str - как прототип, определена позже
struct POINT
{
   int x;
   int y;
} here = { 20, 40 };
struct POINT there, *ther1;
ther1->x = 2;
struct CELL  { //Выбор количества битов для элементов
    unsigned character  : 8;   // 00000000 ????????
    unsigned foreground : 3;   // 00000??? 00000000
    unsigned intensity  : 1;   // 0000?000 00000000
    unsigned blink      : 1;   // ?0000000 00000000
    unsigned :1                // заполнитель
} screen[25][80];
POINT example(POINT there)
{
  there.x = 3;
  there.y = 5;
};
```

1.3.4 Тип объединение (union)

Объявляет тип - объединение, и-или переменную объединения. Если фигурные скобки заданы то, union объявляет тип объединения, состоящий из последовательности переменных, значения (известных как элементы объединения) которых могут иметь различные типы. Переменная типа union может содержать один элемент любого типа, определенного объединением. Размером объединения является размер самого большого типа в объединении. Переменная может быть определена, указанием ее имени после заключительной фигурной скобки. Если фигурные скобки не даны, то ключевое слово union используется, для определения переменной объединения. Например:

```
union UNKNOWN
{
  char   ch;
  int    i;
  long   l;
  float  f;
  double d;
} var1;               // Variable of type UNKNOWN
union UNKNOWN var2;   // Variable of type UNKNOWN
var1.i = 6;           // Use variable as integer
var2.d = 5.327;       // Use variable as double
```

C++ Поддерживает анонимные объединения:

```
union
{
   int my_data;
   float this_data;
};
my_data=3
```

1.3.5 Пустой тип (void)

Если этот тип используется как тип возврата из функции то функция не возвращает значений. Если используется как список параметров функции, то входные параметры у функции отсутствуют. Если используется указатель на тип void то его при использовании необходимо приводить к конкретному типу.

1.3.6 Сокращенный тип (typedef)

Описание typedef используется для замены сложных типов данных или создания своих специфических типов данных:

```
typedef  unsigned long int ULINT;
ULINT    my_const;
```

1.3.7 Тип класс (class)

Общие понятия

В C++ добавлен класс - расширение понятия структуры. Память при определении класса не выделяется. Класс имеет имя (тег) и состоит из полей, представляющих его члены. В C++ допускается использование вложенных классов. Ключевое слово public определяет те члены класса, к которым имеется прямой доступ. Ключевое слово private используется для сокрытия определенных деталей класса, которые доступны только функциям членам класса и дружественным функциям. Все члены класса по умолчанию считаются приватными. Ключевое слово protected используется для открытия доступа только членам этого класса и членам производных от него классов. Функции класса могут определятся как внутри(увеличивается объем программы и скорость выполнения) так и вне(уменьшается объем программы и скорость выполнения) его тела. При создании в программе объекта экземпляра, его членам присваиваются некоторые начальные значения эту операцию выполняет специальная функция - конструктор имя которой совпадает с именем класса. Для освобождения памяти и других операций при закрытии класса используется деструктор имя которого совпадает с именем класса и с добавлением впереди символа "~".

Для присваивания переменной одного класса переменной другого класса можно в классе использовать оператор: `operator char *()` в котором описывается процедура преобразования одной переменной в другую что позволит в дальнейшем упростить обмен: title=big_book.

Одинаковые классы допускают копирования содержимого из одного класса в другой.

Конструкторы с одним параметром определённого типа могут использоваться для неявного преобразования типов (от типа параметра к типу класса). Для исключения этого нужно использовать директиву **explicit**.

```
class book
{
  public:
    char title[256];
    char author[64];
    book(char *title="A", char *autor="B",
         char *publisher= "C")
    {
       strcpy(book::title, title);
       strcpy(book::author, author);
       strcpy(book::publisher, publisher);
    };
    ~book(void);
    char *get_price(*publisher) {*publisher};
    char show_title(void);
  private:
    char publisher[256];
};
book diary;
book::~book(void)
{
  cout << ?Уничтожение экземпляра << title << '\n';  };
  void book::show_title(void) {cout << title << '\n'; };
  book tips(?Jamsa's 1001 C/C++?, ?Jamsa?, ?Jamsa Press?);
}
```

Дружественные классы и члены

Дружественные классы - friend указывает на класс который может использовать private члены текущего класса. Есть возможность узкого указания на член класса friend имеющего доступ к private членам текущего класса. Кроме того есть возможность создавать взаимные friend - классы. Дружественная функция может не принадлежать ни какому классу т.е быть автономной.

```
class book
{
  public:
    char title[256];
    char author[64];
    friend class Reader::show_reader(void);
  private:
    char publisher[256];
};
class Reader {
  public:
    Reader (char *name) {strcpy(Reader::name, name); };
    void show_reader(class book book)
    {cout<<"Читатель"<<name<<' '<< "Книга"<<book.title; };
    class book tips[23];
  private:
    char name[64];
};
```

Наследование

Наследование это когда производный класс наследует свойства родительского класса. Наследование обеспечивает возможность рассмотрения порождённого объекта как базового (но не наоборот). Наследование может быть множественным. В производном классе допускается переопределение функций базового. Для обращения к перегруженной функции базового класса можно записать **d.TBase::getData();**, где TBase - имя базового класса. Наследования бывают:

- **public** - Открытое наследование.
  `class circle: public point {};`
  При этом члены базового класса:
  - **public** - Доступны любым нестатическим функциям-членам и функциям не являющимся членами;
  - **protected** - Доступны любым нестатическим функциям-членам и дружественным функциям;
  - **private** - Невидимы в производном классе;
- **private** - Закрытое наследование.
  `class circle: private point {};`
  - **public** - Доступны любым нестатическим функциям-членам и дружественным функциям (делает public члены базового класса - private в производном);
  - **protected** - Доступны любым нестатическим функциям-членам и дружественным функциям (делает protected члены базового класса - private в производном);
  - **private** - Невидимы в производном классе;
- **protected** - Защищённое наследование.
  `class circle: protected point {};`
  - **public** - Доступны любым нестатическим функциям-членам и дружественным функциям (делает public члены базового класса - protected в производном);
  - **protected** - Доступны любым нестатическим функциям-членам и дружественным функциям;
  - **private** - Невидимы в производном классе;
- **virtual** - Используется для решения проблемы подобъектов-дубликатов (использование базового класса более раза).

```
class Cover
{
  public:
     static int count;
     Cover(char *title) { strcpy(Cover::title, title) ;};
  protected:
     char title[256];
};
class book
{
  public:
     book (char *title) {srcpy(book::title, title); };
     void show_title(void) {cout << title <<endl; };
  protected:
     float cost;
     void show_cost(void) {cout<<cost<<endl; };
  private:
     char title[64];
};
class LibraryCard : public Cover, public book {
  public:
     LibraryCard(char *title, char *author, char *publisher):
     book(title)
     {
        strcpy(LibraryCard::author, author);
        strcpy(LibraryCard::publisher, publisher);
        cost = 39.95;
     };
  private:
     char author[64];
     char publisher[64];
};
```

Полиморфизм

В классах поддерживается позднее(динамическое) связывание посредством механизма виртуальных функций. Динамическое связывание(определение адресов вызываемых в программе функций) происходит во время выполнения программы. В программах могут использоваться объектные переменные, или объектные указатели, значения которых - указатели на объекты-экземпляры того

или иного класса. В языке C++ разрешается использовать объектный указатель базового класса для указания объекта производного класса. В языке C++ полиморфизм обеспечивается использованием механизма виртуальных функций. Для обращения к членам базового и производного класса имеющим одинаковые имена, используется определение виртуальной функции - virtual, что заставляет обращаться к члену последнего активизированного класса. Для корректного удаления объектов из памяти можно создавать виртуальный деструктор который будет вызываться перед вызовом деструктора базового класса. Чистая виртуальная функция (приравнивается 0) является аналогом прототипа, который объявляется в базовом классе а описывается в производном классе. Класс в котором хотя бы одна виртуальная функция приравнена 0 - является абстрактной. Абстрактным также является класс у которого деструктор преравнен к 0, но всёже определён (используется для исключения прямого создания объекта данного класса). Конкретным является класс в котором все чисто виртуальные функции базового класса переопределены:

```
class Base
{
  public:
    void base_mess(void) {cout<<"Base\n"; };
    virtual void show_mess(void) { cout<<"Base";};
    virtual void show_reserve(void) = 0;
};
class Der: public Base
{
  public:
    void der_mess(void){       };
    virtual void show_mess(void) { cout<<"Der";};
    virtual void show_reserve(void){ cout<<"           ";};
};
void main(void)
{
  Base *base_pointer = new Base;
  base_pointer->base_mess();
  base_pointer->show_mess();
  base_pointer = new Der;
  base_pointer->der_mess();
  base_pointer->show_mess();
}
```

Инициализаторы

Для инициализации членов данных из конструктора можно использовать инициализаторы. Которые являются единственным способом инициализации константных членов класса:

```
class Time
{
  public:
    Time();
    const int time;
};
Time::Time() : time(10)
{...};
```

Композиция

Классы допускают композицию т.е включение одного объекта в другой. Включенные объекты уничтожаются после уничтожения содержащего их объекта.

```
class Time
{
  public:
    Time();
    const int time;
};
class Date
{
  public
    Date();
  private
    const Time time;
}
Date::Date : time()
{...};
```

Ссылка на себя

В классах есть возможность ссылаться на себя. Эта функция обеспечивается ключевым словом <this> которое содержит адрес текущего объекта. Может использоваться сцепления путём возврата адреса или ссылки объекта его членами функциями.

Proxy классы

Proxy классами называются классы которые призваны скрывать private члены классов закрытой реализации библиотеки. Создаются они путем создания указателя на скрываемый класс в private поле proxy класса:

```
class Sequry
{
  public
    void setValue(int x);
  private:
    int value;
};
class Proxy
{
  public
    setValue(int x) {ptr->setValue(x);};
  private
    Sequry *ptr;
}
```

1.4 Синтаксис

1.4.1 Полезные обороты

- Приведение типов:
  `long c = (long)a;` - традиционная запись;
  `long c = long(a);` - функциональная запись;

- Декларация нескольких переменных одного типа:

```
int x,     // x
    y,     // y
    z;     // z
```

- Варианты использования параметров функций:
  - для меняющихся аргументов необходимо использовать ссылки или указатели (ссылки на массив);
    `int test(int *ptr, int &alias, int[] &mass );`
  - для неизменяемых аргументов использовать прямую передачу по значению ;
    `int test(int value);`
  - для больших неизменяемых аргументов использовать константные ссылки или указатели на константные данные;
    `int test(const int &value, const int *ptr);`
- Использование ключевого слова const (запрет модификации):
  - Неконстантный указатель на неконстантные данные;
    `void test(char *str);`
  - Неконстантный указатель на константные данные;
    `void test(const char *str);`
  - Константный указатель на неконстантные данные;
    `int * const ptr = &x;`
  - Константный указатель на константные данные;
    `const int * const ptr = &x;`
  - Константный класс. Не позволяет запускать неконстантные функции члены, кроме конструктора и деструктора;
    `const Time moon(12,0,0);`
  - Константная функция. Недопускется ничего (кроме mutable членов) модифицировать в константных функциях, а также вызывать неконстантные функции;
    `int getValue() const;`
- Предотвращение многократного включения заголовочных файлов:

```
#ifndef TEST_H
#define TEST_H

<code>

#endif
```

- Коментирование/исключение больших кусков кода:

```
#if 0
  <code>
#endif
```

- Использование значений параметров функций по умолчанию:

```
void show_values(int one=1, int two=2, int three=3);
show_value();
show_value(23);
```

- Объявление переменных по месту:

```
for (int count = 0; count < 10;count++)
```

- Функции переменного числа параметров:

```
void text(parm x,char *fmt, ...)
{
  char str[100];

  va_start (argptr,fmt);
  vsprintf(str,fmt,argptr);
  va_end(argptr);
  printf(str);
}

text(54,"hello %s","world");
```

- Указание способа(языка) для которого должна компилироваться функция:

```
extern "C" func();  // В стиле "C"
extern "C++" func(): // В стиле "C++"
extern "C++"
{
 int func(void);
}
```

1.4.2 Указатели

Особенностью языка "C/C++" является возможность доступа к переменной не только по имени но и с использованием механизма указателей. Для этого в языке предусмотрены символы: "&" и "*".

Символ "*" используется для индикации переменной (*ptr), которая расположена в памяти по адресу на который указывает одноимённая переменная без звёздочки. Символ "&" используется как для определения адреса ячейки памяти переменной, так и для определения адреса указателя на переменную.

- Назначение адреса указателя

```
int *ptr = (int *)0x0010;}   //при инициализации
ptr      = (int *)0x0010;}   //в программе
```

- Присвоение значения непосредственно переменной на которую указывает указатель:

```
*ptr = 300
```

- Родовой указатель в "C/C++" основан на использовании указателя типа **(void *)**. Ключевое слово **void** говорит об отсутствии данных о размере объекта в памяти. Во всех случаях использования указателя описанного как **void***, необходимо выполнять операцию явного приведения типа указателя:

```
unsigned long block = 0xffeeddccL;
void *ptr = &block;
unsigned char = *(unsigned char *)ptr;
long int four_bytes = *(long int *)ptr;
```

- Определение адреса указателя:

```
int *ptr;
k = &ptr;
```

- Указатель на указатель:

```
int data = 5;
int *ptr = &data;    // ptr[0]==5;
int **ptr1 = &ptr;   // ptr1[0][0]==5;
int ***ptr2 = &ptr1; // ptr2[0][0][0]==5;
```

- Возврат указателя функцией:

```
bool *cimpare(int, int);
```

- Указатель на функцию:

```
bool (* compare)(int, int);
```

- Указатели на члены класса (.* и ->*)

```
class Test
{
  public:
    void funct() { cout << "Функция\n"; }
    int value;
};

Test t;
Test *tPtr = &t;
void (Test::*memPtr)() = &Test::funct;
int  Test::*vPtr = &Test::value;
(tPtr->*memPtr)(); //косвенный вызов функции
cout << (*tPtr).*vPtr << endl;
```

1.4.3 Ссылки

Ссылка являются псевдонимом (алиасом) от переменной на которую она ссылается. При изменении ссылаемой переменной изменяется ссылка. В основном ссылки используются при описании параметров функций и указывают что переменная может меняться.

```
int &test(int &x);
int data = 5;
int &al_data = data;    // al_data == 5;
al_data = 10;           // data == 10;
data = 7;               // al_data == 7;
```

1.4.4 Массивы

Как и в других языках "C/C++" поддерживает массивы которые тесно переплетаются с указателями. Элементы массива имеют один и тот же тип и расположены в памяти друг за другом. Имя массива также можно воспринимать как указатель на начало массива.

В отличие от других языков в "C/C++" отсутствует специальный строковый тип. Вместо него строковые литералы представляются как одномерный массив элементов типа char оканчивающегося символом "0".

- Явное указание числа элементов массива и списка начальных значений:

```
char array[]  = {'A','B','C','D',0};
char array[]  = "ABCD";
char array[5] = {'A','B','C','D',0};
char *string  = "ABCD";
string        = "ABCD";
```

- Обращение к элементам массива с помощью указателя:

```
*(array+i);
array[i][j];
*(array[i]+j);
*(*(array+i)+j);
```

- Многомерные массивы:

```
matrix[2] == &matrix[2][10];
long (* matrix1)[3][2][4];
matrix1 = new long[3][2][4];
char *messages[20] == char messages[20][80]);
char string[][80]=
{
  "Первая строка",
  "Вторая строка",
  "Следующая строка"
};
int m[][3] = { {00}, {10, 11}, {20, 21, 22,} };
char *Names[]= { "Aleksey", "Vladislav", "Vitaly" };
```

- Массив указателей на функцию:

```
int  (* fcmp[5])(int) =
{cmp_name, cmp_title, cmp_year, cmp_price, cmp_totaly};
void (* func[3])(int);    //определение
(* func[choice])(choice); //вызов
```

1.4.5 Перегрузка функций

В языке C++ разрешается иметь множество функций с одним и тем же именем, но отличающиеся типами параметров или их количеством:

```
int sum(int *array, int element) {   }
float sum(float *array, int element) {   }
```

1.4.6 Перегрузка операций

Перегрузкой операция является процедура расширения функций существующих операция для новых типов x(объектов). Операции допускающие перегрузку указанны в табл.1.1. При перегрузке операций их старшинство и ассоциативность не изменяется.

- Унарная операция (префиксная).
  Операция:
  !S
  Вызывает:
  S.operator!()
  operator!(S)
  Объявляется:
  bool operator!() const;
  friend bool operator!(const String &);
- Унарная операция (постфиксная).
  Операция:
  d1++
  Вызывает:
  d1.operator++(0)
  operator++(d1,0)
  Объявляется:
  Date operator++(int);
  friend Date operator++(Date &,int);
- Бинарная операция.
  Операция:
  y+=z
  Вызывает:
  y.operator+=(z)
  operator+=(y,z)
  Объявляется:
  const String &operator+=(const String &);
  friend const String &operator+=(String &, const String &);
- Вызов функции.
  Операция:
  string(2,2);
  Вызывает:
  string.operator()(2,2);
  Объявляется:
  String operator()(int,int);
- Приведение типов.
  Операция:
  *(char *)S;*
  Вызывает:
  S.operator char()*
  Объявляется:
  String operator char();*
  *String(char *);* - конструктор неявного преобразования.
- операция 'new'.
  Операция:
  *Class1 *cls = new Class1;*
  *Class1 *cls = new ("class") Class1;*
  Вызывает:
  *Class1 *cls = Class2.operator new(sizeof(Class1));*
  *Class1 *cls = Class2.operator new("class", sizeof(Class1));*
  Объявляется:
  void Class2::operator new(size_t size);*
  void Class2::operator new(string modul, size_t size);*
- операция 'delete'.
  Операция:
  delete cls;
  Вызывает:
  Class2.operator delete(cls);
  Объявляется:
  void Class2::operator delete(void *addr);*

1.4.7 Шаблоны

Шаблоны определяются с помощью ключевого слова **template** и предназначены для определения функций и классов способных работать с различными типами входных и выходных параметров. Шаблоны и наследование связаны следующим образом:

- шаблон класса может быть производным от шаблонного класса;
- шаблон класса может являться производным от нешаблонного класса;
- шаблон класса может быть производным от шаблона класса;
- нешаблонный класс может быть производным от шаблона класса;

Шаблонные классы:

- Объявление:

