

## Структурированные типы данных

Для простой программы достаточно использовать базовые типы данных. Но чаще всего намного лучше использовать структурированные типы данных. Это типы данных, которые формируются пользователем на основании базовых типов.

Одним из примеров структурированных типов данных является массив, набор элементов одного типа. Если необходимо, чтобы тип содержал несколько разных данных разных типов, используются структуры. Для работы с символьными данными используются строки.

### 1.1. Строки

В языке C++ существует два вида строк — строки, представляющие собой массив символов, завершающиеся нулевым байтом (иногда называемые **C-строки**), и строки, реализованные в виде класса.

#### 1.1.1. Строки в стиле C

Для того, чтобы объявить строку в стиле C, необходимо описать ее как массив символов, но при описании переменной зарезервировать память под нулевой байт: `char` *<имя массива> [ <длина строки + 1> ]*.

Следовательно, для переменной `str`, содержащей 10 символов, необходимо описать переменную вида `char str[11]` или `char *str`

Для работы с такими строками реализовано достаточное количество функций, которые описаны в библиотеке `<string.h>` или `<cstring>`:

**size\_t** — встроенный беззнаковый тип, используемый для работы с данными, связанными, в основном, с номером позиции символа в строке.

**size\_t strlen(char \*s1)** определяет длину строки.

Листинг 1.1.

```
1 char *s1 = "Hello.";
2 size_t k = strlen(s1);
3 cout << "Size of string is " << k << endl;
```

Результат: Size of string is 6

**char \*strcpy(char \*s1, char \*s2)** копирует строку **s2** в **s1**. Необходимо следить, чтобы размер строки **s1** был достаточным для копирования строки **s2**. Возвращает **s1**.

Листинг 1.2.

```
1 char s1[20];
2 char s2[] = "world";
3 strcpy(s1, s2);
4 cout << s1 << endl;
```

Результат: s1 = "world"

**char \*strncpy\_s(char \*s1, char \*s2, size\_t n)** копирует строку **s2** в **s1**, но не более *n* символов. Возвращает **s1**.

Листинг 1.3.

```
1 char s1[20];
2 char s2[] = "world";
3 strncpy_s(s1, s2, 3);
4 cout << s1 << endl;
```

Результат: s1 = "wor"

**char \*strcat(char \*s1, char \*s2)** добавляет строку **s2** в конец строки **s1**. Возвращает **s1**. Длина строки **s1** должна быть достаточной, чтобы добавить новую строку и ноль-символ.

Листинг 1.4.

```
1 char s1[20] = "Hello ";
2 char s2[] = "world";
3 strcat(s1, s2);
4 cout << s1 << endl;
```

Результат: s1 = "Hello world"

**char \*strncat\_s(char \*s1, char \*s2, size\_t n)** добавляет *n* символов строки **s2** в конец строки **s1**. Возвращает **s1**. Длина строки **s1** должна быть достаточной, чтобы добавить новую строку и ноль-символ.

Листинг 1.5.

```
1 char s1[20] = "Hello ";
2 char s2[] = "world";
3 strcat_s(s1, s2, 3);
4 cout << s1 << endl;
```

Результат: s1 = "Hello wor"

**char \*strcmp(char \*s1, char \*s2)** сравнивает строки s2 и s1 посимвольно.

Как только встретятся разные символы, будут проанализированы числовые коды этих символов. Чей код окажется больше, та строка и будет больше. Возвращает ноль, если строки одинаковы; отрицательное значение (чаще всего -1), если вторая строка больше, или первая строка по алфавиту расположена раньше; положительное значение (чаще всего 1), если первая строка больше, или вторая строка по алфавиту расположена раньше.

Если две строки имеют разную длину, но первые символы одинаковы, например, мужская и женская фамилии, то меньшей будет менее длинная строка, то есть, мужская фамилия будет расположена раньше женской.

Прописные буквы в таблице ASCII кодов расположены раньше строчных, следовательно, слова, написанные с прописной буквы, будут расположены раньше.

Листинг 1.6.

```
1 char *s1 = "Hello";
2 char *s2 = "Hello";
3 cout << strcmp(s1, s2) << endl;
4 s1 = "Hello";
5 s2 = "hello";
6 cout << strcmp(s1, s2) << endl;
7 s1 = "world";
8 s2 = "World";
9 cout << strcmp(s1, s2) << endl;
```

Результат: 0, -1, 1.

