Лекция 4. Строки, структуры, объединение, перечисление

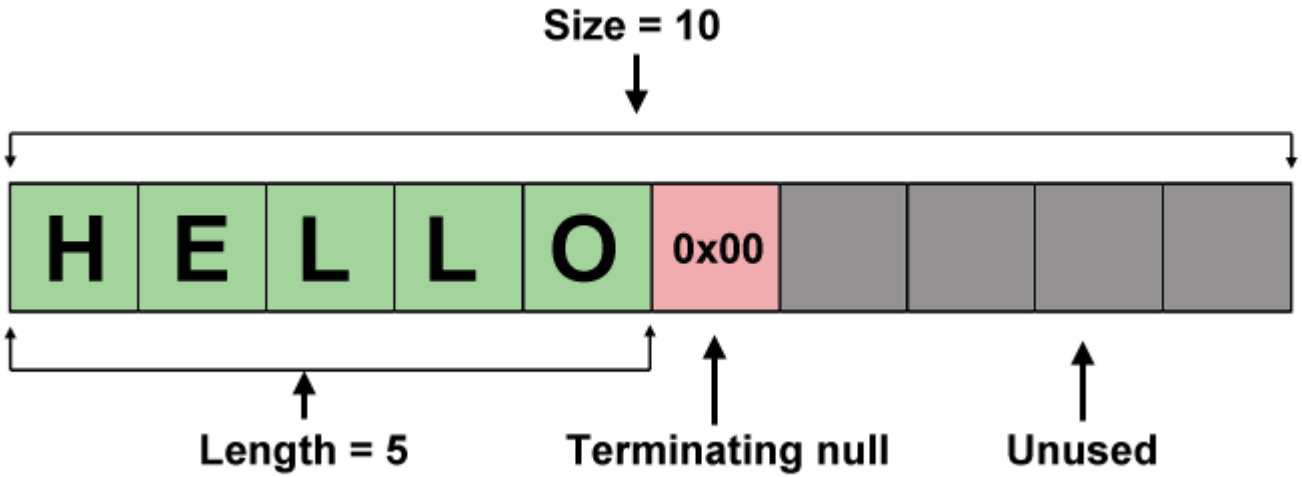
План лекции:

- Строки. Множества.
- Структуры.
- Объединения.
- Перечисление.

Строковые данные

Базовый тип данных «строка» в C/C++ отсутствует.

С точки зрения компилятора и стандартной библиотеки, строка — это массив элементов типа **char**, последним элементом которого является символ '**\0**' (символ, код которого равен нулю).



СТРОКИ

Можно обращаться к отдельным символам строки, написав после её имени в квадратных скобках номер символа. Нумерация символов в строке начинается с нуля, так же как и в векторах. Когда мы выводим переменную типа char, то выводится символ. Хотя на самом деле char – числовая переменная и обозначает номер символа в кодовой таблице. По аналогии с переводом вещественных чисел в целые мы можем сделать перевод из типа char в тип int, чтобы узнать код символа. Вывод кода символа выглядит так:

```
char c = ‘A’;
cout << (int) c;
```

Строки

Для управления наборами объектов в стандартной библиотеке C++ определены **контейнеры.** Контейнер представляет коллекцию объектов определенного типа. Последовательный же контейнер (sequential container) позволяет контролировать порядок, в котором элементы располагаются в коллекции, и управлять доступом к этим элементам.

Типы последовательных контейнеров:

- **vector**: массив переменного размера (поддерживает произвольный доступ к любому элементу в контейнере, обеспечивает добавление и удаление элементов из любого места контейнера)
- **string**: представляет контейнер, аналогичный вектору, который состоит из символов, то есть строку

С другими типами контейнеров познакомимся позже.

Для использования определенного контейнера в программу необходимо добавить соответствующий заголовочный файл, который, как правило, называется по имени класса контейнера.

Для хранения строк в C++ применяется тип **string**. Для использования этого типа его необходимо подключить в код с помощью директивы **#include <string>**

Пример: Пользователь вводит свое имя, а программа здоровается с ним. Полное решение будет записано так:

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int main()
6   {
7       string s;
8       cin >> s;
9       cout << "Hello, " + s;
10      return 0;
11  }
```

Строки

Если при определении переменной типа string мы не присваиваем ей никакого значения, то по умолчанию данная переменная содержит пустую строку

string s;

Можно инициализировать переменную строчным ***литералом***, который заключается в двойные кавычки:

string s2 = “hello “;

string s3 = (“hello “);

string s4 = (5, ‘c’); // пять одинаковых символов ‘c’

Операции над строками символов

Конкатенация строк