```
template <class Templ>
class Tree
{
  public:
    Tree( const Templ& n );
    insertN(const Templ &);
}
template <class Templ>
Tree<Templ>::Tree(const Templ& n) { };
```

- Использование:

```
Tree<int> NewTree(23);
Tree<float> NewTree(56.8);
```

Шаблонные функции:

- Объявление:

```
template <class T> // или template <typname T>;
T max(T val1, T val2, T val3)
{
  T max = val1;
  if(val2 > max) max=val2;
  if(val3 > max) max=val3;
  return max;
}
```

- Использование:

```
int rez = max(1,10,3);
float rez = max(0.5,9.99,6.78);
```

Шаблоны и друзья:

```
friend void f1();    //друг любого класса
friend void f2(x<T> &); //друг конкретного класса
friend void A::f4(); //друг любого класса
friend void C<T>::f5(x<T> &); //друг конкретного класса
friend class Y;      //класс Y дружественен любому классу
friend class Z<T>;   //класс Y дружественен конкретному классу
```

1.4.8 Обработка исключительных ситуаций

В языке C++, добавлен мощный аппарат обработки исключительных ситуаций. Этот аппарат позволяет отлавливать как все типы исключений, так и конкретно взятый тип исключений. Так если записать `catch(...)`, то будут отлавливаться все типы исключений. Кроме того обработка исключительных ситуаций оказываться вынесенной из "основной линии"выполнения программы. Для генерации повторных исключений в **catch** опускаться использование

Таблица 1.1: **Операции допускающие перегрузку**

+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	< <	> >	> >=
< <=	==	!=	<=	>=	&&	||	++
- -	->*	,	->	[]	()	new	delete
new[]	delete[]						

(throw) без параметров. Если происходит глубоковложенное исключение, то выполняется "раскрутка"стека для возвращения до ближайшего **catch** Обработка исключительных ситуаций описывается следующим образом:

```
try  //начало блока испытания
{
  if() throw MyType();    // Принудительная генерация
                          //  (точка генерации)
} catch(MyType &Mt) {...}; // Отлов и обработка в блоке
                          //  перехвата
```

Для ограничения круга генерируемых исключений функцией, можно указать спецификацию исключений для функции:

```
int g(double h) throw(a,b,c) //может генер. a,b,c и
     //unexpected
int g(double h) throw()      //не может генерировать
        //(кроме unexpected)
int g(double h)              //может генерировать все
```

Стандартные обработчики исключений:

- bad_alloc (exception) - Ошибка выделения памяти с помощью new;
- bad_cast (exception) - генерируется dynamic_cast;
- bad_exception (exception) - Неожиданое исключение генерируется в случае неожиданного исключения при включении std::bad_exception в throw-список функции;
- bad_typeid (exception) - генерируется typeid;
- invalid_argument (exception) - Функции передан недопустимый аргумент;
- logic_error (exception) - исключения в логических операциях;
- length_error (exception) - Длина более максимально допустимой;
- out_of_range (exception) - Аргумент вне допустимого диапазона;
- runtime_error (exception) - Ошибка в программе;
- overflow_error (runtime_error) - Математическая ошибка переполнения сверху;
- underflow_error (runtime_error) - Математическая ошибка переполнения снизу;

Чтоб избежать утечек памяти, обусловленных забыванием вызова delete после new, можно использовать шаблон auto_ptr, который будет автоматически разрушаться.

На базе класса exception, можно генерировать собственные исключения.

1.5 Операции

Символ	*Описание*	*Направл.*
Самый высокий приоритет		
::(унарн)	Область видимости	справа
::(бинар)	Область видимости	слева
()	Вызов функции	слева
[]	Выделение элемента массива	слева
->	Выделение элемента структуры адресуемой указателем	слева
.	Выделение элемента структуры или объединения	слева
->*	Обращение по адресу объекта к адресу функции объекта	слева
.*	Обращение через объект к адресу функции объекта	слева
++	Пост-приращение	справа
– –	Пост-декремент	справа
+	Унарный плюс	справа
-	Унарный минус	справа
!	Логическое отрицание	справа
~	Побитовое отрицание	справа
(тип)	Приведение типов < (float)i >	справа
*	Обращение по адресу переменной	справа
&	Определение адреса переменной	справа
sizeof	Определение размера, в байтах	справа
*	Умножение	слева
/	Деление	слева
%	Остаток от деления	слева
+	Сумма	слева
-	Разность	слева
«	Сдвиг влево	слева
»	Сдвиг вправо	слева
<	Меньше	слева
<=	Меньше и равно	слева
>	Больше	слева
>=	Больше и равно	слева
==	Равно	слева
!=	Неравно	слева
&	Поразрядный "И"	слева
^	Поразрядный "исключающий ИЛИ"	слева
|	Поразрядный "ИЛИ"	слева
&&	Логический "И"	слева
||	Логический "ИЛИ"	слева
?:	Условная операция *int i=(val>=0)?val:-val;*	справа
=	Присваивание	справа
+= -= *= /= %= |= &= ^ <= >=	Составное присваивание	справа
,	Операция последования (последовательность выполнения)	слева

1.6 Операторы

1.6.1 C

■ **break**
Прекращает выполнение ближайшего внешнего оператора: do, for, switch или while.
for(;;) { printf("TEST"); break; }

■ **case**
Оценивает <выражение> и выполняет любое утверждение, связанное с <постоянной-выражением>. Если не имеется никакого соответствия с постоянным выражением, утверждение, связанное с заданным по умолчанию ключевым словом выполняется. Если заданное по умолчанию ключевое слово не используется, управление переходит к утверждению после блока переключателя.

■ **continue**
for(i=0;i<2;i++){printf("test"); continue; exit(1);}
Передает управление в начало оператора цикла do, for, или while вызывая следующую итерацию.

■ **default**

Используется в **switch** для выполнения действий если не выполняется не одно из условий.

■ **do**
do оператор while(выражение);
Выполняет <оператор>, пока <выражение> истинно. Условие проверяется в конце цикла.

■ **else**
Выполняет <выражение1>, если <выражение> истинно (отлично от нуля); если else присутствует, и <выражение> - ложно (нуль), выполняется <выражение2>. После выполнения <выражение1> или <выражение2>, управление переходит к следующему утверждению.

■ **for**
for(init-expr; cond-expr; increment) оператор;
Выполняет <оператор>, пока инициализированное число <init-expr>, над которым производится операция <increment>, удовлетворяет условному выражению <cond-expr>.

■ **goto**
Управления безусловно передается на оператор с меткой <имя:>.

■ **if**
if (выражение) выражение1; [else выражение2;]
Выполняет <выражение1>, если <выражение> истинно (отлично от нуля); если else присутствует, и <выражение> - ложно (нуль), выполняется <выражение2>. После выполнения <выражение1> или <выражение2 >, управление переходит к следующему оператору.

■ **longjmp**
void longjmp(jmp_ buf env, int value);
Передает управление по адресу <env> с порядковым номером <value>.

■ **return**
Прекращает выполнение текущей функции и возвращает управление вызывающей программе, с возможностью передачи значения выражения.

■ **setjmp**
int setjmp(jmp_ buf env);
Запоминает в <env> адрес текущего места в программе для дальнейшего возврата к нему посредством longjmp. Функция возвращает порядковый номер сохраненного адреса начиная с "0".

■ **switch**
switch(выражение){
case константа1: выражение1; [break;]
case константа(n): выражение(n); [break;]
[default: выражение(n+1);]
}
Сравнивает <выражение> с константами во всех вариантах case и выполняет оператор связанный с <постоянной-выражением>. Если отсутствует соответствия с постоянных с выражениями то выполняется утверждение, связанное с ключевым словом default. Если ключевое слово default не используется то управление переходит к утверждению после блока переключателя.

■ **while**
while(выражение) оператор;
Выполняет <оператор>, пока <выражение> истинно.

1.6.2 C++

■ **explicit**
Ставится перед конструктором объекта и предотвращает использование конструктора для неявного преобразования типов.

■ **namespace**
Назначение области действия имён:
namespace Example { int myvar; }
k = Example::myvar;

■ **this**
Содержит указатель на собственный объект класса (т.е на себя);

■ **typeid**
Возвращает ссылку на объект **type_info**. Объект **type_info** - это поддерживаемый системой объект, представляющий тип.
const char *dataType = typeid(T).name();

■ **typename**
Указывает, что следующий за ним идентификатор обозначает тип. Обычно используется внутри шаблонов.

■ **using**
Делает глубоковложенную команду, со специальной областью видимости, видимой по умолчанию:
using std::cout;
cout « "test";
using namespace math;

Операторы преобразования типов

■ **static_cast**
Выполняет стандартные преобразования (статическое):
int x = static_cast<int>(d);
string s = static_cast<string>("ch_string");
derivedPtr = static_cast<DerivedClass *>(basePtr); (преобразование потомка к базовому классу)

■ **dinamic_cast**
Выполняет динамическое приведение, иными словами, в процессе выполнения с проверкой возможности приведения. Можно использовать для программной проверки типа наследника из базового класса.
cylinderPtr = dynamic_cast<const Cylinder *>(shapePtr);
if(cylinderPtr != NULL) shapePtr->area(); //Цилиндр

■ **const_cast**
Используется для снятия константности или volatile путём приведения:
const_cast<const CastTest *>(this)->number–; (при вызове из константной функции)

■ **reinterpret_cast**
Для нестандартных преобразований:
count « *reintepret_cast<char *>(ptr) « endl; (преобразует (int *) в (char *) и разыменовует)

Ключевые слова-операторы

Стандарт C++ предоставляет ключевые слова-операторы, которые могут использоваться вместо нескольких операторов C++.

Оператор	*Слово-оператор*	*Описание*
Логические операторы-ключевые слова		
&&	and	логическое И (AND)
||	or	логическое ИЛИ (OR)
!	not	логическое НЕ (NOT)
Оператор "не равноключевое слово		
!=	not_eq	не равно
Побитовые операторы-ключевые слова		
&	bitand	побитовое И (AND)
|	bitor	побитовое включающее ИЛИ (OR)
^	xor	побитовое исключающее ИЛИ (OR)
~	compl	побитовое инвертирование разрядов
Побитовые операторы присваивания-ключевые слова		
&=	and_eq	побитовое И (AND) и присваивание
|=	or_eq	побитовое включающее ИЛИ (OR) и присваивание
^=	xor_eq	побитовое исключающее ИЛИ (OR) и присваивание

1.7 Спецификаторы класса памяти

■ **auto**
Указывающий, что переменная имеет локальную (автоматическую) протяженность. Разрушается при выходе из функции.

■ **register**
Определяет, что переменная должна быть ,если возможно, сохранена в машинном регистре.

■ **extern**
Определяет переменную которая должна быть видна в других

объектных модулях.

■ **mutable**
Определяет в классе переменную которую можно модифицировать даже из константной функции.

■ **static**
Объявляет переменную которая будет являться одной для всех экземпляров функции и которая инициализируется при запуске программы. Статические функции объектов определяются в одном экземпляре и становятся видимыми за пределы класса владельца. Кроме того статическая функция член не может являться константной и имеют доступ только к статическим полям. Static в глобальном отношении может использоваться для ограничения области действия переменной пределами одного файла.
static int getCount();

1.8 Препроцессор

Препроцессор это часть языка C. Препроцессор считывает исходный код и, отвечает на размещенные в нем директивы, производит модифицированную версию этого исходного кода, которая передаётся компилятору. Подстановки препроцессором выполняются рекурсивно, т.е выполняется многопроходная обработка.

1. Определение/удаление макроса:

```
#define VAL_T 345
#undef VAL_T
#define VAL_T 234
```

Макрос может быть определён без значения и может использоваться в качестве проверяемого флага.

2. Для перевода на другую строку используется ;\ :

```
#define FILLSCREAN(color) _AX = 0x0600;\
                          _CX = 0x0000;\
                          _DX = 0x184f;\
                          _BH = color;\
                          geninterrupt(0x10);
```

3. Использование параметров:

```
#define min(a,b) ((a) < (b) ? (a) : (b))
result = min(44,uplimit);
// result = ((44) < (uplimit) ? (44) : (uplimit));
```

4. Для объединения формальных параметров используются символы ## :

```
#define var( i , j ) ( i##j )   // var( x , 6 ) = x6
```

5. Для преобразования фактического параметра в строку используется символ # который помещается перед формальным макропараметром:

```
#define TRACE(flag) printf(#flag "=%d\n", flag)
```

6. Вариантный макрос (...):

```
#define err(...) fprintf(stderr,__VA_ARGS__)
err("%s %d\n","The error code ",48);
// fprintf(stderr,"%s %d\n","The error code ",48);