**char \*strncmp(char \*s1, char \*s2, size\_t n)** сравнивает первые n символов строк s2 и s1.

Листинг 1.7.

```
1 char *s1 = "HeLlO";
2 char *s2 = "HeLLo";
3 cout << strcmp(s1, s2, 3) << endl;
4 s1 = "HeLlO";
5 s2 = "Hello";
6 cout << strcmp(s1, s2, 3) << endl;
7 s1 = "World";
8 s2 = "WoRld";
9 cout << strcmp(s1, s2, 3) << endl;
```

Результат: 0, -32, 32.

**char \*strchr(char \*s1, char symbol)** находит первое вхождение символа `symbol` в строке `s1`.

**char \*strrchr(char \*s1, char symbol)** находит последнее вхождение символа `symbol` в строке `s1`.

В обоих случаях результат работы программы — строка, начинающаяся с позиции найденного символа. Чтобы определить позицию на которой находится символ, необходимо из найденного указателя (имя строки является также указателем на первый элемент строки) вычесть указатель первоначальной строки. Результат — позиция символа `symbol` в ноль-нотации.

Листинг 1.8.

```
1 char *s1 = "Hello";
2 char *k = strchr(s1, 'l');
3 cout << k << endl; //первое вхождение
4 cout << k - s1 << endl; //позиция
5 char *r = strrchr(s1, 'l');
6 cout << r << endl; //последнее вхождение
7 cout << r - s1 << endl; //позиция
```

Результат:

```
llo
2
lo
3
```

**size\_t \*strcspn(char \*s1, char \*s2)** находит первое вхождение любого из символов **s2** в строке **s1**. Возвращает позицию найденного первого символа.

**char \*strpbrk(char \*s1, char \*s2)** находит первое вхождение любого из символов **s2** в строке **s1**. Возвращает указатель на найденный символ.

Листинг 1.9.

```
1 char *s1 = "Hello. My phone 8-917-123-23-23";
2 char *s2 = "1234567890";
3 size_t k = strcspn(s1, s2); //позиция первой цифры
4 cout << "First digit " << k << endl;
5 char *s3 = strpbrk(s1, s2); //указатель на первую цифру
6 cout << s3 << endl; //номер телефона
```

Результат:

```
First digit 16
8-917-123-23-23
```

### 1.1.2. Класс **string**

Для работы с этим типом строк необходимо подключить библиотеку **<string>**. Большая часть данных построена в соответствии с методами обобщенного программирования и заголовки большинства функций соответствуют библиотеке **STL**.

Для описания используется тип **string**. Так как этот тип представляет собой класс, то, при описании переменных он инициализируется конструктором по умолчанию, а именно, пустой строкой.

Для ввода строки можно воспользоваться оператором **<<**. Но данный оператор определяет пробел в качестве разделителя, но в случае строки пробел — это такой же символ, как и все остальные. Поэтому, для работы со строками лучше воспользоваться функцией **getline()**. Данная функция считывает всю строку целиком. Имеет два варианта:

**istream& getline(istream& instream, string& str, char razd)** — считывает из входного потока **instream** набор символов до разделителя **razd** и записывает данные в строку **str**.

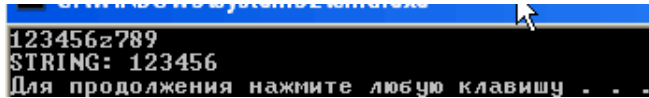
Листинг 1.10.

```

1 string str;
2 getline(cin, str, 'z');
3 cout << "STRING: " << str << endl;

```

Результат:



```

123456z789
STRING: 123456
Для продолжения нажмите любую клавишу . . .

```

**istream& getline(istream& istream, string& str)** — считывает из входного потока **istream** набор символов и записывает данные в строку **str**. В качестве разделителя используется символ перевода строки.

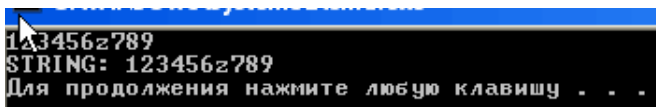
Листинг 1.11.

```

1 string str;
2 getline(cin, str);
3 cout << "STRING: " << str << endl;

```

Результат:



```

123456z789
STRING: 123456z789
Для продолжения нажмите любую клавишу . . .

```

Вывод на экран происходит с помощью оператора **<<**.