```
string s1 = "hello";
string s2 = "world";
string s3 = s1 + " " + s2; // hello world
cout << s3 << endl;
```

Сравнение строк

```
bool  result = s1 == s2;     // false
bool result2 = s1 > s2;      // false
```

Размер строки - методы size() и empty()

```
cout << s1.size() << std::endl;        //  5
string s4 = "";
if(s4.empty())
   cout << "String is empty" << std::endl;
```

Чтение строки с консоли

Слова в строке могут быть разделены любым количеством пробелов, табуляций и переводом строк. Но ***при использовании cin чтение идет только до разделительного символа.*** Т.е. если слова в строке разделены пробелами, то прочитается только первое слово! Все остальные символы остаются внутри cin, ожидая следующего извлечения.

Для чтения всей строки используется специальная функция **getline(cin, s).** Первый параметр в этой функции указывает на поток ввода (cin), а второй – на строку, в которую нужно считывать.

Или такой вид:

```
1 cin.getline(string, streamsize, separator);
```

Функция **getline** извлекает данные из входного потока ***до строкового разделителя (обычно \n), который не записывается в строку***.

Пример

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int main()
6   {
7       setlocale(LC_ALL, "rus");
8       string s;
9       getline(cin, s);
10      cout << s << endl;
11      return 0;
12  }
```

```
> ./main
мама мыла раму
мама мыла раму
>
```

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    setlocale(LC_ALL, "rus");
    int age;
    string s;
    cin >> age;
    getline(cin, s);
    cout << "Hello, " << s <<" your age=" << age << endl;
    return 0;
}
```

```
18
Hello,  your age=18
>
```

Когда вы запустите эту программу и введете возраст, она не будет ожидать ввода строки с именем, а сразу выведет результат (просто пробел вместо вашего имени)! Оказывается, когда вы вводите числовое значение, поток cin захватывает вместе с вашим числом и символ новой строки. Так что, когда мы ввели 18, cin фактически получил 18\n. Затем он извлёк значение 18 в переменную, оставляя \n (символ новой строки) во входном потоке. Затем, когда getline() извлекает данные для myName, он видит в потоке \n и думает, что мы, должно быть, ввели просто пустую строку! ***Правило: При вводе числовых значений не забывайте удалять символ новой строки из входного потока данных***

Удаление символа новой строки ‘\n’ из входного потока

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4   int main() {
5   setlocale (LC_ALL, "rus");
6   string s;
7   int age;
8   cin >> age;
9   getline(cin,s);
10  getline(cin,s);
11  cout << "Hello, "<< s<< " your
    age= " << age <<endl;
12  return 0;
13  }
```

Количество цифровых символов в строке

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4   int main(){
5       setlocale( LC_ALL,"Russian" );
6       string str;
7       cout<<"Введите строку\n";
8       getline(cin,str);
9       cout<<str<< endl;
10      for (auto c : str) {
11        if (c >= '0' && c <= '9') {
12         cout << c;
13        }
14      } return 0;
15  }
```

Получение и изменение символов строки

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4   int main(){
5       setlocale( LC_ALL,"Russian" );
6       string str = "hello";
7       cout<<str<< endl;
8       char c = str[1];  //e
9       str[0] = 'm';    // mello
10      cout << str << endl;
11      return 0;
12  }
```

Длина строки

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4   int main() {
5   setlocale (LC_ALL, "rus");
6   string str;
7   int dl;
8   cout << "Введите строку\n";
9   getline(cin,str);
10  cout << "длина строки = "<< str.size()<<endl;
11  //cout<<"длина строки = "<< str.length()<<endl;
12  return 0;
13  }
```

Поиск подстроки в строке

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
setlocale (LC_ALL, "rus");
int n;
cin >> n;
string s;
getline(cin, s);
for (int i = 0; i < n; i++) {
  getline(cin, s);
   if (s.find("рам") != -1) {
     for (auto c : s) {
       if (c >= '0' && c <= '9') {
           cout << c;
        }
     }
     cout << '\n';
   }
}
  return 0;
}
```

Если подстрока нашлась, то find() возвращает число, равное номеру символа, с которого началось первое вхождение подстроки в строку. А если подстроки не нашлось, то этот метод возвращает -1.

```
> ./main
3
петя рисует т34
мама 11.05 мыла раму
1105
вася купил рамку 13x18
1318
>
```