#define errout(a,b,...) \
    fprintf(stderr,"File %s  Line %d\n",a,b); \
    fprintf(stderr,__VA_ARGS__)
errout(__FILE__,__LINE__,"Unexpected termination\n");
```

1.8.1 Директивы Препроцессора

■ **_Pragma**
_Pragma("GCC poison printf")
Оператор для вызова прагмы из тела макроса.

■ **#define**
#define ident ident1;
Заменяет все последующие определения <ident> на лексему <ident1> (совокупность лексем).

■ **#elif, #if**
#if expresion
#elif expresion1
#endif
Проверяет выражение <expresion>, связанное с директивами #if, или #elif , если выражение истинно (отличный от нуля), то выполняются следующие строки до директив условия или конца условий. Директива #elif является объединением директив #else и #if. Если имеется #else, то нижестоящие строки выполняются когда выражение в #if или #elif имеет нулевое значение. Нельзя использовать в качестве условия оператор sizeof, составные типы, float или enum типы.

■ **defined**
#if defined(VAR)
#elif !defined(NVAR)
#endif
Оператор проверки определённости, используется в паре с #if.

■ **#else**
Нижестоящие строки выполняются если выражение в #if, #ifdef, #ifndef или #elif имеет нулевое значение.

■ **#endif**
Указывает на конец условия.

■ **#error**
#error сообщение
Генерация сообщения ошибки на stderr и завершения процесса компиляции.

■ **#ifdef**
#ifdef <identifier>
Выполняет последующие строки до #endif, если <identifier> был ранее определен.

■ **#ifndef**
#ifndef <identifier>
Выполняет последующие строки до #endif, если <identifier> не был ранее определён.

■ **#include**
#include <filename>
Вставляет содержимое файла <filename> в текущий файл. Если путь к имени файла включен в двойные кавычки ,то поиск осуществляется внутри текущего каталога.

■ **#include_next**
#include_next <filename>
????

■ **#line**
#line n file
Изменяет внутренний номер строки компилятора на <n>, а также изменяет внутреннее имя файла на <file>. Текущий номер строки и имя файла доступны через константы препроцессора __LINE__ и __FILE__.

■ **#pragma**
#pragma directives
Инструктирует компилятор, о выполнение машинно-специфических возможностей, определенных параметром <directives> (таблица 1.2).

■ **#undef**
#undef identifier
Удаляет текущее определение <identifier>, который был предварительно определен директивой #define.

■ **#warning**
#warning сообщение
Генерация сообщения предупреждения на stderr и продолжение компиляции.

Таблица 1.2: Параметры директивы pragma, препроцессора

Имя	Назначение
pack	Определяет, как компилятор выравнивает данные при сохранении в памяти. Может также использоваться с, вталкиванием и выталкиванием параметров. #pragma pack(n) #pragma pack(push, n) #pragma pack(pop)

Таблица 1.3: Предопределённые символьные константы

Имя	Назначение
`__BASE_FILE__`	Полный путь к каталогу исходного файла;
`__CHAR_UNSIGNED__`	Указывает что символьный тип является безнаковым;
`__cplusplus`	Указывает что исходный код является программой на языке C++;
`__DATE__`	Дата компиляции исходного файла (строка);
`__FILE__`	Имя исходного файла (строка);
`__func__`	Имя текущей функции;
`__FUNCTION__`	—-//—-;
`__PRETTY_FUNCTION__`	Полное имя текущей функции. Для C++ включает имена классов;
`__INCLUDE_LEVEL__`	Глубина включения (include) файла;
`__LINE__`	Номер текущей строки исходного текста (целое число);
`__NO_INLINE__`	Указывает на отсутствие inline-функций;
`__OBJC__`	Программа на языке Objective-C;
`__OPTIMIZE__`	Назначен любой уровень оптимизации;
`__OPTIMIZE_SIZE__`	Оптимизация размера программы;
`__STDC__`	Компилятор соответствует правилам стандарта языка C;
`__TIME__`	Время компиляции исходного файла (строка);
`__VERSION__`	Полный номер версии;

1.9 Стандартные заголовочные файлы

- **assert.h | cassert (ANSI)**
Содержит макросы и информацию, для дополнительной диагностики, помогающей при отладке программы.

- **bios.h**
BIOS сервисные функции (INT 10).

- **conio.h**
Подпрограммы Ввода - вывода.

- **ctipe.h | cctype (ANSI)**
Символьная классификация.

- **direct.h**
Управление каталогами.

- **dos.h**
MS-DOS функции (INT 21).

- **env.h (POSIX)**
Содержит прототипы для окружений строковых функций.

- **errno.h (ANSI)**
Переменные, и утилиты обработки ошибок.

- **fcntl.h (POSIX)**
Флажки, используемые в функциях open и sopen.

- **float.h | cfloat (ANSI)**
Содержит предельные размеры переменных с плавающей точкой для данной системы.

- **graph.h**
Программы для работы с графикой низкого уровня.

- **io.h**
Низкоуровневый ввод-вывод.

- **limits.h | climits (ANSI)**
Содержит общие ограничения системы.

- **locale.h (ANSI)**
Содержит информацию для выполнения локализации ПО.

- **malloc.h**
Содержит прототипы функций распределения памяти.

- **math.h | cmath (ANSI)**
Содержит прототипы математических библиотечных функций.

- **memory.h**
Подпрограммы манипуляции с буферами.

- **serach.h**
Функции поиска и сортировки.

- **setjmp.h (ANSI)**
Прототипы Функций setjmp и longjmp (безусловного перехода)

- **sygnal.h (ANSI)**
Прототипы функций для работы с сигналами и описание самих сигналов.

- **stdarg.h (ANSI)**
Содержит макрокоманды для работы с функциями имеющими список параметров переменной длины.

- **stdio.h | cstdio (ANSI)**
Содержит прототипы стандартных библиотечных функций ввода-вывода и используемую ими информацию.

- **stdlib.h | cstdlib (ANSI)**
Содержит прототипы функций для преобразования чисел в текст, текста в числа, для выделения памяти, генерации случайных чисел и других полезных операций.

- **string.h | cstring (ANSI)**
Содержит прототипы функций для обработки строк стиля C.

- **strstream.h | strstream (ANSI)**
Для формирования строк через поток.

- **time.h | ctime (ANSI)**
Содержит прототипы функций для работы со временем и датами.

- **iostream**

Содержит прототипы функций стандартного ввода и вывода.

- **iomanip**
Содержит прототипы функций для операций с потоками, дающие возможность форматировать потоки данных.

- **fstream**
Содержит прототипы функций для операций с файловым вводом-выводом.

- **utility**
Содержит классы и функции, используемые многими заголовочными файлами стандартной библиотеки.

- **vector, list, deque, queue, stack, map, set, bitset**
Содержат классы которые реализуют контейнеры стандартной библиотеки.

- **functional**
Содержит классы и функции, используемые алгоритмами стандартной библиотеки.

- **memory**
Содержит классы и функции, используемые стандартной библиотекой для выделения памяти контейнерам стандартной библиотеки.

- **iterator**
Содержит классы для доступа к данным в контейнерах стандартной библиотеки.

- **algorithm**
Содержит функции для манипулирования данными в контейнерах стандартной библиотеки.

- **exception, stdexcept**
Содержат классы использующиеся для обработки исключительных ситуаций.

- **string**
Определения класса string из стандартной библиотеки.

- **sstream**
Прототипы функций выполняющих ввод из строк в память и наоборот.

- **locale**
Содержит классы и функции, используемые потоковой обработкой различных языков.

- **limits**
Содержит классы для определения интервалов значений численного типа данных для данной платформы.

- **typeinfo**
Содержит классы для идентификации времени выполнения.

Глава 2

Основные функции языка “C/C++”

2.1 Математические функции (math.h|cmath)

■ **ceil, ceilf, ceill (POSIX)**
double ceil(double x);
float ceilf(float x);
long double ceill(long double x);
Функции округления до наименьшего целого, не меньшего, чем аргумент.

■ **cos (POSIX)**
double cos(double x);
Возвращает значение косинуса x, где x - это значение в радианах.

■ **exp (POSIX)**
double exp(double x);
Возвращает значение числа 'e' возведенного в степень x.

■ **fabs, fabsf, fabsl (POSIX)**
double fabs(double x);
float fabsf(float x);
long double fabsl(long double x);
Абсолютное значение числа с плавающей точкой.

■ **floor, floorf, floorl (POSIX)**
double floor(double x);
float floorf(float x);
long double floorl(long double x);
Наибольшее целое значение, но не большее x.

■ **fmod (POSIX)**
double fmod(double x, double y);
Функция получения остатка от деления (с плавающей точкой).

■ **log (POSIX)**
double log(double x);
Возвращает натуральный логарифм x.

■ **log10 (POSIX)**
double log10(double x);
Возвращает десятичный логарифм x.

■ **matherr (ANSI)**
*int matherr(struct expection *error_info);*
Пользовательская функция обработки ошибок математических операций. Описание ошибки передается в указателе на структуру <error_info>

■ **pow (POSIX)**
double pow(double x, double y);
Возвращает значение x в степени y.

■ **rand, srand <stdlib.h>**
int rand(void);
void srand(unsigned int seed);
rand() - возвращает псевдослучайное число в диапазоне от нуля до RAND_MAX.
srand() - устанавливает свой аргумент как основу (seed) для новой последовательности псевдослучайных целых чисел, возвращаемых функцией rand().

■ **sin (POSIX)**
double sin(double x);
Возвращает значение синуса аргумента x, где x указан в радианах.

■ **sqrt (POSIX)**
double sqrt(double x);
Функция вычисления квадратного корня.

■ **tan (POSIX)**
double tan(double x);
Возвращает тангенс аргумента x, где x задан в радианах.

2.2 Функции для работы с дисками, директориями и файлами

2.2.1 Функции потокового ввода-вывода

■ **fopen, fdopen, freopen (ANSI POSIX) <stdio.h>**
*FILE *fopen(const char *path, const char *mode);*
*FILE *fdopen(int fildes, const char *mode);*
*FILE *freopen(const char *path, const char *mode, FILE *stream);*
fopen - открывает файл с именем path и связывает его с потоком.
fdopen - связывает поток с существующим описателем файла <fildes>.
freopen - открывает файл с именем path и связывает его с потоком stream. Исходный поток (если такой существовал) закрывается.

■ **fclose (ANSI) <stdio.h>**
*int fclose(FILE *stream);*
Закрывает поток <stream>.

■ **fgetc,getchar (ANSI) <stdio.h>**
*int fgetc(FILE *stream);*
int getchar(void);
Считывает очередной символ из потока <stream> или из stdin.

■ **fprintf (ANSI) <stdio.h>**
*int fprintf(FILE *stream, const char *format, ...);*
Осуществляет форматированный вывод в поток <stream>. Табл **2.2**

■ **fflush (ANSI) <stdio.h>**
*int fflush(FILE *stream);*
"Сбрасывает"буферы потока <stream>.

■ **perror (ANSI) <stdio.h>**
*void perror(const char *s);*
Выводит в стандартный поток ошибки сообщения, описывая ошибку, произошедшую при последнем системном вызове или вызове библиотечной функции.

2.2.2 Работа с директориями

■ **chdir, fchdir (POSIX) <unistd.h>**
*int chdir(const char *path);*
int fchdir(int fd);
Установка текущей директории <path>, <fd>.

■ **ftw, nftw <ftw.h>**
*int ftw(const char *dir, int (*fn)(const char *file, const struct stat *sb, int flag), int depth);*
*int nftw(const char *dir, int (*fn)(const char *file, const struct stat *sb, int flag, struct FTW *s), int depth, int flags);*

Таблица 2.1: **Символы управления форматированием**

Элемент	*Эффект* %[флаги][ширина][.точность][F | N | h | l]<тип>
	Флаги
0	Для чисел ширина поля будет заполнена слева нулям.
-	Производится выравнивание выводимого числа по левому краю в пределах выделенного поля. Правая сторона выделенного поля дополняется пробелами.
+	Выводится знак числа символом '-' или '+'
	Обозначает пропуск при вводе поля, определенного данной спецификацией. Введенное значение не присваивается ни какой переменной.
Пробел	Выводится пробел перед положительным числом и знак '-' перед отрицательным.
#	Выводится идентификатор системы счисления для целых: - 0 перед числом в восьмеричной с/с; - 0x или 0X в шеснадцатеричной с/с; - ничего для чисел в десятичной с/с.
	Ширина (воздействует только на вывод)
n	Определяет минимальную ширину поля в <n> символах. Если после преобразозования ширины недостаточно, выводится столько символов,сколько есть, с дополнением пробелами.
0n	Все тоже, но позиции слева для целого числа дополняются нулям.
	Число выводимых символов определяется значением соответствующей переменной.
	Точность (воздействует только на вывод)
ничего	Точность по умолчанию.
.0	Для d, i, o, u, x - точность по умолчанию. Для e, E, f - десятичная точка отсутствует.
.n	Для e, E, f не более <n> знаков после точки.
	Следующий аргумент из списка аргументов - точность.
	Модификатор (воздействует там, где применимо)
h	Перед d, i, o, u, x, X аргумент является - short int.
l	Перед d, i, o, u, x, X аргумент является - long int.
	Перед e, E, f, g, G аргумент является - double (только для scanf).
L	Длиное двойной точности.
F	Указатель (FAR).
N	Указатель (NEAR).
	Поле образец - %[*][ширина] [образец]
Определяет множество символов, из которых может состоять вводимая строка. Если в образце стоит символ '^ ', то вводится будут все символы кроме перечисленных: [a-z],[A-F0-9]; [^ 0-9]	
	Тип переменной: **char**
c	При вводе, читается и передается переменной один байт. При выводе - байт переменной преобразуется к типу char и записуется в выходной поток.
	Тип переменной: **int**
d	Десятичное int со знаком.
i	Десятичное int со знаком.
o	Восьмеричное int без знака.
u	Десятичное int без знака.
x	Шестнадцатеричное int без знака (0 - f).
X	Шестнадцатеричное int без знака (0 - F).
n	Указатель NEAR (только смещение).
p	Указатель FAR (сегмент[селектор]:смещение).
	Тип переменной: **float**
f	Значение со знаком в форме [-]dddd.dddd .
e	Значение со знаком в форме [-]d.ddde[+ | -]ddd .
E	Значение со знаком в форме [-]d.ddddE[+ | -]ddd .
g	Значение со знаком в формате 'e' или 'f' в зависимости от значения и специфицированной точности.
G	Значение со знаком в формате 'E' или 'F' в зависимости от значения и специфицированной точности.
	Тип переменной: **char ***
s	При вводе принимает символы без преобразования до тех пор, пока не встретится **'\n'** или пока не достигнута специфицированная точноч. При выводе выдаёт в поток все символы до тех пор пока не встретится **'\0'** или не достигнута специфицированная точность.

Перемещается по дереву каталогов, начиная с указанного каталога <dir>. Для каждого найденного элемента дерева вызываются: <fn()> с указанием полного имени этого элемента, указатель на структуру элемента <stat> и целое число. Функция nftw() выполняет то же самое, что и ftw(), только имеет еще один параметр, flags, то есть вызывает свои функции еще с одним параметром.

■ **getcwd (POSIX) <unistd.h>**
*char *getcwd(char *buf, size_t size);*
Копирует абсолютный путь к текущему рабочему каталогу в массив <buf>, имеющий длину <size>.

■ **mkdir (POSIX) <sys/stat.h, sys/types.h>**
*int mkdir(const char *pathname, mode_t mode);*
Создаёт каталог <pathname> с режимом <mode>.

■ **opendir, closedir (POSIX) <sys/types.h, dirent.h>**
*DIR *opendir(const char *name);*
*int closedir(DIR *dir);*
Открывает/закрывает поток каталога, соответствующий каталогу <name>.

■ **readdir (POSIX) <sys/types.h, dirent.h>**
*struct dirent *readdir(DIR *dir);*
Возвращает указатель <dir> на следующую запись каталога.

■ **rmdir (POSIX) <unistd.h>**
*int rmdir(const char *pathname);*
Удаляет каталог <pathname>.

2.2.3 Доступ к файлам

■ **access (POSIX) <unistd>**
*int access(const char *pathname, int mode);*
Проверка, имеет ли процесс <pathname> права <mode> на чтение(R_OK), запись(W_OK), выполнение(X_OK) или существование файла(F_OK).

■ **close (POSIX) <unistd.h>**
int close(int fd);
Закрывает открытый описатель файла <fd>.

■ **create (POSIX) <sys/types.h, sys/stat.h, fcntl.h>**
*int creat(const char *pathname, mode_t mode);*
Создаёт файл <pathname> и возвращает его описатель.

■ **dup, dup2 (POSIX) <unistd.h>**
int dup(int oldfd);
int dup2(int oldfd, int newfd);
dup - предоставляет новому описателю наименьший свободный номер.
dup2 - делает <newfd> копией <oldfd> (если это необходимо), закрывая newfd.

■ **fcntl (POSIX) <unistd.h, fcntl.h>**
int fcntl(int fd, int cmd);
int fcntl(int fd, int cmd, long arg);
*int fcntl(int fd, int cmd, struct flock * lock);*
Выполняет различные операции над файловым описателем fd.

■ **flock (BSD) <sys/file.h>**
int flock(int fd, int operation);
Устанавливает или снимает <operation> "мягкую"блокировку открытого файла <fd>.

■ **fsync, fdatasync (POSIX) <unistd.h>**
int fsync(int fd);
int fdatasync(int fd);
fsync - копирует все части файла, находящиеся в памяти, на устройство <fd>. fdatasync - тоже что и fsync, но без метаданных.

■ **fileno (ANSI) <stdio.h>**
*int fileno(FILE *stream);*
Возвращает описатель <stream>.

■ **lseek (POSIX) <sys/types.h, unistd.h>**
off_t lseek(int fildes, off_t offset, int whence);
Устанавливает позицию чтения/записи информации в файле.

■ **mknod (BSD) <sys/types.h, sys/stat.h, fcntl.h, unistd.h>**
*int mknod(const char *pathname, mode_t mode, dev_t dev);*
Создаёт файл (обычный файл, файл устройства или именованный

канал) <pathname>, с правами доступа <mode> и дополнительной информацией <dev>.

■ **open (POSIX) «sys/types.h, sys/stat.h, fcntl.h>**
*int open(const char *pathname, int flags);*
*int open(const char *pathname, int flags, mode_t mode);*
Открывает файл <pathname> и возвращает описатель файла.

■ **read, readv (POSIX) <unistd.h, sys/uio.h>**
*ssize_t read(int fd, void *buf, size_t count);*
*int readv(int fd, const struct iovec * vector, int count);*
Записывает <count> байтов файлового описателя <fd> в буфер <buf> или вектор <vector>.

■ **readlink (BSD) <unistd.h>**
*int readlink(const char *path, char *buf, size_t bufsiz);*
Помещает содержимое символьной ссылки <path> в буфер <buf> длиной <bufsiz>.

■ **remove (ANSI,POSIX) <stdio.h>**
*int remove(const char *pathname);*
Удаляет имя файла и, возможно, сам файл.

■ **rename (ANSI) <unistd.h>**
*int rename(const char *oldpath, const char *newpath);*
Изменяет имя или расположение файла <oldpath> на <newpath>.

■ **select (POSIX) <sys/time.h, sys/types.h, unistd.h>**
*int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);*
*int pselect(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, const struct timespec *timeout, sigset_t *sigmask);*
Ждут изменения статуса нескольких файловых описателей в течении <timeout>. n - на единицу больше самого большого номера описателей из всех наборов. Для манипуляции набором существуют 4 макроса:
FD_ZERO - очищающий набор;
FD_SET - Добавляет заданный описатель к набору;
FD_CLR - Удаляет описатель из из набора;
FD_ISSET - Проверяет, является ли описатель частью набора;

■ **sendfile (*) <sys/sendfile.h>**
*ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);*
Производит чтение данных из <in_fd> по смещению <offset> длиной <count> и запись их в <out_fd>.

■ **write, writev (POSIX) <unistd.h, sys/uio.h>**
*ssize_t write(int fd, const void *buf, size_t count);*
*int writev(int fd, const struct iovec * vector, int count);*
Записывает <count. байтов из буфера <buf> или вектора <vector> в файл <fd>.

■ **unlink (POSIX) <unistd.h>**
*int unlink(const char *pathname);*
Удаляет имя файла из файловой системы. Если это имя было последней ссылкой на файл и нет ни одного процесса, которые бы открыли этот файл, то файл удаляется.

Таблица 2.2: **Стандартные потоки ввода-вывода**

Имя	Назначение
stdin(0)	Стандартный вход;
stdout(1)	Стандартный выход (буфериз.);
stderr(2)	Стандартный выход ошибок (небуфериз.);

2.2.4 Функции работы с временными файлами

■ **mkstemp (BSD) <stdlib.h>**
*int mkstemp(char *template);*
Создает временный файл с уникальным именем определённым <template>. <template> должен иметь в конце символы "XXXXXX". Возвращает описатель fd созданного файла и заполняет XXXXXX, в template, сгенерированными символами. Нуждается в удалении!

■ **mktemp (POSIX) <stdlib.h>**
*char *mktemp(char *template);*
Создает временный файл с уникальным именем определённым <template>. <template> должен иметь в конце символы "XXXXXX". Возвращает имя созданного файла.

■ **tmpfile (POSIX) <stdio.h>**
*FILE *tmpfile (void);*
Создает уникальное имя временного файла с помощью префикса пути P_tmpdir, определенного в <stdio.h>. Файл автоматически удаляется при его закрытии или в случае завершения основной программы.

2.3 Функции поддержки переменного числа параметров <stdarg.h>

■ **va_arg**
type va_arg(va_list arg_ptr, type);
Значение параметра типа <type> выбирается из стека <arg_ptr>, по одному за каждое обращение к функции. Макрокоманда <va_arg> может использоваться любое число раз для отыскания параметров в списке.

■ **va_end**
void va_end(va_list arg_ptr);
Присваивает аргументу-указателю <arg_ptr> значение запрещающее его последующее использование (без повторной инициализации va_start).

■ **va_start**
void va_start(va_list arg_ptr, prev_param);
Назначает указателю <arg_ptr> адрес первого параметра в стеке входных параметров <prev_param>.

2.4 Функции/переменные работы со временем и временными интервалами

■ **alarm (SVr4, SVID, POSIX, X/OPEN, BSD 4.3) <unistd.h>**
unsigned int alarm(unsigned int seconds);
Настраивающая таймер на подачу сигнала ALARM.

■ **daylight (POSIX) <time.h>**
extern int daylight;
Переход на летнее время.

■ **timezone (POSIX) <time.h>**
long int timezone;
Содержит разницу, в секундах, между локальным временем и по Гринвичу.

■ **tzname (POSIX) <time.h>**
*extern char *tzname[2];*
Имя текущей временной зоны.

■ **asctime, ctime (POSIX) <time.h>**
*char *asctime(const struct tm *timeptr);*
*char *ctime(const time_t *timep);*
Преобразует время <timep>, <timeptr> в строку в формате "Wed Jun 30 21:49:08 1993".

■ **gmtime, localtime (POSIX) <time.h>**
*struct tm *gmtime(const time_t *timep);*
*struct tm *localtime(const time_t *timep);*
Преобразуют календарное время <timep> во время по Гринвичу и локальное;

■ **mktime (POSIX) <time.h>**
*time_t mktime(struct tm *timeptr);*
Преобразует структуру локального представления <timeptr> времени в структуру календарного представления.

■ **setitimer, getitimer (*NIX) <sys/time.h>**

*int setitimer(int which, const struct itimerval *value, struct itimerval *ovalue);*
*int getitimer(int which, struct itimerval *value);*
Получить и установить значение value интервального таймера which. Старое значение таймера сохраняется в ovalue. Типы таймеров which:

ITIMER_REAL Декр. в РВ и вызывает SIGALARM;

ITIMER_VIRTUAL Декр. только при выполнении процесса и вызывает SIGVTALRM;

ITIMER_PROF Декр. при выполнении процесса и при выполнении системы на защите процесса, вызывает SIGPROF.

———————————————-

■ **strftime (ANSI) <time.h>**
*size_t strftime(char *s, size_t max, const char *format, const struct tm *tm);*
Форматирует время <tm> в соответствии с указанным форматом <format> и помещает результат в символьный массив <s> размером <max>.

■ **time (POSIX) <time.h>**
*time_t time(time_t *t);*
Возвращает/устанавливает текущее время <t>

■ **times (POSIX) <sys/times.h>**
*clock_t times(struct tms *buf);*
Возвращает текущие состояние счетчика тиков, а также информации о времени выполнения процесса и его порожденных процессов в <buf>.

■ **sleep, usleep, nanosleep (POSIX) <unistd.h, time.h>**
unsigned int sleep(unsigned int seconds);
void usleep(unsigned long usec);
*int nanosleep(const struct timespec *req, struct timespec *rem);*
Функция задаёт интервал паузы: seconds - секунды, usec - микросекунды и req - наносекундах. В rem помещается реально прошедшее время.

2.5 Функции проверки и преобразования символов <ctype.h, cctype>

■ **isalnum (ANSI)**
int isalnum(int c);
Проверяет символ на принадлежность к текстовым символам.

■ **isalpha (ANSI)**
int isalpha(int c);
Проверяет символ на принадлежность к алфавитным символам (в стандартном окружении "C").

■ **isascii (ANSI)**
int isascii(int c);
Проверяет, является ли <c> 7-битным unsigned char, значение которого попадает в таблицу символов ASCII.

■ **iscntrl (ANSI)**
int iscntrl(int c);
Проверяет, является ли символ управляющим.

■ **isdigit (ANSI)**
int isdigit(int c);
Проверяет, является ли символ цифрой.

■ **isgraph (ANSI)**
int isgraph(int c);
Проверяет, является ли символ печатаемым (не пробел).

■ **islower (ANSI)**
int islower(int c);
Проверяет, является ли символ символом нижнего регистра.

■ **isprint (ANSI)**
int isprint(int c);
Проверяет, является ли символ печатаемым (включая пробел).

■ **ispunct (ANSI)**
int ispunct(int c);
Проверяет, является ли символ печатаемым (не должен быть пробелом или текстовым символом).

■ **isspace (ANSI)**
int isspace(int c);
Проверяет, являются ли символы неотображаемыми.

■ **isupper (ANSI)**
int isupper(int c);
Проверяет, расположен ли символ в верхнем регистре.

■ **isxdigit (ANSI)**
int isxdigit(int c);
Проверяет, принадлежит ли символ к шестнадцатеричному числу.

■ **toascii (ANSI)**
int toascii(int c);
Преобразует <c> в 7-битное значение unsigned char, т.е. превращает его в ASCII-символ посредством "сбрасывания"старшего бита.

■ **toupper, tolower (ANSI)**
int toupper(int c);
int tolower(int c);
Преобразует символ <c> к верхнему или нижнему регистру.

2.6 Функции работы со строками

■ **atof (POSIX) <stdlib.h, cstdlib>**
*double atof(const char *nptr);*
Преобразует строку <nptr> в вещественное число типа double.

■ **atoi, atol, atoll, atoq (POSIX) <stdlib.h, cstdlib>**
*int atoi(const char *nptr);*
*long atol(const char *nptr);*
*long long atoll(const char *nptr);*
*long long atoq(const char *nptr);*
Преобразуют строку в целое число.

■ **bzero (BSD) <string.h>**
*void bzero(void * s ,size_t n);*
Заполняет нулями байты строки.

■ **ecvt, fcvt, gcvt <stdlib.h, cstdlib>**
*char *ecvt(double number, int ndigits, int *decpt, int *sign);*
*char *fcvt(double number, int ndigits, int *decpt, int *sign);*
*char *gcvt(double number, size_t ndigit, char *buf);*
преобразует число с плавающей точкой в строку.

■ **strcat, strncat (POSIX) <string.h>**
*char *strcat(char *dest, const char *src);*
*char *strncat(char *dest, const char *src, size_t n);*
Добавляет строку <str> (n символов для strncat) к строке <dest>, перезаписывая символ окончания в конце <dest> и добавляя к строке символ окончания.

■ **strchr,strrchr (POSIX) <string.h>**
*char *strchr(const char *s, int c);*
*char *strrchr(const char *s, int c);*
Определение местонахождения символа в строке.

■ **strcmp, strncmp (POSIX) <string.h>**
*int strcmp(const char *s1, const char *s2);*
*int strncmp(const char *s1, const char *s2, size_t n);*
сравнивает две (n символов для strncmp) строки: s1 и s2. Возвращая целое число, которое меньше, больше нуля или равно ему, если s1 соответственно меньше, больше или равно s2.

■ **strcpy, strncpy (POSIX) <string.h>**
*char *strcpy(char *dest, const char *src);*
*char *strncpy(char *dest, const char *src, size_t n);*
Копирует строку (n символов для strncpy), на которую указывает <src> (включая завершающий символ окончания), в массив, на который указывает <dest>.

■ **strdup, strndup, strdupa, strndupa (BSD) <string.h>**
*char *strdup(const char *s);*
*char *strndup(const char *s, size_t n);*
*char *strdupa(const char *s);*
*char *strndupa(const char *s, size_t n);*
Дублируют строку;

■ **strerror (POSIX) <string.h>**
*char *strerror(int errnum);*
Возвращает строку с описанием кода ошибки, переданного в аргументе errnum.

■ **strlen (POSIX) <string.h>**
*size_t strlen(const char *s);*
Вычисляет длину строки <s>. Завершающий символ окончания не учитывается.

■ **strpbrk (POSIX) <string.h>**
*char *strpbrk(const char *s, const char *accept);*
Ищет первое совпадение в строке <s> с любым символом из строки <accept>.

■ **strspn, strcspn (POSIX) <string.h>**
*size_t strspn(const char *s, const char *accept);*
*size_t strcspn(const char *s, const char *reject);*
поиск набора символов в строке.

■ **strstr <string.h>**
*char *strstr(const char *haystack, const char *needle);*
Находит первую встретившуюся подстроку <needle> в строке <haystack>.

■ **strtok, strtok_r (POSIX) <string.h>**
*char *strtok(char *s, const char *delim);*
*char *strtok_r(char *s, const char *delim, char **ptrptr);*
Разбивает строку <s> на "лексемы"логические куски, такие, как слова в строке текста - разделённые символами, содержащимися в <delim>. Последующие вызовы производить с <s> = NULL.

■ **strtod, strtof, strtold (ANSI) <stdlib.h, cstdlib>**
*double strtod(const char *nptr, char **endptr);*
*float strtof(const char *nptr, char **endptr);*
*long double strtold(const char *nptr, char **endptr);*
Конвертируют строки ASCII в число с плавающей запятой.

■ **strtoul, strtoull (POSIX) <stdlib.h, cstdlib>**
*unsigned long int strtoul(const char *nptr, char **endptr, int base);*
*unsigned long long int strtoull(const char *nptr, char **endptr, int base);*
Конвертирует строку в беззнаковое целое число.

■ **strxfrm (BSD) <string.h>**
*size_t strxfrm(char *dest, const char *src, size_t n);*
Преобразует строку <src> в такую форму, что выполнение strcmp() над двумя такими строками, преобразованными посредством strxfrm(), будет таким же, как и выполнение strcoll над исходными строками.

2.7 Потоковая обработка строк (string) <string>

Для работы со строками в языке C++ предусмотрен шаблонный класс **basic_string** из которого определяется класс **string**:

```
typedef basic_string <char> string;
typedef basic_string <wchar_t> string; \\для UNICODE
```

Примеры инициализации строк:

```
string s1("Hello");  //Строка "Hello"
string s2(8,x);      //Восемь символов 'x'
string month = "March"; //Строка "March"
s = 'n'              //символ 'n'
cin >> s;            //вводить в <s>
getline(cin,s);      //--//--
str1.assign(str2);
str1[2] = 'r';       //Присвоить 'r'
string s1(s2+"test");
```

Для потоковой обработки строк используются классы: <ostringstream>

```
outstring << S1 << S2 << S3 << 2 << 3.58;
cout << outstring.str();
instring >> S1 >> S2 >> i;
```

2.7.1 Функции

■ **operator==, !=, >, <, >=, <=**
Операторы сравнения строк.

■ **assign**
basic_string& assign (const basic_string& str, size_type pos = 0, size_type n = npos);
basic_string& assign (const charT s, size_type n);*
basic_string& assign (const charT s);*
basic_string& assign (size_type n, charT c);
Создаёт новую строку из участка старой.

■ **append**
basic_string& append(const basic_string& str, size_type pos = 0, size_type n = npos);
basic_string& append (const charT s, size_type n);*
basic_string& append (const charT s);*
basic_string& append (size_type n, charT c);
basic_string& append(InputIterator first, InputIterator last);
Добавление к строке.

■ **at**
reference at(size_type pos);
const_reference at(size_type pos) const;
Получить символ по с индексом <POS>.

■ **begin, end**
iterator begin ();
iterator end ();
Возвращает итератор;

■ **capacity**
size_type capacity() const;
Максимальный размер строки без увеличения памяти под строку.

■ **compare**
int compare (const basic_string& str, size_type pos = 0, size_type n = npos) const;
int compare (const charT s, size_type pos, size_type n) const;*
int compare (const charT s, size_type pos = 0) const;*
Сравнение строк.

■ **copy**
size_type copy (charT s, size_type n, size_type pos = 0) const;*
Копировать строку в адрес определённый указателем.

■ **c_str**
const charT c_str() const;*
Возвращает указатель на строку с нулём в конце.

■ **data**
charT data()*
Возвращает указатель на строку без нуля в конце.

■ **empty**
bool empty() const;
Проверка строки на пустоту.

■ **erase**
basic_string& erase(size_type pos = 0, size_type n = npos);
iterator erase(iterator p);
iterator erase(iterator f, iterator l);
Удаление всех символов начиная от указанной позиции.

■ **begin, end**
iterator begin();
iterator end();
Получение начального/конечного итератора.

■ **find, rfind**
size_type find(const basic_string& str, size_type pos = 0) const;
size_type rfind(const basic_string& str, size_type pos = npos) const;
Поиск сначала/конца строки.

■ **find_first_of, find_last_of**
size_type find_first_of(const basic_string& str, size_type pos = 0) const;
size_type find_last_of (const basic_string& str, size_type pos = npos) const;
Поиск сначала/конца строки одного из перечисленных символов.

■ **find_first_not_of, find_last_not_of**
size_type find_first_not_of(const basic_string& str, size_type pos = 0) const;

size_type find_last_not_of (const basic_string& str, size_type pos = npos) const;
Поиск сначала/конца первого символа отсутствующего в списке.

■ **insert**
basic_string& insert(size_type pos1, const basic_string& str, size_type pos2 = 0, size_type n = npos);
Вставка в строку части другой строки.

■ **length, size**
size_type length() const;
size_type size() const;
Длина строки.

■ **max_size**
size_type max_size() const;
Максимальная длина строки.

■ **replace**
basic_string& replace(size_type pos1, size_type n1, const basic_string& str, size_type pos2 = 0, size_type n2 = npos);
Заменить участок строки.

■ **resize**
void resize (size_type n, charT c);
void resize (size_type n);
Изменить длину строки.

■ **substr**
basic_string substr(size_type pos = 0, size_type n = npos) const;
Получение части строки;

■ **swap**
void swap (basic_string &s)
Перестановка строк.

2.8 Стандартная библиотека шаблонов (STL)

STL представляет собой большую библиотеку шаблонов включающую в себя три ключевых компонента: контейнеры, итераторы, алгоритмы. Кроме того STL является расширяемой библиотекой. STL избегает операторов new и delete и использует распределители памяти(allocators) для выделения и высвобождения памяти. Существует возможность создавать пользовательские распределители памяти. Типы исключений в STL:

out_of_range - Индекс находится вне диапазона;
invalid_argument - Передаче недопустимого аргумента функции;
length_error - Создание слишком длинного контейнера;
bad_alloc - Неудачная попытка выделения области памяти;

2.8.1 Контейнеры

Контейнеры делятся на три основных категории: контейнеры последовательностей, ассоциативные контейнеры и адаптеры контейнеров. Контейнеры последовательностей (sequence containers) и Ассоциативные контейнеры имеют общее название - контейнеры первого класса (first-class containers). Существует ещё четыре типа контейнеров, которые считаются "почти контейнерами"("near-containers") - C-подобные массивы, string, bitset и valarray.

Контейнерные заголовочные файлы стандартной библиотеки:

- <vector>
- <list>
- <deque>
- <queue> - Содержит как queue, так и priority_queue.
- <stack>
- <map> - Содержит как map, так и multimap.
- <set> - Содержит как set, так и multiset.
- <bitset>

Общие имена типа **typedef** имеющиеся в контейнерах первого класса:

value_type - Тип элемента, хранимого в контейнере.

reference - Ссылка на тип элемента, хранимого в контейнере.

const_reference - Ссылка-константа на тип элемента, хранимого в контейнере.

pointer - Указатель на тип элемента, хранимого в контейнере.

iterator - Итератор, который указывает на тип элемента, хранимого в контейнере.

const_iterator - Констатный итератор, который указывает на тип элемента, хранимого в контейнере.

reverse_iterator - Обратный итератор, который указывает на тип элемента, хранимого в контейнере.

const_reverse_iterator - Константный обратный итератор, который указывает на тип элемента, хранимого в контейнере.

difference_type - Тип результата вычитания двух итераторов, которые ссылаются на один и тот же контейнер.

size_type - Тип, используемый для подсчёта элементов в контейнере и для индексации в контейнере последовательности.

Таблица 2.3: **Контейнерные классы STL**

Имя	Назначение	Тип итератора
Контейнеры последовательностей		
vector	Быстрые вставки и удаление в конец контейнера, прямой доступ к любому элементу.	произв. доступ
deque	Быстрые вставки и удаления в начало и конец контейнера, прямой доступ к любому элементу.	произв. доступ
List	Двухсвязный список, быстрая вставка и удаление элементов везде.	двунапр.
Ассоциативные контейнеры		
set	Быстрый поиск, дубликаты (одинаковые ключи)не допускаются.	двунапр.
multiset	Быстрый поиск, допускаются дубликаты.	двунапр.
map	Взаимно однозначное соответствие, дубликаты не допускаются, быстрый поиск значения по ключу.	двунапр.
multimap	Соответствие "один ко многим", дублирование ключей допускается, быстрый поиск значения по ключу.	двунапр.
Адаптеры контейнеров		
stack	"Последним пришел, первым вышел"(LIFO)	не поддерж.
queue	"Первым пришел, первым вышел"(FIFO)	не поддерж.
priority_queue	Элемент с наивысшим приоритетом всегда достигает выхода из очереди первым.	не поддерж.

Контейнеры последовательностей

Классы **vector** и **deque** реализованы на базе массивов. Класс **list** реализует связанный список. Дополнительные операции характерные для контейнеров последовательностей:

■ **front**
front();
Возвращает ссылку на первый элемент в контейнере.

■ **back**

Таблица 2.4: **Функции для всех STL-контейнеров**

Имя	Назначение
default constructor	Конструктор для обеспечения инициализации по умолчанию.
copy constructor	Конструктор, который инициализирует контейнер в качестве копии существующего контейнера.
destructor	Деструктор контейнера.
empty	Проверка контейнера на пустоту.
max_size	Возвращает максимальное число элементов для контейнера.
size	Возвращает число элементов в контейнере.
operator=	Присваивает один контейнер другому.
operator<, operator<=, operator>, operator>=, operator==, operator!=	Сравнивает два контейнера.
swap	Поменять местами элементы двух контейнеров.
Функции, только в контейнерах первого класса	
begin	Возвращает iterator, либо const_iterator который ссылается на первый элемент контейнера.
end	Возвращает iterator, либо const_iterator который ссылается на последний элемент контейнера.
rbegin	Возвращает reverse_iterator, либо const_reverse_iterator который ссылается на последний элемент контейнера.
rend	Возвращает reverse_iterator, либо const_reverse_iterator который ссылается на позицию перед первым элементом контейнера.
erase	Удаляет один или несколько элементов из контейнера.
clear	Удаляет все элементы из контейнера.

back();
Возвращает ссылку на последний элемент в контейнере.

■ **push_back**
push_back();
Вставляет новый элемент в конец контейнера.

■ **pop_back**
pop_back();
Выталкивает последний элемент контейнера.

Контейнер **vector** обеспечивает структуру данных непрерывной областью памяти. Это позволяет обеспечивает эффективный прямой доступ к любому элементу контейнера **vector** посредством операции индексирования []. Все алгоритмы STL могут работать с контейнером **vector** Итераторы для **vector** обычно реализуются как указатели на элементы контейнера **vector**.