В качестве встроенного типа для работы с числовыми данными, связанными со строками, используется тип **string::size\_type**. Он соответствует беззнаковому типу **unsigned int**. Поскольку данный тип не входит в пространство имен **std**, при его описании необходимо использовать оператор прямого доступа **::**.

В функциях, в которых необходимо вывести номер позиции, результатом будет либо позиция нужного символа, либо специальная константа **string::npos**, если данного символа нет. Обычно **npos** равна максимально возможному значению типа (во всех разрядах стоят единицы).

Для обращения к символу строки используются либо квадратные скобки (**str[k]**), либо оператор **str.at(k)**. Оператор **at()** в случае обращения в несуществующему символу генерирует исключение **out\_of\_range**.

Существует достаточное количество встроенных функций, которые могут использоваться при работе со строками. Эти функции являются членами класса, поэтому вызываются `str.function()`, где вместо `function` должна быть любая из перечисленных ниже функций:

**size\_type size(), length()** — обе функции возвращают длину строки. `length()` является аналогом `strlen` для строк в стиле C. `size()` используется для всех контейнерных классов библиотеки STL.

**bool empty()** проверяет, является ли строка пустой. Возвращает `true`, если строка пустая и `false` в противном случае

Листинг 1.12.

```
1 string str;
2 getline(cin, str); //ввод строки
3 string::size_type n = str.size(); //размер
4 string::size_type n1 = str.length(); // размер
5 cout << n << " " << n1 << endl;
6 cout << str.empty() << endl; //проверка на пустоту
7 string str1;
8 cout << str1.empty() << endl;
```

Результат:

```
123456789
9 9
0
1
```

**Сравнение строк** Синтаксис: `s.compare(s1)`. Возвращает 0, если `s` и `s1` равны; отрицательное значение, если `s` меньше `s1`; положительное — `s` больше `s1`.

Можно воспользоваться операциями сравнения. Сравнение происходит аналогично, только возвращает логический тип.

Листинг 1.13.

```
1 string str;
2 getline(cin, str); //ввод строки
3 string str1;
4 getline(cin, str1);
```

```
5 cout << str.compare(str1) << endl;
```

Результат:

```
123456789
```

```
123455
```

```
1
```

или

```
123456
```

```
123456789
```

```
-1
```

### **Очистка строки** Синтаксис:

```
s1.erase();//удаление всех символов
```

```
s1.clear();//очистка строки
```

### **Вставка, удаление и замена символов**

Присоединение:

- присоединение строки `s2` в конец строки `s1`:  
`s1 += s2` или `s1.append(s2)`.
- присоединение подстроки `s2` от элемента `a` до элемента `b` в конец строки `s1`:  
`s1.append(s2, a, b)`.
- присоединение подстроки `s2` от элемента `a` до конца строки в конец строки `s1`:  
`s1.append(s2, a, string::npos)`.

Вставка символов:

- вставка строки `s2` перед символом, расположенном на позиции `pos`:  
`s1.insert(pos, s2)`.  
`s2` обязательно должна иметь тип `string`.



- вставка `n` символов строки `s2`, начиная с позиции `pos2` перед символом, расположенном на позиции `pos` строки `s1`:

```
s1.insert(pos, s2, pos2, n).
```

`s2` обязательно должна иметь тип `string`.

- вставка `n` символов `ch` перед символом, расположенном на позиции `pos` строки `s1`:

```
s1.insert(pos, n, ch).
```

`ch` обязательно должна иметь тип `char`.

Замена символов:

- замена `n` символов строки `s2`, начиная с позиции `pos` строкой `s2`:

```
s1.replace(pos, n, s2).
```

`s2` обязательно должна иметь тип `string`.

- замена `n` символов строки `s2`, начиная с позиции `pos` `n2` символами строки `s2`, начиная с позиции `pos2`:

```
s1.replace(pos, n, s2, pos2, n2).
```

`s2` обязательно должна иметь тип `string`.

- замена `n` символов строки `s2`, начиная с позиции `pos` `n2` символами `ch`:

```
s1.replace(pos, n, n2, ch).
```

`ch` обязательно должна иметь тип `char`.