Символьные массивы

Массив символов, последний элемент которого представляет нулевой символ '\0', может использоваться как строка:

```
1   #include <iostream>
2   using namespace std;
3   int main(){
4       char m_str[] = {'h','e','l','l','o','\0'};
5       cout << m_str << endl;
6       return 0;
7   }
```

Однако подобное использование массива строк унаследовано от языка Си, а при написании программ на C++ при работе со строками следует отдавать предпочтение встроенному типу string, а не массиву символов.

```
> ./main
hello
```

Максимальная длина слова

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4   int main()
5   {
6   string s1;
7   int i=0, dl, max_dl=0;
8   int start=0;
9   getline(cin,s1);//считывает до Enter + \0
10  cout << s1 << endl;
11  while (s1[i] != '\0')//ЦИКЛ ПО ВСЕЙ СТРОКЕ
12    {
13      if (s1[i] != ' '){
14  //----- ВЫДЕЛЯЕМ СЛОВО---------
15        start = i;
16        while ((s1[i] != ' ') && (s1[i] != '\0')) i++;
17        dl = i-start;
18        if (dl > max_dl)  max_dl = dl;
19       }
20       i++;
21    }
22  cout << "max_length= " << max_dl<< endl;
23  return 0;
24  }
```

Количество слов начинающихся и оканчивающихся на

одну и ту же букву

(обработка слов в

стиле Си)

```
1   #include <iostream>
2   #include <cstring>
3   #include <string>
4   using namespace std;
5   int main(){
6       setlocale( LC_ALL,"Russian" );
7       char str[100];
8       cout<<"Введите строку\n";
9       cin.getline(str, 100);
10      cout<<str<<"\n";
11      char * razd = strtok( str, " ");
12      int last=0;
13      int kol_1=0;
14      while( razd != NULL){
15          last=strlen(razd);
16          if(razd[0]==razd[last-1])
17              kol_1++;
18          razd = strtok ( NULL, " ");}
19      cout<<"Количество слов:"<< kol_1;
20      return 0;
21  }
```

08:01

Множества

Множества — это математические структуры, которые могут хранить в себе уникальные элементы (то есть, каждый элемент может входить в множество только один раз). Для работы с множествами нужно подключить одноимённую библиотеку

#include <set>

Множества создаются по аналогии с векторами. Пишем ключевое слово set, за ним — название типа каждого из элементов множества (в треугольных скобках), а после этого указываем имя для нового множества. Так мы получаем пустое множество. Добавление элементов в него происходит с помощью метода insert. Чтобы проверить, входит ли элемент во множество, используется метод find. Если элемент в множестве не нашелся, то он выдает то же значение, что и метод end. Удаление отдельного элемента из множества выполняется с помощью метода erase.

Пример: даны N запросов трёх типов:
добавить элемент во множество; проверить, входит ли элемент во множество;
удалить элемент из множества.

```cpp
#include <iostream>
#include <set>
using namespace std;
int main() {
  set <int> s;
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    int type, x;
    cin >> type >> x;
    if (type == 1) { // добавление
      s.insert(x);
    } else if (type == 2) { // проверка
      if (s.find(x) == s.end()) {
      cout << "NO\n";
      } else {
      cout << "YES\n";}
        } else { // удаление
          s.erase(x);}
  }
return 0;
}
```

```
> ./main
3
1 6
1 5
2 6
YES
>
```

```
> ./main
3
1 6
1 5
2 4
NO
>
```

```cpp
#include <iostream>
#include <set>
using namespace std;
int main() {
  set <int> s;
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    int type, x;
    cin >> type >> x;
    if (type == 1) { // добавление
      s.insert(x);
    } else if (type == 2) { // проверка
      if (s.find(x) == s.end()) {
      cout << "NO\n";
      } else {
      cout << "YES\n";}
        } else { // удаление
          s.erase(x);}
  }
  for (auto now : s) {
    cout << now << ' ';}
return 0;
}
```

```
> ./main
3
1 6
1 5
1 6
5 6 >
```

Или вывод так:

```cpp
#include <iostream>
#include <set>
using namespace std;
int main() {
  set <int> s;
  int n;
  cin >> n;
  for (int i = 0; i < n; i++) {
    int type, x;
    cin >> type >> x;
    if (type == 1) { // добавление
      s.insert(x);
    } else if (type == 2) { // проверка
      if (s.find(x) == s.end()) {
      cout << "NO\n";
      } else {
      cout << "YES\n";}
          } else { // удаление
            s.erase(x);}
  }
  for (auto now = s.begin(); now != s.end(); now++) {
    cout << *now << ' ';}
return 0;
}
```

В данном случае now — это не очередной элемент, а указатель на него. Метод begin возвращает указатель на самый маленький элемент множества, end — это конец множества (он идёт после самого большого элемента), а операция ++ осуществляет переход к указателю на следующий элемент. Чтобы посмотреть, что за элемент хранится по указателю, нужно перед его именем написать символ *.

Структуры

- ***Структура*** – это именованная совокупность переменных возможно разных типов, расположенная в памяти последовательно друг за другом.
- Структуры называются пользовательскими типами данных и помогают в организации сложных данных, поскольку позволяют группу связанных между собой переменных трактовать не как множество отдельных элементов, а как единое целое.
- Структуры могут копироваться, над ними могут выполняться операции присваивания, их можно передавать функциям в качестве аргументов, а функции могут возвращать их в качестве результата.
- Для структур допускается инициализация.

Структуры

Объявление структуры начинается с ключевого слова **struct** и содержит список объявлений, заключенный в фигурные скобки:

```
struct имя_структуры {
        список объявлений;
};
```

Элементами структур могут быть:

- переменные и массивы базовых типов,
- переменные и массивы пользовательских типов, кроме типа самой структуры *имя_структуры*,
- указатели на любые типы, включая и тип самой структуры *имя_структуры*,
- функции.

Структуры

Пример структуры **time**:

```
struct time {
  int hour;
  int minutes;
};
struct time t {20,30};
```

В приведенном примере элементами структуры будут *hour* и *minutes*.

Объявление структуры ***не резервирует памяти***. Оно является информацией компилятору о введении пользовательского типа данных. Память выделится при определении структурных переменных.

Доступ к отдельному элементу структуры осуществляется посредством бинарной операции «точка».

```
t.hour = 21;
cout << t.hour;
```

Структуры

Структуры могут быть вложены друг в друга. Например, структура *chronos* содержит две структуры *time* – *begin* и *end*:

```
struct chronos {
        struct time begin, end;
};
struct chronos timer = {{2,4}, {10, 10}};
```

Выражение ***timer.begin.minutes*** обращается к минутам ***minutes*** времени ***begin*** из ***timer***.

Ключевое слово struct при объявлении структурных переменных можно опускать, то есть допустима и общепринята запись

```
chronos timer;
```

Структуры

Над структурами возможны следующие операции:

- присваивание,
- взятие адреса с помощью &,
- осуществление доступа к ее элементам.

Существует, по крайней мере, три подхода передачи структуры в функцию: передавать компоненты по отдельности, передавать всю структуру целиком и передавать указатель на структуру. Каждый подход имеет свои плюсы и минусы.

Указатели на структуры

Так как имя структурного типа обладает всеми правами имен типов, то разрешено определять указатели на структуры:

имя_структурного_типа * имя_указателя_на_структуру;

Если функции передается большая структура, то, чем копировать ее целиком, эффективнее передать указатель на нее. Указатели на структуры ничем не отличаются от указателей на обычные переменные.

Объявление

time *pt;

сообщает, что *pt* - это указатель на структуру типа struct time. Если pt указывает на структуру ***time***, то ****pt*** - это сама структура, а ***(*pt).hour*** и ***(*pt).minutes*** - ее элементы. Скобки в ***(*pt).hour*** необходимы, поскольку приоритет операции . (точка) выше, чем приоритет операции разыменования * (звездочка).

Указатели на структуры

Указатели на структуры используются весьма часто, поэтому для доступа к ее элементам была придумана операция **«обращение к элементу структуры по указателю».** Если t — указатель на структуру, то ***t -> элемент-структуры*** есть ее отдельный элемент.

Операции . и -> выполняются слева направо.

Таким образом, при наличии объявления

следующие четыре выражения будут эквивалентны:

```
chronos ch, *cht = &ch;
```

```
ch.begin.hour
cht->begin.hour
(ch.begin).hour
(cht->begin).hour
```

Указатели на структуры

Операции доступа к элементам структуры . и -> вместе с операторами вызова функции () и индексации массива [] занимают самое высокое положение в иерархии приоритетов и выполняются раньше любых других операторов.

Например, если задано объявление