```
std::vector<int> v;
v.push_back(2);
cout << "\nРазмер вектора v: " << v.size();

std::vector<int>::const_iterator p1;
for(p1 = v.begin(); p1 != v.end(); p1++) cout << *p1 << ' ';
std::vector<int>::reverse_iterator p2;
for(p2 = v.rbegin(); p2 != v.rend(); ++p2) cout << *p2 << ' ';

int a[ 6 ] = { 1, 2, 3, 4, 5, 6 };
std::vector<int> v1(a, a+6);
std::ostream_iterator<int> output(cout," ");
std::copy(v1.begin(), v1.end(), out); //печать вектора
try
{
  v1.at(100) = 777;   //доступ вне массива
}
catch(std::out_of_range e)
{ cout << "\nИсключение" << e.what(); }
v1.erase(v.begin());
```

Контейнер **list** предоставляет эффективную реализацию операции вставки и удаления в любую позицию контейнера. Класс **list** реализуется как двухсвязный список, то есть каждый узел в **list** содержит указатель на предыдущий и на следующий узел. Любой алгоритм, который требует итераторов для чтения и для записи, прямых и двунаправленных итераторов, может выполняться с list. Дополнительные функции класса **list**:

■ **inplace_merge**
void inplace_merge(_BidirectionalIter __first, _BidirectionalIter __middle, _BidirectionalIter __last, _Compare __comp);
Объединяет две возростающие последовательности в одном и том же контейнере.

■ **splice**
splice();
Вырезает элементы из одного контейнера и помещает их в другой.

■ **push_front**
push_front();
Вставить элемент в начало контейнера

■ **pop_front**
pop_front();
Вытолкнуть элемент с начала контейнера

■ **remove**
remove();
–//–

■ **merge**
_OutputIter merge(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result)
Объединяет две возростающие последовательности в одну.

■ **sort**
void sort(_RandomAccessIter __first, _RandomAccessIter __last);
Сортировка элементов в возрастающем порядке.

```
std::list<int> Values, otherValues;
Values.push_front(1);
Values.push_back(2);
values.sort();
```

Контейнер **deque** объединяет многие возможности классов **vector** и **list**. Реализуется на основе очереди с двумя концами и обеспечивает эффективный индексный доступ с эффективные операции вставки в начало и конец контейнера. Класс **deque** обеспечивает поддержку итераторов произвольного доступа. Наиболее часто **deque** используется в качестве очереди FIFO.

```
std::deque<double> values;
std::ostream_iterator<double> output(cout," ");
values.push_front(2.2);
values.push_back(1.1);
for(int i=0; i < values.size(); ++i)
  cout << values[i] << ' ';
values.pop_front();
values[1]=5.4;
```

Ассоциативные контейнеры

Ассоциативные контейнеры STL предназначены для обеспечения прямого доступа с целью сохранения и выборки элементов с помощью ключей (ключи поиска). Имеется четыре ассоциативных контейнера: multiset, set, multimap, map. В каждом контейнере ключи сохраняются упорядоченными. Ассоциативные контейнеры поддерживают дополнительные функции:

■ **equal_range**
pair<_ForwardIter, _ForwardIter> equal_range(_ForwardIter __first, _ForwardIter __last, const _Tp& __val);
Возвращает пару прямых итераторов, содержащих результаты **lower_bound** и **upper_bound**.

■ **find**
find();
Поиск ключа.

■ **lower_bound**
_ForwardIter __lower_bound(_ForwardIter __first, _ForwardIter __last, const _Tp& __val, _Distance)*
Определения позиции первого вхождения указанного ключа.

■ **upper_bound**
_ForwardIter upper_bound(_ForwardIter __first, _ForwardIter __last, const _Tp& __val)
Определения позиции за последним вхождением указанного ключа.

■ **count**
count();
Возвращает количество указанных ключей в контейнере.

Контейнеры **multiset** и **set** используются для быстрого сохранение и выборки ключей. Контейнер **multiset** допускает использование одинаковых ключей. Упорядочение элементов определяется компараторным объектом-функцией. Контейнеры поддерживает двунаправленные итераторы (но не итераторы произвольного доступа).

```
int a[10] = {7,22,9,1,18,30,100,22,85,13};
typedef std::multiset<int, std::<int> > ims;
ims intMultiset;
std::ostream_iterator<int> output(cout, " ");
intMultiset.insert(15);
result=intMultiset.find(20);
if ( result != intMultiset.end() ) cout << "Найдено значение";
intMultiset.insert(a, a+10);
std::copy(intMultiset.begin(),intMultiset.end(), output);
```

Контейнеры **multimap** и **map** используются для быстрого сохранения и нахождения ключей и ассоциированных значений (пара ключ/значение). При вставке в эти контейнеры используется объект **pair** Контейнер **multimap** позволяет ассоциировать несколько значений с одним ключем. Упорядочение элементов определяется компараторным объектом-функцией. Контейнеры поддерживает двунаправленные итераторы (но не итераторы произвольного доступа).

```
typedef std::multimap<int, double, std::less<int> > mmid;
mmid pairs;
pairs.insert(mmid::value_type(15, 2.7) );
for(mmid::const_iterator iter = pairs.begin();
    iter != pairs.end(); ++iter)
  cout << iter->first << '\t' << iter->second << '\n';
pairs[25]=45.65;
```

Адаптеры контейнеров

STL предоставляет три адаптера контейнера - **stack, queue, priority_queue**. Адаптеры не являются контейнерами первого класса, поскольку они не предоставляют реализации фактической структуры данных в которой могут сохраняться элементы, и поскольку адаптеры не поддерживают итераторы. Все три класса адаптеров предоставляют функции-члены **push** и **pop**, которые реализуют соответствующий метод вставки элемента.

Адаптер **stack** обеспечивает структуру LIFO и может быть реализован с любым из контейнеров последовательности. Адаптер **queue** обеспечивает структуру FIFO и может быть реализован с контейнерами **list** и **deque**. Адаптер **priority_queue** обеспечивает очередь в которой наиболее приоритетное значение всегда будет удаляться первым. **priority_queue** может быть реализован с контейнерами **vector** и **deque**. Сравнение элементов выполняется с помощью функции-объекта **less<T>**.

```
std::stack<int> intDequeStack;
std::stack<int, std::vector<int> > intVectorStack;
std::stack<int, std::list<int> > intListStack;