Удаление символов:

- удаление всех символов строки `s1`, начиная с позиции `pos`:

```
s1.erase(pos)
```

- удаление символов строки `s1`, начиная с позиции `first` и заканчивая позицией `last`:

```
s1.erase(first, last)
```

Пример:

Листинг 1.14.

```
1 #include <iostream>
```

```

2 #include<string>
3
4 using namespace std;
5
6 int main(){
7     string s1 = "Hello ";
8     string s2 = "world";
9     s1.append(s2); //добавление s2 в конец s1
10    cout << "Append all: " << s1 << endl;
11    s1 = "Hello";
12    s1.append(s2, 1, 3); //добавление "orl"в конец s1
13    cout << "Append 1-3: " << s1 << endl;
14    s1 = "Hello";
15    s1.append(s2, 2, string::npos); //добавление подстроки s2, начиная с 2
        позиции
16    cout << "Append 2-end: " << s1 << endl;
17    s1 = "Hello";
18    s1.insert(2, s2); //вставка s2 во вторую позицию s1
19    cout << "Insert all: " << s1 << endl;
20    s1 = "Hello";
21    s1.insert(2, s2, 1, 3); //вставка "orl"во вторую позицию s1
22    cout << "insert 1-3: " << s1 << endl;
23    s1 = "Hello";
24    char ch = 'x';
25    s1.insert(2, 4, ch); //вставка 4 'x' во вторую позицию s1
26    cout << "insert char: " << s1 << endl;
27    s1 = "Hello";
28    s1.replace(1, 3, s2); //замена "ell"строкой s2
29    cout << "Replace all: " << s1 << endl;
30    s1 = "Hello";
31    s1.replace(1, 3, s2, 1, 3); //замена "ell"подстрокой "orl"
32    cout << "Replace 1-3: " << s1 << endl;
33    s1 = "Hello";
34    ch = 'x';
35    s1.replace(1, 3, 5, ch); //замена "ell"5 символами 'x'
36    cout << "Replace char: " << s1 << endl;
37    s1 = "Hello";
38    s1.erase(3); //удаление символов, начиная с 3 позиции
39    cout << "Erase 3: " << s1 << endl;
40    s1 = "Hello";

```

```

41  s1.erase(1, 3); //удаление "ell"
42  cout << "Erase 1-3: " << s1 << endl;
43  return 0;
44  }

```

Результат:

**Поиск в строке** Результатом работы будет переменная типа `size_type`.

Поиск первого вхождения строки:

- поиск первого вхождения строки `s2` в строке `s1`. Результат — позиция первого символа строки `s2` в строке `s1`:

```
s1.find(s2)
```

- поиск первого вхождения строки `s2` в строке `s1`, начиная с позиции `pos`. Результат — позиция первого символа строки `s2` в строке `s1`:

```
s1.find(s2,pos)
```

- поиск первого вхождения символа `ch`:

```
s1.find(ch)
```

Поиск последнего вхождения строки:

- поиск последнего вхождения строки `s2` в строке `s1`. Результат — позиция первого символа строки `s2` в строке `s1`:

```
s1.rfind(s2)
```

- поиск первого вхождения строки `s2` в строке `s1`, начиная с позиции `pos`. Результат — позиция первого символа строки `s2` в строке `s1`:

```
s1.rfind(s2,pos)
```

- поиск первого вхождения первых `n` символов строки `s2` в строке `s1`, начиная с позиции `pos`. Результат — позиция первого символа строки `s2` в строке `s1`:  
`s1.rfind(s2, pos, n)`
- поиск первого вхождения символа `ch`:  
`s1.rfind(ch)`

Поиск первого вхождения любого символа строки `s2`:

- поиск первого вхождения любого символа строки `s2` в строке `s1`:  
`s1.find_first_of(s2)`
- поиск первого вхождения любого символа строки `s2` в строке `s1`, начиная с позиции `pos`.  
`s1.find_first_of(s2, pos)`

Поиск последнего вхождения любого символа строки `s2`:

- поиск последнего вхождения любого символа строки `s2` в строке `s1`:  
`s1.find_last_of(s2)`
- поиск последнего вхождения любого символа строки `s2` в строке `s1`, начиная с позиции `pos`:  
`s1.find_last_of(s2, pos)`

Поиск первого вхождения любого символа, не входящего в строку `s2`:

- поиск первого вхождения любого символа, не входящего в строку `s2`, в строке `s1`:  
`s1.find_first_not_of(s2)`
- поиск первого вхождения любого символа, не входящего в строку `s2`, в строке `s1`, начиная с позиции `pos`:  
`s1.find_first_not_of(s2, pos)`

Поиск последнего вхождения любого символа, не входящего в строку `s2`:

- поиск последнего вхождения любого символа, не входящего в строку `s2`, в строке `s1`:  
`s1.find_last_not_of(s2)`

- поиск последнего вхождения любого символа, не входящего в строку s2, в строке s1, начиная с позиции pos:

`s1.find_last_not_of(s2,pos)`

Пример:

Листинг 1.15.

```
1 #include <iostream>
2 #include<string>
3
4 using namespace std;
5
6 int main(){
7     string s1 = "Hello world! Hello, world! Hello world, world. Hello";
8     string s2 = "world";
9     cout << s1 << endl;
10    cout << s2 << endl;
11    string::size_type k = s1.find(s2); //поиск строки s2
12    cout << "Find <<world>> in begin " << k << endl;
13    string::size_type pos = 7;
14    k = s1.find(s2, pos); //поиск строки s2, начиная с pos
15    cout << "Find <<world>> after 7th position " << k << endl;
16    char ch = 'l';
17    k = s1.find(ch); //поиск символа
18    cout << "Find first char <<l>> " << k << endl;
19    k = s1.rfind(s2); //поиск строки s2 с конца
20    cout << "Find <<world>> in end " << k << endl;
21    pos = 7;
22    k = s1.rfind(s2, pos); //поиск строки s2, начиная с pos
23    cout << "Find <<world>> before 7th position " << k << endl;
24    ch = 'l';
25    k = s1.rfind(ch); //поиск символа
26    cout << "Find last char <<l>> " << k << endl;
27    k = s1.find_first_of(s2); //поиск первого вхождения любого символа s2
28    cout << "Find first any symbols of s2 " << k << endl;
29    k = s1.find_first_of(s2, pos); //поиск первого вхождения любого символа s2,
        начиная с pos
30    cout << "Find first any symbols of s2 after 7th position " << k << endl;
31    k = s1.find_last_of(s2); //поиск последнего вхождения любого символа s2
32    cout << "Find last any symbols of s2 " << k << endl;
```

```

33 k = s1.find_last_of(s2, pos); //поиск последнего вхождения любого символа s2,
    начиная с pos
34 cout << "Find last any symbols of s2 before 7th position " << k << endl;
35 k = s1.find_first_not_of(s2); //поиск первого вхождения любого символа не из
    s2
36 cout << "Find first any symbols except of s2 " << k << endl;
37 k = s1.find_first_not_of(s2, pos); //поиск первого вхождения любого символа не
    из s2, начиная с pos
38 cout << "Find first any symbols except of s2 after 7th position " << k
    << endl;
39 k = s1.find_last_of(s2); //поиск последнего вхождения любого символа не из
    s2
40 cout << "Find last any symbols except of s2 " << k << endl;
41 k = s1.find_last_of(s2, pos); //поиск последнего вхождения любого символа не
    из s2, начиная с pos
42 cout << "Find last any symbols except of s2 before 7th position " << k
    << endl;
43 return 0;
44 }

```

Результат:

```

C:\WINDOWS\system32\cmd.exe
Hello world? Hello, world? Hello world, world. Hello
world
Find <<world>> in begin 6
Find <<world>> after 7th position 20
Find first char <<1>> 2
Find <<world>> in end 40
Find <<world>> before 7th position 6
Find last char <<1>> 50
Find first any symbols of s2 2
Find first any symbols of s2 after 7th position 7
Find last any symbols of s2 51
Find last any symbols of s2 before 7th position 7
Find first any symbols except of s2 0
Find first any symbols except of s2 after 7th position 11
Find last any symbols except of s2 51
Find last any symbols except of s2 before 7th position 7
Для продолжения нажмите любую клавишу . . .

```

## Выделение и слияние подстрок

Выделение подстрок. Результат работы — переменная типа `string`.

- Возврат копии строки `s1`:

```
s1.substr()
```

- Возврат копии строки `s1`, начиная с позиции `pos`:

```
s1.substr(pos)
```

- Возврат `n` символов строки `s1`, начиная с позиции `pos`:

```
s1.substr(pos, n)
```

Слияние подстрок происходит с помощью оператора `+`. Результат работы — переменная типа `string`:

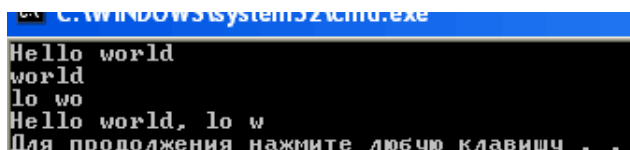
```
s2 = s1 + s2
```

Пример:

Листинг 1.16.

```
1 #include <iostream>
2 #include<string>
3
4 using namespace std;
5
6 int main(){
7     string s1 = "Hello world";
8     string s2 = s1.substr(); //копия строки s1
9     cout << s2 << endl;
10    string ::size_type pos = 6;
11    s2 = s1.substr(pos); //копия, начиная с pos
12    cout << s2 << endl;
13    string ::size_type n = 5;
14    pos = 3;
15    s2 = s1.substr(pos, n); //копия n символов, начиная с pos
16    cout << s2 << endl;
17    s2 = s1 + ", " + s1.substr(3, 4); //слияние строк
18    cout << s2 << endl;
19    return 0;
20 }
```

Результат:



```
C:\WINDOWS\system32\cmd.exe
Hello world
world
lo wo
Hello world, lo w
Для продолжения нажмите любую клавишу...
```

### 1.1.3. Некоторые функции для работы со строками

Иногда необходимо иметь возможность перевести строку из переменной типа `string` в строку в стиле C, например, чтобы работать с цифрами. Для этого используется функция `c_str()`. Результат — строка типа `const char*`.

Работа с числами определена только для строк в стиле C. Для этой цели используются функции `atoi(str)`, `atol(str)`, `atof(str)`. Первые два служат для перевода в целочисленные переменные (`atoi(str)` — в тип `int`, `atol(str)` — в тип `long`.) `atof` служит для тип `double`.

Функции работают следующим образом: считываются посимвольно элементы до тех пор, пока эти элементы можно интерпретировать как число соответствующего типа (для целых типов — набор цифр, для вещественных — может встретиться символ точки.) Строка обязательно должна начинаться с цифры или символов — или +.

Если считать невозможно, результатом будет ноль. В случае целочисленных типов для больших чисел произойдет переполнение и результатом работы функции будет `INT_MAX` или `INT_MIN`.

Пример:

Листинг 1.17.

```
1 #include <iostream>
2 #include<string>
3
4 using namespace std;
5
6 int main(){
7     string s1 = "123.456";
8     const char *s2 = s1.c_str(); //C-строка
9     cout << "string: " << s1 << endl;
10    int k = atoi(s2); //перевод в int
11    cout << "int: " << k << endl;
12    double k1 = atof(s2); // перевод в double
13    cout << "double: " << k1 << endl;
14    long k2 = atol(s2); //перевод в long
15    cout << "long : " << k2 << endl;
16    s1 = "dd1234";
17    s2 = s1.c_str(); //C-строка
18    cout << "string: " << s1 << endl;
```



```

19  k = atoi(s2); //перевод в int
20  cout << "int: " << k << endl;
21  k1 = atof(s2); // перевод в double
22  cout << "double: " << k1 << endl;
23  k2 = atol(s2); //перевод в long
24  cout << "long : " << k2 << endl;
25  s1 = "1234567889123456";
26  s2 = s1.c_str(); //C-строка
27  cout << "string: " << s1 << endl;
28  k = atoi(s2); //перевод в int
29  cout << "int: " << k << endl;
30  k1 = atof(s2); // перевод в double
31  cout << "double: " << k1 << endl;
32  k2 = atol(s2); //перевод в long
33  cout << "long : " << k2 << endl;
34  return 0;
35  }

```

Результат:

```

C:\WINDOWS\system32\cmd.exe
string: 123.456
int: 123
double: 123.456
long : 123
string: dd1234
int: 0
double: 0
long : 0
string: 1234567889123456
int: 2147483647
double: 1.23457e+015
long : 2147483647
Для продолжения нажмите любую клавишу . . .

```

Существуют также функции для работы с символами. Большая часть является булевыми функциями, служат в основном для определения типа символа (буква, цифра, печатный символ и т. д.):

Функция	Результат
isalpha(ch)	Латинские буквы в верхнем и нижнем регистре
isupper(ch)	Латинские буквы в верхнем регистре
islower(ch)	Латинские буквы в нижнем регистре
isdigit(ch)	Десятичные цифры: '0' .. '9'
isxdigit(ch)	Шестнадцатиричные цифры: '0' .. '9', 'a' .. 'f', 'A' .. 'F'
isspace(ch)	Символы-разделители, в основном используется для поиска пробела
isctrl(ch)	Управляющие символы (0..31 и 127 символы в таблице ASCII )
ispunct(ch)	Знаки пунктуации
isalnum(ch)	Латинские буквы и десятичные цифры
isprint(ch)	Символы, доступные для печати
isgraph(ch)	Латинские буквы, десятичные цифры и знаки пунктуации
toupper(ch)	Эквивалент ch в верхнем регистре
tolower(ch)	Эквивалент ch в нижнем регистре

*Пример 1.1.* Дано предложение, содержащее слова, разделенное пробелами и знаками препинания. Найти слово максимальной длины.