intDequeStack.push(23);
intVectorStack.push(23);
intListStack.push(34);
```

2.8.2 Итераторы

Итераторы схожи с указателями и используются для указания на элементы контейнеров первого класса. STL контейнеры первого класса предоставляют функции-члены **begin()** и **end()**, которые возвращают итератор указывающий соответственно на первый и последний элемент контейнера. Если итератор **i** указывает на определённый элемент, то **++i** указывает на "следующий"элемент, а ***i** ссылается на элемент, на который указывает **i**. Итератор, полученый из функции **end()**, может использоваться в сравнении на равенство и неравенство для определения окончания "движущегося итератора". Для ссылки на элемент контейнера используется объект типа **iterator** или **const_iterator**.

Категории итераторов:

input - Используется для чтения элементов из контейнера. Итератор для чтения может перемещаться только по направлению вперёд и поддерживает только однопроходные алгоритмы.

output - Используется для записи элемента в контейнер. Итератор для записи может перемещаться только по направлению вперёд и поддерживает только однопроходные алгоритмы.

forward - Объединяет возможности итераторов для чтения и для записи и сохраняет их позицию в контейнере.

bidirectional - Объединяет возможности однонаправленного итератора с возможностью перемещаться в обратном направлении.

random access - Объединяет возможности двунаправленного итератора с возможностью прямого доступа к любому элементу контейнера.

Категория итератора, поддерживаемая каждым контейнером, определяет, может ли этот контейнер использоваться со специфическими алгоритмами в STL.

2.8.3 Алгоритмы

Контейнеры инкапсулируют некоторые базовые операции, но STL-алгоритмы реализуются независимо от контейнеров. Алгоритмы оперируют элементами контейнеров только с помощью итераторов. Можно создавать собственные алгоритмы.

Алгоритмы, модифицирующие последовательности

■ **copy**
copy();
-//-

■ **copy_backward**
_BidirectionalIter2 __copy_backward(_BidirectionalIter1 __first, _BidirectionalIter1 __last, _BidirectionalIter2 __result,

Таблица 2.5: **Операции с итераторами для каждого типа итератора**

Имя	Назначение
Все итераторы	
++p	Преинкримент итератора
p++	Постинкримент итератора
Итераторы ввода	
*p	Разыменование итератора
p=p1	Присвоение итератора итератору
p==p1	Сравнение итераторов на равенство
p!=p1	Сравнение итераторов на неравенство
Итераторы вывода	
*p	Разыменование итератора
p=p1	Присвоение итератора итератору
Однонаправленные итераторы	
*	Обеспечивают все функциональные возможности итераторов ввода и вывода
Двунаправленные итераторы	
–p	Предекремент итератора
p–	Постдекремент итератора
Итераторы с произвольным доступом	
p+=i	Инкремент итератора p на i позиций
p-=i	Декремент итератора p на i позиций
p+i	Итератор помещается на позицию p+i
p-i	Итератор помещается на позицию p-i
p[i]	Возвращение ссылку на элемент, смещённый от p на i позиций
p<p1, p<=p1, p>p1, p>=p1	Сравнение итераторов

bidirectional_iterator_tag, _Distance)*
Обратное копирование части одного контейнера в другой контейнер.

■ **fill**
void fill(_ForwardIter __first, _ForwardIter __last, const _Tp& __value);
Заполняет все элементы контейнера значением <value>.

■ **fill_n**
_OutputIter fill_n(_OutputIter __first, _Size __n, const _Tp& __value);
Заполняет указанные элементы контейнера значением <value>.

■ **generate**
void generate(_ForwardIter __first, _ForwardIter __last, _Generator __gen)
Заполняет все элементы контейнера значением возвращаемым функцией <__gen>.

■ **generate_n**
generate_n();
Заполняет указанные элементы контейнера значением возвращаемым функцией <__gen>.

■ **iter_swap**
void iter_swap(_ForwardIter1 __a, _ForwardIter2 __b);
Перестановка значений контейнера (по ссылке).

■ **includes**
bool includes(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2);
Проверяет находятся ли элементы второго множества в первом.

■ **next_permutation**
bool next_permutation(_BidirectionalIter __first, _BidirectionalIter __last);
Следующая перестановка в лексикографическом порядке.

■ **prev_permutation**
bool prev_permutation(_BidirectionalIter __first, _BidirectionalIter __last);
Предыдущая перестановка в лексикографическом порядке.

■ **partition**
_ForwardIter partition(_ForwardIter __first, _ForwardIter __last, _Predicate __pred)
Разделение диапазонов элементов.

■ **remove**
_ForwardIter remove(_ForwardIter __first, _ForwardIter __last, const _Tp& __value);
Удаление из указанного участка контейнера всех указанных объектов <__value>

■ **remove_copy**
_OutputIter remove_copy(_InputIter __first, _InputIter __last, _OutputIter __result, const _Tp& __value)
Перенос из указанного участка контейнера в другой контейнер всех указанных объектов <__value>.

■ **remove_copy_if**
remove_copy_if();
Перенос из указанного участка контейнера в другой контейнер объектов выбранных функцией сравнения <__pred>.

■ **remove_if**
_ForwardIter remove_if(_ForwardIter __first, _ForwardIter __last, _Predicate __pred)
Удаление из указанного участка контейнера объектов выбранных функцией сравнения <__pred>

■ **replace**
void replace(_ForwardIter __first, _ForwardIter __last, const _Tp& __old_value, const _Tp& __new_value)
Производит замену объекта <__old_value> на <__new_value> по указанному участку контейнера.

■ **replace_copy**
_OutputIter replace_copy(_InputIter __first, _InputIter __last, _OutputIter __result, const _Tp& __old_value, const _Tp& __new_value)
Производит замену объекта <__old_value> на <__new_value> по указанному участку контейнера и помещений старых значений

в контейнер <__result>.

■ **replace_copy_if**
_OutputIter replace_copy_if(Iterator __first, Iterator __last, _OutputIter __result, _Predicate __pred, const _Tp& __new_value)
Производит замену объекта выбраного функцией <__pred> на <__new_value> по указанному участку контейнера и помещений старых значений в контейнер <__result>.

■ **replace_if**
void replace_if(_ForwardIter __first, _ForwardIter __last, _Predicate __pred, const _Tp& __new_value)
Производит замену объекта выбраного функцией <__pred> на <__new_value> по указанному участку контейнера.

■ **reverse**
void __reverse(_BidirectionalIter __first, _BidirectionalIter __last, bidirectional_iterator_tag)
Инвертирование последовательности указанных элементов контейнера./

■ **reverse_copy**
_OutputIter reverse_copy(_BidirectionalIter __first, _BidirectionalIter __last, _OutputIter __result)
Копирует указанные элементы в обратном порядке.

■ **rotate**
_ForwardIter rotate(_ForwardIter __first, _ForwardIter __middle, _ForwardIter __last)
Ротация элементов.

■ **rotate_copy**
_OutputIter rotate_copy(_ForwardIter __first, _ForwardIter __middle, _ForwardIter __last, _OutputIter __result)
Ротация элементов с копированием.

■ **set_difference**
_OutputIter set_difference(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result);
Определение элементов из первого множества отсутствующих во втором.

■ **set_intersection**
_OutputIter set_intersection(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result);
Определение элементов из первого множества присутствующих во втором.

■ **set_symmetric_difference**
_OutputIter set_symmetric_difference(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result);
Определение элементов из первого множества отсутствующих во втором и элементов второго отсутствующих в первом.

■ **set_union**
OutputIter set_union(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result);
Создаёт множество отсортированных элементов их двух контейнеров в третьем.

■ **stable_partition**
_ForwardIter stable_partition(_ForwardIter __first, _ForwardIter __last, _Predicate __pred);
Подобна **partition**.

■ **swap**
void swap(_Tp& __a, _Tp& __b);
Перестановка значений контейнера.

■ **swap_ranges**
_ForwardIter2 swap_ranges(_ForwardIter1 __first1, _ForwardIter1 __last1, _ForwardIter2 __first2)
Перестановка группы элементов контейнера.

■ **transform**
transform();
-//-

■ **unique**
_ForwardIter unique(_ForwardIter __first, _ForwardIter __last);
Удаляет из контейнера одинаковые элементы

■ **unique_copy**
_OutputIter __unique_copy(_InputIter __first, _InputIter __last, _OutputIter __result, _Tp)*
Копирует все уникальные элементы в другой контейнер.

Алгоритмы, не модифицирующие последовательности

■ **adjacent_find**
_ForwardIter adjacent_find(_ForwardIter __first, _ForwardIter __last, _BinaryPredicate __binary_pred)
Возвращает итератор для чтения, указывающий на первый из двух идентичных смежных элементов в последовательности.

■ **equal**
inline bool equal(_InputIter1 __first1, _InputIter1 __last1,_InputIter2 __first2)
Сравнение указанных участков двух контейнеров.

■ **find**
_InputIter find(_InputIter __first, _InputIter __last, const _Tp& __val, input_iterator_tag)
Возвращает положение искомого значения <__val>

■ **find_each**
find_each();
-//-

■ **find_end**
find_end();
-//-

■ **find_first_of**
find_first_of();
-//-

■ **find_if**
InputIter find_if(_InputIter __first, _InputIter __last, _Predicate __pred, input_iterator_tag)
Возвращает положение значения определённого функцией <__pred>.

■ **lexicographical_compare**
bool lexicographical_compare(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2);
Используется для лексикографического сравнения двух массивов символов.

■ **max**
const _Tp& max(const _Tp& __a, const _Tp& __b);
Определение максимального значения.

■ **min**
const _Tp& min(const _Tp& __a, const _Tp& __b);
Определения минимального значения.

■ **mismatch**
pair<_InputIter1, _InputIter2> mismatch(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2);
Выполняет сравнение указанных участков двух контейнеров и возвращает пару итераторов указывающих различные позиции контейнеров.
std::pair<std::vector<int>::iterator,
std::vector<int>::iterator> location;
location = std::mismatch(v1.begin(),v1.end(),v3.begin());

■ **search**
search();
-//-

■ **search_n**
search_n();
-//-

Числовые алгоритмы <numeric>

■ **accumulate**
_Tp accumulate(_InputIterator __first, _InputIterator __last, _Tp

__init);
Суммирование значений указанной области контейнера.

■ **adjacent_difference**
_OutputIterator adjacent_difference(_InputIterator __first, _InputIterator __last, _OutputIterator __result)
Вычисление разницы между парой смежных элементов

■ **count**
void count(_InputIter __first, _InputIter __last, const _Tp& __value,_Size& __n)
Выполняет подсчет количества объектов <__value> в указанной области контейнера.

■ **count_if**
void count_if(_InputIter __first, _InputIter __last, _Predicate __pred, _Size& __n)
Выполняет подсчет количества объектов выбранных функцией <__pred> в указанной области контейнера.

■ **for_each**
_Function for_each(_InputIter __first, _InputIter __last, _Function __f);
Применение функции <__f> к указанным элементам контейнера. Общая функция должна принимать текущий элемент в качестве аргумента и не должна модифицировать этот элемент.

■ **inner_product**
_Tp inner_product(_InputIterator1 __first1, _InputIterator1 __last1, _InputIterator2 __first2, _Tp __init)
Вычисление суммы произведений двух последовательностей.

■ **max_element**
_ForwardIter max_element(_ForwardIter __first, _ForwardIter __last);
Возвращает указатель на максимальный элемент в контейнере.

■ **min_element**
_ForwardIter min_element(_ForwardIter __first, _ForwardIter __last);
Возвращает указатель на минимальный элемент в контейнере.

■ **nth_element**
void nth_element(_RandomAccessIter __first, _RandomAccessIter __nth, _RandomAccessIter __last);
Разделение диапазонов элементов.

■ **partial_sum**
_OutputIterator partial_sum(_InputIterator __first, _InputIterator __last, _OutputIterator __result)
Вычисление суммы элементов двух контейнеров с накоплением.

■ **partial_sort**
void partial_sort(_RandomAccessIter __first, _RandomAccessIter __middle, _RandomAccessIter __last);
Сортировка части последовательности.

■ **partial_sort_copy**
_RandomAccessIter partial_sort_copy(_InputIter __first, _InputIter __last, _RandomAccessIter __result_first, _RandomAccessIter __result_last);
Сортировка части последовательности с копированием результата.

■ **random_shuffle**
inline void random_shuffle(_RandomAccessIter __first, _RandomAccessIter __last)
Располагает элементы указанного участка контейнера в произвольном порядке.

■ **stable_sort**
void stable_sort(_RandomAccessIter __first, _RandomAccessIter __last);
Сортировка (аналогично **sort**)

■ **transform**
_OutputIter transform(_InputIter __first, _InputIter __last, _OutputIter __result, _UnaryOperation __oper);
Применение функции <__f> к указанным элементам контейнера. Общая функция должна принимать текущий элемент в качестве аргумента, не должна модифицировать этот элемент и должна возвращать трансформированное значение.

Алгоритмы, сортировки кучи

■ **make_heap**
make_heap(_RandomAccessIterator __first, _RandomAccessIterator __last);
Создание и инициализация кучи.

■ **pop_heap**
void pop_heap(_RandomAccessIterator __first, _RandomAccessIterator __last);
Удаление элемента с вершины кучи.

■ **push_heap**
void push_heap(_RandomAccessIterator __first, _RandomAccessIterator __last);
Добавление нового элемента в кучу.

■ **sort_heap**
sort_heap(_RandomAccessIterator __first, _RandomAccessIterator __last, _Compare __comp);
Сортировка последовательности значений.

2.8.4 Класс <bitset>

Класс **bitset** обеспечивает операции для создания и манипуляции наборами битов. Наборы битов имеют фиксированный размер: `bitset< size > b;`. Операции:

■ **set**
Установка указанного бита.

■ **reset**
Сброс указанного бита.

■ **flip**
Переключает бит.

■ **at**
Получить бит.

■ **test**
Проверка бита.

■ **size**
Возвращает число битов в наборе.

■ **count**
Возвращает число установленных битов.

■ **any**
Возвращает **true** если хоть один бит в наборе установлен.

■ **none**
Возвращает **true** если не один бит в наборе неустановлен.

■ **==, !=**
Сравнение наборов битов.

■ **&=, |=, ^ =, »=, «=**
Битовые операции над наборами битов.

■ **to_string**
Преобразует набор битов в строку.

■ **to_ulong**
Преобразует набор битов в **unsigned long**.

2.8.5 Объекты-функции

Объекты-функции и адаптеры-функций предназначены для того чтобы сделать STL более гибкой. Объект-функция содержит функцию, которая может быть интерпретирована с синтаксической и семантической точки зрения как функция, использующая operator(). Объекты-функции STL:

divides<T> - арифметический;

equal_to<T> - реляционный;

greater<T> - реляционный;

greater_equal<T> - реляционный;

less<T> - реляционный;

less_equal<T> - реляционный;

logical_and<T> - логический;
logical_not<T> - логический;
logical_or<T> - логический;
minus<T> - арифметический;
moduls<T> - арифметический;
negate<T> - арифметический;
not_equal_to<T> - реляционный;
plus<T> - арифметический;
multiplies<T> - арифметический;

2.9 функции работы с памятью

■ **calloc, malloc, free, realloc (ANSI) <stdlib.c, cstdlib>**
*void *calloc(size_t nmemb, size_t size);*
*void *malloc(size_t size);*
*void free(void *ptr);*
*void *realloc(void *ptr, size_t size);*
Распределяют и освобождает динамическую память.
calloc - выделяет блок памяти с очисткой;
malloc - выделяет блок памяти;
free - освобождение блока памяти;
realloc - меняет размер блока памяти;

■ **delete (C++)**
delete terget;
Освобождает динамическую память выделенную с помощью оператора new. Кроме того оператор delete активизирует диструктор объекта:
- **target *r= new char[256]; delete r;**
- **target *r= new char[256]; delete []r;**

■ **memchr, memrchr (POSIX) <string.h>**
*void *memchr(const void *s, int c, size_t n);*
*void *memrchr(const void *s, int c, size_t n);*
Ищет символ <c> в первых/последних <n> байтах той области памяти, на которую указывает <s>. Совпадение первого байта с <c> (представленным как беззнаковый символ) останавливает выполнение операции.

■ **memcmp (BSD) <string.h>**
*int memcmp(const void *s1, const void *s2, size_t n);*
Сравнивает первые <n> байтов областей памяти <s1> и <s2>. Возвращается целое число, меньшее, большее нуля или равное ему, если определено, что <s1> меньше, больше, или равно <s2>.

■ **memcpy (BSD) <string.h>**
*void *memcpy(void *dest, const void *src, size_t n);*
Копирует <n> байтов из области памяти <src> в область памяти <dest>.

■ **memmove (BSD) <string.h>**
*void *memmove(void *dest, const void *src, size_t n);*
Копирует <n> байтов из области памяти <src> в область памяти <dest>. Области памяти могут перекрываться.

■ **memset (BSD) <string.h>**
*void *memset(void *s, int c, size_t n);*
Заполняет первые <n> байтов области памяти <s> байтом <c>.

■ **mlock, munlock (POSIX) <sys/mman.h>**
*int mlock(const void *addr, size_t len);*
*int munlock(const void *addr, size_t len);*
Запрещает/разрешает страничный обмен памяти в области, начинающейся с адреса <addr> длиной <len> байтов.

■ **mlockall, munlockall (POSIX) <sys/mman.h>**
int mlockall(int flags);
int munlockall(void);
Запрещает/разрешает страничный обмен для всех страниц в области памяти вызывающего процесса.

■ **mmap, munmap (POSIX) <unistd.h, sys/mman.h>**
*void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset);*
*int munmap(void *start, size_t length);*
mmap - возвращает адрес отражённых <length> байтов, начиная со смещения <offset> файла (или другого объекта), определенного файловым описателем <fd>, в память, начиная с адреса <start>.
munmap - удаляет все отражения из заданной области памяти.

■ **mprotect (POSIX) <sys/mman.h>**
*int mprotect(const void *addr, size_t len, int prot);*
Контролирует доступ <prot> к области памяти <addr> <len>.

■ **msync (POSIX) <unistd.h, sys/mman.h>**
*int msync(const void *start, size_t length, int flags);*
Записывает на диск изменения, внесенные в файл, отраженный в память при помощи функции mmap.

■ **new (C++)**
*Type *array = new Type[256];*
Используется для динамического выделения памяти (для размещения объекта в свободной памяти). Указателю array присваивается адрес выделенной памяти или присваивается NULL при её отсутствии. Оператор new автоматически активизирует конструктор объекта. Если определена функция _new_handler то она вызывается при отсутствии памяти. Способы применения оператора new:
- **float *r = new float; delete r;**
- **float *r = new float(3.14); delete r;**
- **float *r = new float[10]; delete []r;**

■ **set_new_handler (C++) <new.h | new>**
*void set_new_handler(void (*VFP)());*
Присваивает адресу глобальной переменной _new_handler адрес функции <VFP> обработки ошибок оператора new.

2.10 Специальные функции

■ **asm (*NIX)**
asm(asm_kod,output,input,modifi);
Позволяет выполнять встроенные инструкции языка asm <asm_kod> входными параметрами <input>, выходными <output>, и модифицируемыми <modifi>.

■ **assert (ANSI) <assert.h|cassert>**
void assert (int expression);
Выводит на стандартный вывод сообщение об ошибке и прекращает работу программы, если утверждение expression ложно (т.е., равно нулю). Происходит только в случае, если не определена переменная NDEBUG.

■ **chroot (BSD)**
*int chroot(const char *path);*
Функция установки нового корневого каталога.

■ **main (ANSI)**
*main(int argc, char *argv[], char *envp[]);*
Основная функция. Переопределяется для выполнения в ней пользовательской программы. Переменной argc присваивается общее число параметров разделенных пробелами. Переменной argv присваиваются отдельные параметры командной строки. Переменной envp присваиваются переменные среды вызываемой программы.

■ **mtrace (*) <mcheck.h>**
void mtrace(void);
Запуск трассировки - используемой программой памяти. Лог помещается в файл на который указывает переменная окружения MALLOC_TRACE. Для обработки лога вызывается команда: "mtrace my_prog $MALLOC_TRACE.

■ **optarg (POSIX) <unistd.h>**
*extern char *optarg;*
Содержит текстовый аргумент параметра.

■ **optind, opterr, optopt (POSIX) <unistd.h>**
extern int optind, opterr, optopt;
optind - индекс аргумента;
opterr - ошибка опции;
optopt - необработанная опция;

■ **getopt (POSIX) <unistd.h, getopt.h >**
*int getopt(int argc, char * const argv[], const char *optstring);*
Обрабатывает параметры <argc> <argv> команды на предмет поиска коротких опций <optstring>

■ **getopt_long (POSIX) <unistd.h, getopt.h>**
*int getopt_long(int argc, char * const argv[], const char *optstring, const struct option *longopts, int *longindex);*

Обрабатывает параметры <argc> <argv> команды на предмет поиска коротких <optstring> и длинных <longopts> опций.

■ **getenv (POSIX) <stdlib.h>**
*char *getenv(const char *name);*
Получает значения переменной окружения <name>;

■ **getpagesize (BSD) <unistd.h>**
size_t getpagesize(void);
Возвращает количество байтов в странице.

■ **pathconf, fpathconf (POSIX) <unistd.h>**
*long pathconf(char *path, int name);*
long fpathconf(int filedes, int name);
Возвращает ограничение параметра <name> для файловой системы на которой находится файл <path>,<filedes>

■ **putenv (POSIX) <stdlib.h>**
*int putenv(char *string);*
Добавляет или изменяет переменную окружения.

■ **setenv, unsetenv (BSD) <stdlib.h>**
*int setenv(const char *имя, const char *значение, int overwrite);*
*void unsetenv(const char *name);*
Изменение, добавление или удаление переменной окружения <имя> на <значение>.

■ **sysinfo (Linux) <sys/sysinfo.h>**
*int sysinfo(struct sysinfo *info);*
Возвращает общесистемную статистику.

■ **uname (POSIX) <sys/utsname.h>**
*int uname(struct utsname *buf);*
Возвращает информацию о системе в структуру с адресом <buf>.

2.10.1 Работа с терминалом

■ **isatty (SVID, AT&T, X/OPEN, BSD 4.3) <unistd.h>**
int isatty(int desc);
Определяет, ссылается ли данный описатель на терминал.

■ **tcgetattr, tcsetattr <termios.h, unistd.h>**
*int tcgetattr(int fd, struct termios *termios_p);*
*int tcsetattr(int fd, int optional_actions, struct termios *termios_p);*
Получить/установить атрибуты терминала.

■ **ttyname (POSIX.1) <unistd.h>**
*char *ttyname(int desc);*
Возвращает название терминала.

2.10.2 Работа с динамическими библиотеками

■ **dlopen (*NIX) <dlfcn.h>**
*void *dlopen (const char *filename, int flag);*
Открывает и возвращает адрес динамической библиотеки <filename> с флагами <flag>

■ **dlerror (*NIX) <dlfcn.h>**
*char *dlerror()*
Возвращает текстовую строку ошибки возникшей при работе с динамической библиотекой.

■ **dlsym (*NIX) <dlfcn.h>**
*void *dlsym(void *handle, char *symbol);*
Получить адрес функции с именем <symbol> которая ищется в библиотеке <handle>

mycomdlclose (*NIX) <dlfcn.h>
int dlclose(void *handle);
Закрытие, ранее открытой динамической библиотеки
■ **_init() (*NIX)**
Выполняется при открытии динамической библиотеки

■ **_fini() (*NIX)**
Выполняется при закрытии динамической библиотеки

2.10.3 Лимитирование

■ **getrlimit (*NIX) <sys/resource.h>**
Получение различных лимитов для пользователя.

■ **setrlimit (*NIX) <sys/resource.h>**
Устанавливает различные лимиты для пользователя.

■ **getrusage (*NIX) <sys/resource.h>**
Получение различных лимитов и статистики использования ресурсов для пользователя.

■ **ulimit (SVID) <ulimit.h>**
long ulimit(int cmd, long newlimit);
Установка или определение ограничений пользователя.

2.10.4 Документирование и ведение логов

■ **openlog (BSD) <syslog.h>**
*void openlog(char *ident, int option, int facility);*
Связывает с программой <facility>, ведущей системный журнал с опциями <option>. <indent> указывает на строку идентифицирующую программу генерирующую логи.

■ **closelog (BSD) <syslog.h>**
void closelog(void);
Закрывает описатель, используемый для записи данных в журнал.

■ **syslog (BSD) <syslog.h>**
*void syslog(int priority, char *format, ...);*
Создает сообщение для журнала из <format>, с приоритетом <priority>.

2.10.5 Функции управления безопасностью

■ **chmod, fchmod (POSIX) <sys/types.h, sys/stat.h>**
*int chmod(const char *path, mode_t mode);*
int fchmod(int fildes, mode_t mode);
Изменяют режим доступа к файлу, заданному параметром <path> или описателем файла <fildes>.

■ **getuid, geteuid (POSIX) <unistd.h, sys/types.h>**
uid_t getuid(void);
uid_t geteuid(void);
Возвращает идентификатор действительного/эффективного пользователя текущего процесса.

■ **getgid, getegid (POSIX) <unistd.h, sys/types.h>**
gid_t getgid(void);
gid_t getegid(void);
Возвращает идентификатор действительной/эффективной группы текущего процесса.

■ **stat, fstat, lstat (POSIX) <sys/types.h, sys/stat.h, unistd.h>**
*int stat(const char *file_name, struct stat *buf);*
*int fstat(int filedes, struct stat *buf);*
*int lstat(const char *file_name, struct stat *buf);*
stat возвращает информацию о файле <file_name> и заполняет буфер <buf>;
lstat дополнительно информацию о ссылке;
fstat информацию о <filedes>.

■ **setreuid, setregid (BSD) <sys/types.h, unistd.h>**
int setreuid(uid_t ruid, uid_t euid);
int setregid(gid_t rgid, gid_t egid);
Устанавливает действительный и действующий идентификатор пользователя/группы текущего процесса

■ **setgid (SVID) <sys/types.h, unistd.h>**
int setgid(gid_t gid);
Устанавливает идентификатор эффективной группы текущего процесса.

■ **setuid (POSIX) <sys/types.h, unistd.h>**
int setuid(uid_t uid);
Устанавливает фактический идентификатор владельца текущего

процесса.

2.11 Потоковые функции языка C++

Потоковые функции языка C++ представляют собой подборку классов предоставляющих функции для работы с основными потоковыми устройствами системы. Для подключения потоковых функций (классов) необходимо включать следующие заголовочные файлы:

<iostream> Включает описание классов: cin, cout, cerr, clog;

<iomanip> Включает информацию для обработки форматированного ввода-вывода;

<fstream> Включает информацию для выполнения операций с файлами;

<strstream> Включает информацию для выполнения операций со строкой;

Потоковые классы имеют следующую иерархию:
istream ⇒ ios;
ostream ⇒ ios;
iostream ⇒ istream, ostream;
ifstream ⇒ istream;
ofstream ⇒ ostream;
fstream ⇒ iostream;
strstream ⇒ iostream;

Примеры использования:

```
cout << "Сообщение" << endl;        \\печать строки
cout << "Address"   << (void *)ptr; \\указатель
cin.tie(&cout);     \\связывание, для печати
                    \\приглашения ранее запроса
cin.tie(NULL);      \\развязывание потока
ofstream outClnt("client.dat",ios::out); \\открыть
    \\файл для записи
ifstream inClnt("client.dat",ios::in);   \\открыть
                    \\файл для чтения
strstream ss
ss << "TEXT" << 23 << ends;
String msg( ss.str() );
```

Операции взятия из потока возвращают "false"при вводе признака конца файла. Для расширенной манипуляций с потоками допускается определение манипуляторов пользователя в виде `ostream &tab(ostream &output) { return output << \t; }`

2.11.1 Манипуляторы потока

■ **endl <iostream>**
ostream& endl(ostream& outs);
Перевод курсора на следующую строку;

■ **ends <iostream>**
ostream& ends(ostream& outs);
Вывести нулевой байт (символ конца строки);

2.11.2 Компонентные функции класса ios <iostream>

■ **bad**
int bad();
При ошибке возвращает ненулевое значение.

■ **bitalloc**
static long bitalloc();
Возвращает установки флагов. Полученное значение может быть использовано для очистки, установки или проверки флагов.

■ **clear**
void clear(int = 0);
Устанавливает состояние потока (обнулить или установить указанные биты).

eofbit - признак конца файла;
failbit - ошибка форматирования, но символы не утеряны;
badbit - потеря данных;
goodbit - ошибок нет;

■ **eof**
int eof();
Возвращает ненулевое значение, если имеет место условие конца файла (EOF).

■ **fail**
int fail();
Возвращает ненулевое значение, если операция обмена с потоком терпит неудачу.

■ **fill**
char fill();
char fill(char);
Устанавливает символ заполнения потока. Возвращает старое значение символа заполнения.

■ **flags**
long flags();
ong flags(long);
Устанавливает флаги форматирования. Возвращает ранее установленное значение флагов.

■ **good**
int good();
Возвращает ненулевое значение если не установлен ни один флаг состояния (ошибок нет).

■ **init**
*void init(streambuf *);*
Связывает ios с указанным <streambuf>.

■ **precision**
int precision();
int precision(int);
Устанавливает точность вещественных чисел. Возвращает предыдущее значение точности.

■ **rdbuf**
streambuf rdbuf();*
Возвращает указатель на буфер (объект класса <bufstream>), связанный с потоком.

■ **rdstate**
int rdstate();
Возвращает текущее состояние потока.

■ **setf**
long setf(long);
long setf(long setbits, long field);
Устанавливает флаги по значению параметра или сбрасывает те биты состояния, которые отмечены в <field>, и устанавливает указанные в <setbits>. Возвращает предыдущие значения флагов.

■ **setstate**
void setstate(int);
Устанавливает указанные биты состояния.

■ **tie**
ostream tie();*
ostream tie(ostream *);*
Организует поток, взаимосвязанный с потоком, на который указывает <ostream*>. Возвращает указатель на взаимосвязанный предыдущий поток, если такой есть.

■ **unself**
long unself(long);
Очищает указанные биты состояния потока. Возвращает предыдущее значение.

■ **width**
int width();
int width(int);
Устанавливает ширину. Возвращает предыдущее значение.

■ **xalloc**
static int xalloc();
———

2.11.3 Компонентные функции класса ostream <iostream>

■ **flush**
ostream& flush(ostream& outs);
Сброс строки из буферов в поток;

■ **put**
cout.put(char c);
ostream& put(char c);
Вставить в поток символ <c>

■ **seekp**
ostream& seekp(long beg);
ostream& seekp(long beg, seek_dir);
Перемещает указатель текущей позиции выходного потока.

■ **tellp**
long tellp();
Возвращает текущую позицию указателя записи для выходного потока.

■ **write**
*oatream& write(const signed char *string, int n);*
*ostream& write(const unsigned char *string, int n);*
Помещает в выходной поток <n> символов из массива, на который указывает <string>. Нуль-символы включаются в число переносимых символов.

2.11.4 Компонентные функции класса istream <iostream>

■ **gcount**
int gcount;
Возвращает число символов, извлеченных из потока последним обращением.

■ **get**
istream& get(char& c);
istream& get(char ptr, int len, char delim = '\n');*
istream& get(streambuf& sb, char delim = '\n');
Извлекает из входного потока символы.

■ **getline**
istream& getline(signed char ,int sizeof(char), '\n');
Тоже что и get, но символ-разделителя, также, помещается в принятую строку символов.

■ **ignore**
istream& ignore(int n = 1,int delim = EOF);
Пропускает до n символов входного потока. Останавливается, если встретился разделитель (второй параметр), по умолчанию равный EOF.

■ **peek**
int peek();
Извлекает следующий символ из входного потока не удаляя его в потоке.

■ **putback**
istream& putback(char);
Помещает символ назад во входной поток.

■ **read**
*istream& read(signed char *string, int n);*
*istream& read(unsigned char *string, int n);*
Извлекает из входного потока группу символов <n> и помещает их в массив <string>.

■ **seekg**
istream& seekg(long beg);
istream& seekg(long beg, seek_dir);
Перемещает указатель чтения входного потока.

■ **tellg**
long tellg();
Возвращает текущую позицию указателя чтения входного потока.

Таблица 2.6: **Флаги класса ios, управляющие форматированием ввода/вывода (манипуляторы потока)**

Имя	Назначение
app	Записать все данные в конец файла;
ate	Переместиться в конец исходного открытого файла. Данные могут быть записаны в любое место файла;
binary	Открыть файл для двоичного ввода или вывода;
dec	Десятичная система счисления (OCC=10);
fixed	Использовать формат 123.45 для вывода вещественных чисел (с фиксированной точкой);
hex	Шестнадцатеричная система счисления (OCC = 16);
in	Открыть файл для ввода;
internal	Поместить разделительные символы после знака или основания системы счисления (OCC);
left	Выравнивать по левой стороне поля;
oct	Восьмеричная система счисления (OCC= 8);
out	Открыть файл для вывода;
right	Выравнивать по правой стороне поля;
setfill(n)	Установка заполняющего символа <n>;
scientific	Использовать формат 1.2345E2 для вывода вещественных чисел (экспоненциальная или научная нотация);
setbase(n)	Установка значения основания <n>;
setprecision(n)	Установка точности <n>;
setw(n)	Установка ширины поля <n>;
showbase	Указывать OCC при выводе;
showpoint	Печатать десятичную точку и следующие за ней нули при выводе вещественных чисел;
showpos	Добавлять ' + ' при выводе положительных чисел;
skipws	Игнорировать символы разделители при вводе;
stdio	Флэшировать stdout, sfcderr после операции;
trunc	Отбрасывать содержимое файла, если он существует;
uppercase	Шестнадцатеричные цифры печатать в верхнем регистре;
unitbuf	Флэшировать потоки после операции;

2.12 Процессы.

■ **atexit (BSD) <stdlib.h, cstdlib>**
*int atexit(void (*function)(void));*
Регистрирует функцию, вызываемую при обычном завершении программы/процесса. Регистрация происходит цепочкой что позволяет регистрировать несколько функций.

■ **clone (LINUX) <sched.h>**
*int __clone(int (*fn) (void *arg), void *child_stack, int flags, void *arg);*
Клонирует новую задачу на базе функции <fn> со стеком <child_stack> флагами <flags> и аргументами <arg>.

■ **exec[c,l,e,p,v] (POSIX) <unistd.h>**
*int execl(const char *path, const char *arg, ...);*
*int execlp(const char *file, const char *arg, ...);*
*int execle(const char *path, const char *arg , ..., char * const envp[]);*
*int execv(const char *path, char *const argv[]);*
*int execvp(const char *file, char *const argv[]);*
*int execve(const char *filename, char *const argv[], char *const envp[]);*
Заменяет выполняющуюся программу другой программой. Отличия функций:
p - Производить поиск программы в каталогах описанных в PATH;
v - Принимать список аргументов программы в виде массива строковых указателей оканчивающихся NULL-указателем;
l - Принимает список аргументов переменного размера;
e - Принимать в дополнительном аргументе массив переменных сред;

■ **exit, _exit, _Exit (POSIX) <stdlib.h, unistd.h>**
void exit(int status);
void _Exit(int status);
exit - обычное завершение работы программы/процесса.
_exit - немедленное завершение работы программы/процесса.

■ **fork (POSIX) <sys/types.h; unistd.h>**
int fork();
Порождает дочерний процесс. Родительскому процессу возвращается ID дочернего процесса ,а дочернему 0.

■ **getpid, getppid (POSIX) <sys/types.h, unistd.h>**
pid_t getpid(void);
pid_t getppid(void);
getpid - возвращает идентификатор текущего процесса (PID). getppid - возвращает идентификатор родительского процесса(PPID).

■ **nice (BSD) <unistd.h>**
int nice(int inc);
Устанавливает фактор уступчивости вызывающего процесса в величину <inc> (-39..39). Чем фактор больше, тем ниже приоритет процесса.

■ **sched_setparam, sched_getparam (POSIX) <sched.h>**
*int sched_setparam(pid_t pid, const struct sched_param *p);*
*int sched_getparam(pid_t pid, struct sched_param *p);*
Установка/получение параметров диспетчеризации процесса.

■ **sched_setscheduler, sched_getscheduler (POSIX) <sched.h>**
*int sched_setscheduler(pid_t pid, int policy, const struct sched_param *p);*
int sched_getscheduler(pid_t pid);
Установка/получение политики диспетчеризации процесса.

■ **sched_get_priority_max, sched_get_priority_min (POSIX) <sched.h>**
int sched_get_priority_max(int policy);
int sched_get_priority_min(int policy);
Получение максимального/минимального значения приоритета процесса.

■ **sched_rr_get_interval (POSIX) <sched.h>**
*int sched_rr_get_interval(pid_t pid, struct timespec *tp);*
Получить SCHED_RR интервал для указанного процесса.

■ **sched_yield (POSIX) <sched.h>**
int sched_yield(void);
Отбать процесорное время другим процессам.

■ **setsid (POSIX) <unistd.h>**
pid_t setsid(void);
Вызывающий процесс становится ведущим в группе, ведущим процессом нового сеанса и не имеет контролирующего терминала.

■ **setpgid, getpgid, setpgrp, getpgrp (POSIX, BSD) <unistd.h>**
int setpgid(pid_t pid, pid_t pgid);
pid_t getpgid(pid_t pid);
int setpgrp(void);
pid_t getpgrp(void);
Устанавливает/получает идентификатор группы процессов

■ **system (ANSI,POSIX) <stdlib.h>**
*int system (const char * string);*
Выполняет, указанные в string, системные команды.

■ **wait, waitpid, wait3, wait4 (POSIX) <sys/types.h, sys/wait.h, sys/resource.h>**
*pid_t wait(int *status);*
*pid_t waitpid(pid_t pid, int *status, int options);*
*pid_t wait3(int *status, int options, struct rusage *rusage);*
*pid_t wait4(pid_t pid, int *status, int options, struct rusage *rusage);*
Приостанавливают выполнение процесса до тех пор, пока дочерний процесс не завершится, или до появления сигнала, который либо завершает текущий процесс, либо требует вызвать функцию-обработчик. Если дочерний процесс к моменту вызова функции уже завершился (так называемый "зомби"("zombie")), то функция немедленно возвращается. Параметр pid:
< -1 - нужно ждать любого дочернего процесса, идентификатор группы процессов которого равен абсолютному значению pid;
-1 - ожидание любого дочернего процесса;
0 - означает ожидание любого дочернего процесса, идентификатор группы процессов которого равен идентификатору текущего процесса;
> 0 - ожидание дочернего процесса, чей идентификатор равен pid;
В <rusage> записывается структура struct rusage, заданная в <sys/resource.h>, заполненная соответствующей информацией.

2.13 Сигналы.

■ **kill (POSIX) <sys/types.h, signal.h>**
int kill(pid_t pid, int sig);
Системный вызов kill используется для того, чтобы послать любой сигнал любому процессу или группе процессов. Если:
pid > 0 - сигнал sig посылается процессу pid.
pid = 0 - сигнал sig посылается всем процессам текущей группы.
pid = -1 - сигнал sig посылается всем процессам текущей группы, кроме первого.

■ **pause (SVr4, SVID, POSIX, X/OPEN, BSD 4.3) <unistd.h>**
int pause(void);
Ожидание сигнала.

■ **raise (ANSI) <signal.h>**
int raise(int sig);
Посылает сигнал <sig> текущему процессу.

■ **sigaction (POSIX) <signal.h>**
*int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);*
Используется для изменения действий процесса при получении соответствующего сигнала <signum>.

■ **signal (ANSI) <signal.h>**
*void (*signal(int signum, void (*sighandler)(int)))(int);*
Устанавливает функцию <sighandler> обработки сигнала <signum>. Таблица **2.7**

■ **sigpending (POSIX) <signal.h>**
*int sigpending(sigset_t *set);*
Определяет наличие ожидающих сигналов.

■ **sigprocmask (POSIX) <signal.h>**
*int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);*
Изменяет список блокированных в данный момент сигналов.

■ **sigsuspend (POSIX) <signal.h>**
*int sigsuspend(const sigset_t *mask);*
Временно изменяет значение маски блокировки сигналов процесса

на указанное в mask, и затем приостанавливает работу процесса до получения соответствующего сигнала.

■ **sigwait (POSIX) <signal.h, pthread.h>**
*int sigwait(const sigset_t *set, int *sig);*
Блокирует поток до генерации сигналов указанных в <set>. Номер сгенерированного сигнала помещается в <sig>.

Таблица 2.7: **Стандартные сигналы**

Сигнал	Назначение
_HUP (01)	Освобождение линии.
_INT (02)	Прерывание процесса.
_QUIT (03)	Выход.
_ILL (04)	Недостоверная инструкция.
_TRAP (05)	Трассировочное прерывание.
_IOT (06)	Машинная команда IOT.
_ABRT	–//–
_EMPT (07)	Машинная команда EMT.
_FPE (08)	Исключение floatpoint.
_KILL (09)	Уничтожение процесса (KILL).
_BUS (10)	Ошибка шины (bus error).
_SYS (12)	Ошибка сегментации памяти.
_PIPE (13)	Запись в канал, из которого некому читать.
_ALRM (14)	Будильник (alarm clock).
_PROF	Срабатывание профилирующего таймера. Устанавливается: setitimer(ITIMER_PROF, ..)
_VTALRM	Срабатывание виртуального таймера. Устанавл.: setitimer(ITIMER_VIRTUAL, ..)
_TERM (15)	Программный сигнал завершения.
_USR1 (16)	Пользовательский сигнал 1.
_USR2 (17)	Пользовательский сигнал 2.
_CHLD (18)	Завершение порожденного процесса.
_PWR (19)	Ошибка питания (power fail).
_WIND (20)	Изменение окна.
_PHONE (21)	Изменение строки состояния.
_POLL (22)	Возникновение опрашиваемого события.
NSIG (23)	Максимальное количество сигналов.
_CONT	Продолжения работы остановленого процесса.
_STOP	Сигнал останова.
_TSTR	Терминальный сигнал остановки (Ctrl Z).
_TTIN	Попытка ввода с терминала фоновым процессом.
_TTOUT	Попытка вывода на терминал фоновым процессом.
_URG	Поступление в буфер сокета срочных данных.
_XCPU	Превышение лимита процессорного времени.
_XFSZ	Превышение предела на размер файла.

2.14 Потоки

■ **pthread_attr_init (POSIX) <pthread.h>**
*int pthread_attr_init(pthread_attr_t *attr);*
Инициализация атрибутов <attr> канала, стандартными значениями.

■ **pthread_kill (POSIX) <pthread.h, signal.h>**
int pthread_kill(pthread_t thread, int signo);
Послать сигнал <signo> потоку <thread>.

■ **pthread_attr_destroy (POSIX) <pthread.h>**
*int pthread_attr_destroy(pthread_attr_t *attr);*
Удаление из памяти атрибутов <attr> ранее инициализированных.

■ **pthread_attr_setschedpolicy, pthread_attr_getschedpolicy (POSIX) <pthread.h>**
*int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);*
*int pthread_attr_getschedpolicy(const pthread_attr_t *attr, int *policy);*
Получить, установить тактику <policy> управления процессом <attr>. Тактика может быть:

SCHED_OTHER Регулярное, не РВ разделение;

SCHED_RR РВ, (round-robin);

SCHED_FIFO РВ, (first in first out);

———————————————-

■ **pthread_attr_setschedparam, pthread_attr_getschedparam (POSIX) <pthread.h>**
*int pthread_attr_setschedparam(pthread_attr_t *attr, const struct sched_param *param);*
*int pthread_attr_getschedparam(const pthread_attr_t *attr, struct sched_param *param);*
Установка/получение параметров(приоритета) <param> разделения времени для РВ тактик.

■ **pthread_attr_setdetachstate, pthread_attr_getdetachstate (POSIX) <pthread.h>**
*int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);*
*int pthread_attr_getdetachstate(const pthread_attr_t *attr, int *detachstate);*
Установка/получение статуса отсоединения потока.

■ **pthread_create (POSIX) <pthread.h>**
*int pthread_create(pthread_t * thread, pthread_attr_t *attr, void * (*start_routine)(void *), void * arg);*
Создает новый поток <thread> с атрибутами <attr> и запускает в потоке функцию <start_routine> с аргументами <arg>. Созданная задача разделяет код и память с родителем;

■ **pthread_detach (POSIX) <pthread.h>**
int pthread_detach(pthread_t th);
Делает указанный <th> поток в отсоединенное состояние.

■ **pthread_equal (POSIX) <pthread.h>**
int pthread_equal(pthread_t thread1, pthread_t thread2);
Сравнивает два потока на эквивалентность.

■ **pthread_exit (POSIX) <pthread.h>**
*void pthread_exit(void *retval);*
Прекращает выполнение вызвавшего функцию потока с кодом возврата в <retval>

■ **pthread_join (POSIX) <pthread.h>.**
*int pthread_join(pthread_t *th, void **thread_return);*
Ожидает завершение потока <th>. После завершения код возврата помещается в <thread_return>.

■ **pthread_self (POSIX) <pthread.h>**
pthread_t pthread_self(void);
Возвращает идентификатор текущего потока.

■ **pthread_sigprocmask (POSIX) <signal.h, pthread.h>**
*int pthread_sigprocmask(int how, const sigset_t *set, sigset_t *oldset);*
Изменяет список блокированных в данный момент сигналов для текущего потока.

2.14.1 Отмена потоков

■ **pthread_cancel (POSIX) <pthread.h>**
int pthread_cancel(pthread_t thread);
Отмена указанного <thread> потока.

■ **pthread_setcanceltype (POSIX) <pthread.h>**
*int pthread_setcanceltype(int type, int *oldtype);*
Установка типа отмены в <type> и сохранение текущего в <oldtype>.

■ **pthread_testcancel (POSIX) <pthread.h>**
void pthread_testcancel(void);
Установка точки отмена.

■ **pthread_setcancelstate (POSIX) <pthread.h>**
*int pthread_setcancelstate(int state, int *oldstate);*
Установка состояния отмены потока в <state> и сохранение текущего в <oldstate>.

2.14.2 Глобальные данные потоков

■ **pthread_key_create (POSIX) <pthread.h>**
*int pthread_key_create(pthread_key_t *key, void (*destr_function)(void *));*
Создание ключа <key>, новой глобальной переменной. Можно указать функцию очистки ключа <destr_function> при уничтожении потока.

■ **pthread_key_delete (POSIX) <pthread.h>**
int pthread_key_delete(pthread_key_t key);
Удаление ключа <key>.

■ **pthread_setspecific, pthread_getspecific (POSIX) <pthread.h>**
*int pthread_setspecific(pthread_key_t key, const void *pointer);*
*void *pthread_getspecific(pthread_key_t key);*
Запись/чтение глобальных потоковых переменных по ключу <key>.

2.14.3 Обычные потоковые семафоры

■ **sem_init, sem_destroy (POSIX) <semaphore.h>**
*int sem_init(sem_t *sem, int pshared, unsigned int value);*
*int sem_destroy(sem_t * sem);*
Инициализация/удаление семафора <sem>. <pshared> должен быть равен нулю. <value> - Стартовое значение семафора.

■ **sem_getvalue (POSIX) <semaphore.h>**
*int sem_getvalue(sem_t * sem, int * sval);*
Получить текущее состояние <sval> семафора <sem>.

■ **sem_post (POSIX) <semaphore.h>**
*int sem_post(sem_t * sem);*
Установка семафора (+1).

■ **sem_wait (POSIX) <semaphore.h>**
*int sem_wait(sem_t * sem);*
Ожидает ненулевого значения семафора. При вызове функции значение семафора уменьшается на единицу.

■ **sem_trywait (POSIX) <semaphore.h>**
*int sem_trywait(sem_t * sem);*
Неблокирующие ожидания ненулевого значения семафора.

2.14.4 Исключающие семафоры

■ **thread_mutex_init, pthread_mutex_destroy (POSIX) <pthread.h>**
*int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr);*
*int pthread_mutex_destroy(pthread_mutex_t *mutex);*
Инициализация/удаление исключающего семафора <mutex> с атрибутами <mutexattr>.

■ **int pthread_mutex_lock, pthread_mutex_unlock (POSIX) <pthread.h>**
*int pthread_mutex_lock(pthread_mutex_t *mutex)*
*; int pthread_mutex_unlock(pthread_mutex_t *mutex);*
Захват/освобождение исключающего семафора <mutex>. Если семафор занят поток блокируется до освобождения.

■ **pthread_mutex_trylock (POSIX) <pthread.h>**
*int pthread_mutex_trylock(pthread_mutex_t *mutex);*
Попытка захвата исключающего семафора <mutex>. Если семафор занят возвращается BUSY.

■ **pthread_mutexattr_init, pthread_mutexattr_destroy (POSIX) <pthread.h>**
*int pthread_mutexattr_init(pthread_mutexattr_t *attr);*
*int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);*
Создание/удаление атрибутов для <attr> исключающего семафора.

■ **pthread_mutexattr_setkind_np, pthread_mutexattr_getkind_np (POSIX) <pthread.h>**
*int pthread_mutexattr_setkind_np(pthread_mutexattr_t *attr, int kind);*
*int pthread_mutexattr_getkind_np(const pthread_mutexattr_t *attr, int *kind);*
Установка/получение типа исключающего семафора <kind>.

2.14.5 Сигнальные переменные

■ **pthread_cond_init, pthread_cond_destroy (POSIX) <pthread.h>**
*int pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t *cond_attr);*
*int pthread_cond_destroy(pthread_cond_t *cond);*
сохранение/удалении сигнальной переменной <cond>. <cond_attr> должна быть NULL.

■ **pthread_cond_signal, pthread_cond_broadcast (POSIX) <pthread.h>**
*int pthread_cond_signal(pthread_cond_t *cond);*
*int pthread_cond_broadcast(pthread_cond_t *cond);*
Разблокирует один/все потоки ожидающие сигнальной переменной <cond>;

■ **pthread_cond_wait (POSIX) <pthread.h>**
*int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);*
*int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime);*
Блокирует текущий поток на базе исключающего семафора <mutex> до изменения сигнальной переменной <cond> с таймаутом <abstime>.

2.15 IPC

■ **ftok <sys/types.h, sys/ipc.h>**
*key_t ftok(char *pathname, char proj);*
Преобразовывает имя файла и идентификатор проекта в ключ для системных вызовов.

2.15.1 Сообщения

■ **msgctl (SVr4, SVID) <sys/types.h, sys/ipc.h, sys/msg.h>**
*int msgctl(int msqid, int cmd, struct msqid_ds *buf);*
Послать команду <cmd> сообщению <msqid> с параметрами <buf>;

■ **msgget (SVr4, SVID) <sys/types.h, sys/ipc.h, sys/msg.h>**
int msgget(key_t key, int msgflg);
Возвращает дескриптор очереди сообщений для ключа <key> с правами доступа <msgflg>.

■ **msgsnd (SVr4, SVID) <sys/types.h, sys/ipc.h, sys/msg.h>**
*int msgsnd(int msqid, struct msgbuf *msgp, size_t msgsz, int msgflg);*
Отправить сообщение <msgp> длиной <msgsz> и с флагами <msgflg>, по дескриптору сообщения <msqid>.

■ **msgrcv (SVr4, SVID) <sys/types.h, sys/ipc.h, sys/msg.h>**
*ssize_t msgrcv(int msqid, struct msgbuf *msgp, size_t msgsz, long msgtyp, int msgflg);*
Получить сообщение из дескриптора сообщения msqid, в буфер <msgp> размером <msgsz> с типом <msgtyp> и флагами <msgflg>. Если <msgtyp>=0 – читать первое сообщение из очереди, если < 0 – получить сообщение с минимальным типом не менее msgtyp.

2.15.2 Семафоры

■ **semget (SVID) <sys/types.h, sys/ipc.h, sys/sem.h>**
int semget(key_t key, int nsems, int semflg);
Возвращает идентификатор семафора в соответствии с ключом <key>, опциями <semflg> и количеством семафором <nsems> (Табл 2.8).

■ **semctl (SVID) <sys/types.h, sys/ipc.h, sys/sem.h>**
int semctl(int semid, int semnum, int cmd, union semun arg);
Производит операции <cmd> (Табл 2.9) по управлению семафорами.

■ **semop (SVID) <sys/types.h, sys/ipc.h, sys/sem.h>**
*int semop(int semid, struct sembuf *sops, unsigned nsops);*
Производит операции над выбранными элементами из набора семафоров <semid>. Каждый из элементов <nsops> в массиве <sops> определяет операцию, производимую над семафором в структуре <struct sembuf>.

2.15.3 Разделяемая память

■ **shmget (SVID) <sys/ipc.h, sys/shm.h>**
int shmget(key_t key, int size, int shmflg);
Возвращает идентификатор разделяемой памяти в соответствии с ключом <key>, размером <size> и опциями <shmflg> (Табл 2.8).

■ **shmat, shmdt (SVID) <sys/types.h, sys/shm.h>**
*void *shmat(int shmid, const void *shmaddr, int shmflg);*
*int shmdt(const void *shmaddr);*
<shmat> - присоединяет сегмент разделяемой памяти <shmid> к сегменту данных вызывающего процесса. Адрес присоединяемого сегмента определяется функцией <shmaddr>.
<shmdt> - отстыковывает от сегмента данных вызывающего процесса сегмент разделяемой памяти, находящийся по адресу <shmaddr>.

■ **shmctl (SVID) <sys/ipc.h, sys/shm.h>**
*int shmctl(int shmid, int cmd, struct shmid_ds *buf);*
Производит операции <cmd> (Табл 2.9) по управлению разделяемыми сегментами памяти.

Таблица 2.8: **Флаги доступа к IPC**

Флаг	Назначение
IPC_CREAT	Создавать при отсутствии;
IPC_EXCL	Ошибка, если существует и используется PC_CREAT;
IPC_NOWAIT	Выход по ошибке отсутствия, без ожидания;
MSG_NOERROR	Не выходить по ошибке в длине сообщения, а обрезать его.
SHM_RN	Округление до ближайшей страницы.
SHM_RDONLY	Открыть разделяемую область в режиме только для чтения.
SEM_UNDO	Отменить действия при завершении процесса.

2.16 Каналы

■ **mkfifo (POSIX) <sys/types.h, sys/stat.h>**
*int mkfifo (const char *pathname, mode_t mode);*
Создает FIFO-файл <pathname> (именованный канал). Размер атомарного буфера составляет PIPE_BUF.

■ **pipe (POSIX) <unistd.h>**
int pipe(int filedes[2]);
Создает два файловых описателя, указывающих на неименнованый канал, и помещает их в массив filedes. filedes[0] - для чтения, filedes[1] - для записи.

■ **popen, pclose (POSIX) <stdio.h>**
*FILE *popen(const char *command, const char *type);*
*int pclose(FILE *stream);*
Открывает/закрывает канал с запускаемым процессом <command>.

2.17 Сокеты

■ **accept (BSD) <sys/types.h, sys/socket.h>**
*int accept(int s, struct sockaddr *addr, socklen_t *addrlen);*
Возвращает описатель сокета клиента пославшего запрос на сокет <s> с адресом <addr, addrlen>.

■ **bind (BSD) <sys/types.h, sys/socket.h>**
*int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);*
Связывает открытый сокет <sockfd> с именем длиной <addrlen> в <my_addr>.

■ **listen (POSIX) <sys/socket.h>**
int listen(int s, int backlog);
Прослушивает соединения на сокете <s> с длиной очереди полностью установленных сокетов <backlog>

■ **recv, recvfrom, recvmsg (BSD) <sys/uio.h sys/types.h sys/socket.h>**
*int recv(int s, void *buf, size_t len, int flags);*
*int recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);*
*int recvmsg(int s, struct msghdr *msg, int flags);*
Системные вызовы recvfrom и recvmsg используются для получения сообщений из сокета независимо от того, является ли сокет ориентированным на соединения или нет. Вызов recv обычно делается только через соединенный сокет.

■ **send, sendto, sendmsg (BSD, POSIX) <sys/types.h, sys/socket.h>**

Таблица 2.9: **Операции над IPC**

Операция	Описание
GETALL	Получение значений всех семафоров множества;
GETNCNT	Число процессов ожидающих ресурсов;
GETPID	Возвращает PID процесса выполнившего вызов <semop>;
GETVAL	Возвращает значение одного семафора из множества;
GETZCNT	Число процессов ожидающих 100% освобождения ресурса;
IPC_RMID	Удалить очередь из ядра;
IPC_STAT	Сохранить структуру с ID в буфере;
IPC_SET	Устанавливает ipc_perm структуры с ID;
SETALL	Устанавливает значения семафоров множества, взятые из элемента <array> объединения;
SETVAL	Устанавливает значение конкретного семафора как элемент <val> объединения;

*int send(int s, const void *msg, size_t len, int flags);*
*int sendto(int s, const void *msg, size_t len, int flags, const struct sockaddr *to, socklen_t tolen);*
*int sendmsg(int s, const struct msghdr *msg, int flags);*
send, sendto, и sendmsg используются для пересылки сообщений в другой сокет. send можно использовать, только если сокет находится в состоянии соединения, тогда как sendto и sendmsg можно использовать в любое время.

■ **socket (BSD) <sys/types.h, sys/socket.h>**
int socket(int domain, int type, int protocol);
Возвращает описатель нового сокета <socket>. <domain> - задает домен соединения (Табл 2.10).
<type> - задающий семантику коммуникации (Табл 2.11).
<protocol> - задает конкретный протокол, который работает с сокетом.

■ **shutdown (BSD) <sys/socket.h>**
int shutdown(int s, int how);
Приводит к закрытию всего полнодуплексного соединения или его части в сокете, связанном с описателем <s>.

■ **socketpair (BSD) <sys/types.h, sys/socket.h>**
int socketpair(int d, int type, int protocol, int sv[2]);
Функция создает пару неименнованых связанных сокетов в заданном домене <d> типа <type>, используя заданный протокол <protocol>. Описатели заданных сокетов возвращаются в sv[0] и sv[1].

■ **gethostbyname (BSD) <sys/socket.h>**
*struct hostent *gethostbyname(const char *name);*
Возвращает структуру типа hostent машине с именем name.

■ **htonl, htons, ntohl, ntohs (BSD)***uint32_t htonl(uint32_t hostlong);*
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
Переводят данные из хостового порядка расположения байтов в сетевой и наоборот

■ **inet_pton (BSD) <sys/types.h, sys/socket.h, arpa/inet.h>**
*int inet_pton(int af, const char *src, void *dst);*
Преобразует строку символов <src> в сетевой адрес (типа <af>), затем копирует полученную структуру с адресом в <dst>.

■ **inet_aton (BSD) <sys/socket.h, netinet/in.h, arpa/inet.h>**
*int inet_aton(const char *cp, struct in_addr *inp);*
Преобразовывает обычный вид IP-адреса **cp** (из номеров и точек) в двоичный код **inp**.

■ **inet_addr (BSD) <sys/socket.h, netinet/in.h, arpa/inet.h>**
*in_addr_t inet_addr(const char *cp);*
Преобразует обычный вид IP-адреса **cp** (из номеров и точек) в двоичный код в сетевом порядке расположения байтов.

■ **inet_network (BSD) <sys/socket.h, netinet/in.h, arpa/inet.h>**
*in_addr_t inet_network(const char *cp);*
Извлекает сетевой номер в хостовом порядке расположения байтов из адреса **cp**, записанном в виде номеров и точек.

■ **inet_ntoa (BSD) <sys/socket.h, netinet/in.h, arpa/inet.h>**
*char *inet_ntoa(struct in_addr in);*
Преобразует IP-адрес **in**, заданный в сетевом порядке расположения байтов, в стандартный строчный вид, из номеров и точек.

■ **inet_makeaddr (BSD) <sys/socket.h, netinet/in.h, arpa/inet.h>**
struct in_addr inet_makeaddr(int net, int host);
Создает IP-адрес в сетевом порядке расположения байтов, комбинируя номер сети **net** с локальным адресом **host** в сети **net** (оба в хостовом порядке расположения байтов).

■ **inet_lnaof (BSD) <sys/socket.h, netinet/in.h, arpa/inet.h>**
in_addr_t inet_lnaof(struct in_addr in);
Возвращает часть адреса для локального хоста из IP-адреса **in**. Адрес локального хоста возвращается в хостовом порядке размещения байтов.

■ **inet_netof (BSD) <sys/socket.h, netinet/in.h, arpa/inet.h>**
in_addr_t inet_netof(struct in_addr in);
Возвращает сетевую часть IP-адреса **in**. Сетевой номер возвращается в виде байтов, порядок которых определяется системой локального хоста.

Таблица 2.10: **Домены соединений**

Имя	Описание
PF_UNIX	Локальное соединение;
PF_LOCAL	
PF_INET	IPv4 протоколы Интернет;
PF_INET6	IPv6 протоколы Интернет;
PF_IPX	IPX - протоколы Novell;
PF_NETLINK	Устройство для взаимодействия с ядром;
PF_X25	Протокол ITU-T X.25 / ISO-8208;
PF_AX25	Протокол AX.25 - любительское радио;
PF_ATMPVC	ATM - доступ к низкоуровневым PVC;
PF_APPLETALK	Appletalk;
PF_PACKET	Низкоуровневый пакетный интерфейс;

Таблица 2.11: **Типы сокетов**

Имя	Описание
SOCK_STREAM	Двусторонний, надежных последовательных потоков байтов, с поддержкой соединения. Может также поддерживаться механизм внепоточных данных.
SOCK_DGRAM	Датаграммы (ненадежные сообщения с ограниченной длиной и не поддерживающие соединения).
SOCK_SEQPACKET	Последовательный двусторонний канал для передачи датаграмм с поддержкой соединений.
SOCK_RAW	Обеспечивает доступ к низкоуровневому сетевому протоколу.
SOCK_RDM	Обеспечивает надежную доставку датаграмм без гарантии, что они будут расположены по порядку.