Листинг 1.18.

```

1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 int main(){
7     string s1, smax;
8     string razdel = "!?, .:;- "; //знаки пунктуации и пробел
9     getline(cin, s1);           //ввод строки
10    s1 += ' ';                   //добавляем точку в конец строки
11    string::size_type pos = 0, k;
12    string::size_type max = 0;
13    k = s1.find_first_of(razdel); //ищем конец первого слова
14    while(k != string::npos){    //пока находятся слова
15        string word = s1.substr(pos, k - pos); //копия слова
16        if (word.length() > max){ //ищем слово max длины
17            max = word.length();
18            smax = word;

```

```

19     }
20     if(ispunct(s1[k])) pos = k + 2; //если нашли знак препинания
21     else pos = k + 1;
22     k = s1.find_first_of (razdel, pos); //ищем конец следующего слова
23 }
24     cout << "max word " << smax << endl;
25 return 0;
26 }

```

Результаты работы программы:

Hello, my name is Alexandr.

max word Alexandr

□

## 1.2. Структуры

Массив — это набор элементов одинакового типа. Структура — это набор элементов произвольных типов.

Синтаксис структуры:

```

struct <имя типа> {
    <тип 1> <имя 1>
    <тип 2> <имя 2>
    <тип 3> <имя 3>
    <тип N> <имя N>
};

```

Создается новый тип, состоящий из полей (в английской версии — members). Описываются поля — имя и тип переменной. Определение структуры заканчивается точкой с запятой после закрывающей фигурной скобки.

Переменные созданного типа можно объявлять так же как и остальные типы. Единственная операция, доступная для переменной типа **struct** — операция присвоения одной переменной другой. Присвоение означает, что каждому полю одной переменной присваивается значение соответствующих полей другой переменной.

Индивидуальные поля достижимы либо с помощью операции «точка», либо, в случае использования указателей, с помощью «операции разыменования указателей на структуры» ( $\rightarrow$ ).

Пример 1.2. Дана последовательность точек на плоскости, заданных своими координатами. Подсчитать количество треугольников, которые можно создать, используя эти точки.

Листинг 1.19.

```
1 #include <iostream>
2 #include<math.h>
3
4 using namespace std;
5
6 struct point{//структура точка
7     double x, y;
8 };
9
10 point *create(int &n){//создание массива точек
11     cout << "n = ";
12     cin >> n;
13     point *a = new point[n]; //массив точек
14     for(int i = 0; i < n; i++){
15         cout << "Input " << i << " point coordinates (x, y): \n";
16         cin >> a[i].x >> a[i].y;
17     }
18     return a;
19 }
20
21 double dlina(point a, point b){ //расстояние между двумя точками
22     return sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
23 }
24
25 bool treug(double a, double b, double c){//существование треугольника
26     return (a + b > c && a + c > b && c + b > a);
27
28 }
29
30 int kol(point *a, int n){//число треугольников
31     int kol = 0;
32     for (int i = 0; i < n - 2; i++)
33         for(int j = i + 1; j < n - 1; j++)
34             for(int k = j + 1; k < n; k++){
35                 double ab = dlina(a[i], a[j]);
36                 double ac = dlina(a[i], a[k]);
```

```

37     double bc = dlina(a[j], a[k]);
38     if (treug(ab, ac, bc)) {
39         kol++;
40         cout << i << ", " << j << ", " << k << endl;
41     }
42 }
43 return kol;
44 }
45
46 int main(){
47     int n;
48     point *a = create(n);
49     cout << kol(a, n) << endl;
50     return 0;
51 }

```

□

### 1.3. Пример работы со строками и структурами

*Пример 1.3.* Дана дата в виде `dd.mm.yyyy`. Вывести дату следующего дня, используя строки и структуры.

Для удобства, введем дату в виде структуры:

```

struct date{
    int day;
    int month;
    int year; };

```

Данные вводятся с клавиатуры как строка. Функция `int correct(string)` проверяет корректность введенных данных как на ввод (2 цифры, потом любой символ, 2 цифры, любой символ, 4 цифры), так и правильность ввода (день от 1 до 31, месяц от 1 до 12, не было нулевого года).

Функция `date STR_date(string)` переводит данные из строки в структуру. Предполагается, что строка корректна.

Функции `vis` и `EndOfMonth` определяют високосный ли год и сколько дней в текущем месяце.

Функция `Next` определяет дату следующего дня: если текущий день не последний в месяце, просто прибавляем единицу к значению поля `day`; последний день месяца, но

не декабрь — значение поля `day` равно единице, а значение поля `month` увеличиваем на единицу; если 31 декабря — значения полей `day` и `month` равны единице, а значение поля `year` увеличиваем на единицу.

Функция `print` выводит данные на экран, добавляя нули в начале, если числа меньших разрядов.

Листинг 1.20.

```
1 #include<iostream>
2 #include<string>
3
4 using namespace std;
5
6 struct date{
7     int day;
8     int month;
9     int year;
10 };
11
12 date STR_date(string str){//перевод из строки в дату
13     date data;
14     //--day-----
15     string str1 = str.substr(0, 2);
16     const char *sd = str1.c_str();
17     int dd = atoi(sd);
18     data.day = dd;
19     //--month-----
20     str1 = str.substr(3, 2);
21     sd = str1.c_str();
22     dd = atoi(sd);
23     data.month = dd;
24     //--year-----
25     str1 = str.substr(6, 4);
26     sd = str1.c_str();
27     dd = atoi(sd);
28     data.year = dd;
29
30     return data;
31 }
32
```

```

33 bool vis(int y){//високосный год
34     return (y%4 == 0 && y%100 != 0) || y%400 == 0;
35 }
36
37 int EndOfMonth(int m, int y){//конец месяца
38     switch (m) {
39         case 1: case 3: case 5:
40         case 7: case 8: case 10: case 12: return 31;
41         case 4: case 6: case 9: case 11: return 30;
42         case 2: if(vis(y)) return 29;
43             else return 28;
44     }
45 }
46
47 date Next(date x){//следующий день
48     date y = x;
49     if(x.day < EndOfMonth(x.month, x.year)) y.day = x.day + 1;//не конец месяца
50     else if(x.month < 12){//конец месяца, но не конец года
51         y.day = 1;
52         y.month = x.month + 1;
53     }
54     else{//31 декабря
55         y.day = 1;
56         y.month = 1;
57         y.year = x.year + 1;
58     }
59     return y;
60 }
61
62 void print(date x){//печать данных на экран
63     if (x.day < 10) cout << "0" << x.day << ".";
64     else cout << x.day << ".";
65     if (x.month < 10) cout << "0" << x.month << ".";
66     else cout << x.month << ".";
67     if (x.year < 10) cout << "00" << x.year;
68     else if (x.year < 100) cout << "0" << x.year;
69     else if (x.year < 1000) cout << "0" << x.year;
70     else cout << x.year;
71     cout << endl;
72 }

```

```

73
74 int correct(string str){//проверка корректности данных
75     if (str.length() != 10) return -1;
76     for(string::size_type i = 0; i < str.length(); i++)
77         if (i != 2 && i != 5){
78             if (!isdigit (str[i])) return -1;
79         }
80     date data = STR_date(str);
81     if (data.day <= 0 || data.day > EndOfMonth(data.month, data.year)) return -2;
82     if (data.month <= 0 || data.month > 12) return -3;
83     if (data.year == 0) return -4;
84     return 1;
85 }
86
87 int main(){
88     string str;
89     cout << "Input date as dd.mm.yyyy\n";
90     getline(cin, str);
91     int fl = correct(str);
92     if (fl == -1) cout << "Error input\n";
93     else if (fl == -2) cout << "Error day\n";
94     else if (fl == -3) cout << "Error month\n";
95     else if (fl == -4) cout << "Error year\n";
96     else {
97         date data = STR_date(str);
98         date N_date = Next(data);
99         print(N_date);
100     }
101     return 0;
102 }

```

Результаты работы программы:



Ввод	Вывод
12/05/2001	13.05.2001
28.02.2000	29.02.2000
28.02.1900	01.03.1900
31/12/0005	01.01.0006
31.5.1000	Error input
31/31/2000	Error month
29.02.1900	Error day