```
struct {
int len;
char *str;
} *p;
```

то ***++p->len*** увеличит на 1 значение элемента структуры *len*, а не указатель p, поскольку в этом выражении как бы неявно присутствуют скобки: *++(p->len)*.

Указатели на структуры

- Чтобы изменить порядок выполнения операций, нужны явные скобки. Так, в ***(++p)->len***, прежде чем взять значение *len*, программа прирастит указатель *p*.
- В ***(p++)->len*** указатель *p* увеличится после того, как будет взято значение *len* (в данном случае скобки не обязательны).
- По тем же правилам ****p->str*** обозначает содержимое объекта, на который указывает *str*; ****p->str++*** увеличит указатель *str* после получения значения объекта, на который он указывал, ***(*p->str)++*** увеличит значение объекта, на который указывает *str*; ****p++->str*** увеличит p после получения того, на что указывает *str*.

Указатели на структуры

Часто возникает проблема, при объявлении структуры объявить там указатель на эту же структуру.

Несмотря на то, что описание структуры еще не завершено, формат языка это позволяет.

Например, опишем структуру *List*, представляющую собой однонаправленный список (будет рассмотрено позже), где в качестве данных опишем переменную типа *int*.

```
struct List{
   int dat;
   List* next;
};
```

Массивы структур

Возможно определение массива структур

```
struct key {
    char *word;
    int count;
};
key keytab[NKEYS];
```

объявляет структуру типа *key* и определяет массив *keytab*, каждый элемент которого является структурой этого типа и которому где-то будет выделена память.

```
struct key {
    char *word;
    int count;
} keytab[NKEYS];
```

Объединения

Объединение подобно структуре, однако в каждый момент времени может использоваться только один из элементов объединения. Тип объединения может задаваться в следующем виде:

union {описание элемента 1; ... описание элемента n;};

Главной особенностью объединения является то, что ***для каждого из объявленных элементов выделяется одна и та же область памяти***, достаточная для размещения наиболее длинного элемента, т.е. они перекрываются.

Когда используется элемент меньшей длины, то переменная типа объединения может содержать неиспользуемую память.

Объединения

Пример:

```
union {
          char   name[30];
          char   addres[80];
          int    age;
          int    phone;
} inform;
```

При использовании объекта *inform* типа *union* можно обрабатывать только тот элемент, который получил значение, т.е. после присвоения значения элементу *inform.name*, не имеет смысла обращаться к другим элементам.

Объединения

```
union {
        int ax;
        char al[4];
} ua;
```

Объединение **ua** позволяет получить отдельный доступ к каждому байту числа ***ua.ax***, который занимает в памяти 4 байта.

Перечисления

Перечисления (enum) представляют еще один способ определения своих типов. Их отличительной особенностью является то, что они содержат набор числовых констант.

Определим простейшее перечисление:

```
1  enum seasons
2  {
3      spring,
4      summer,
5      autumn,
6      winter
7  };
```

Для определения перечисления применяется ключевое слово **enum**, после которого идет название перечисления. Затем в фигурных скобках идет перечисление констант через запятую. Каждой константе по умолчанию будет присваиваться числовое значение начиная с нуля. То есть в данном случае spring=0, а winter=3.

Перечисления могут использоваться, когда у нас есть ряд логически связанных констант, которые естественно лучше определить в одном общем типе данных.

Если нас не устраивают значения по умолчанию для констант, то мы можем явным образом задать значения. Например, установить начальное значение:

```
enum seasons
{
    spring = 1,
    summer, //2
    autumn, //3
    winter  //4
};
```

Также можно задать значение для каждой константы:

```
enum seasons
{
    spring = 1,
    summer = 2,
    autumn = 4,
    winter = 8
};
```

Перечисление

Переменная этого типа может принимать значение из некоторого списка значений

```cpp
#include <iostream>
using namespace std;
enum seasons // declare union
{
    spring,
    summer,
    autumn,
    winter
};
int main() {
  seasons t = autumn;
  cout << t << endl;

  return 0;
}
```

```
2
```

```cpp
#include <iostream>
using namespace std;
enum seasons // declare union type
{
    spring =1,
    summer =2,
    autumn =4,
    winter =8
};
int main() {
  seasons t = autumn;
  cout << t << endl;

  return 0;
}
```

```
4
```

Перечисление

```cpp
#include <iostream>
using namespace std;
enum seasons // declare union ty
{
    spring = 1,
    summer = 2,
    autumn = 4,
    winter = 8
};
int main() {
  int x;
  //int vv;
  cin >> x;
  switch (x){
    case seasons::spring:
    cout << "spring" << endl;
    break;
    case seasons::summer:
    cout << "summer" << endl;
    break;
    case seasons::autumn:
    cout << "autumn" << endl;
    break;
    case seasons::winter:
    cout << "winter" << endl;
    break;
    default: break;
  }
  return 0;
}
```

Объявление типа

Любой тип может быть объявлен с использованием ключевого слова *typedef*, включая типы указателя, функции или массива.

Имя с ключевым словом *typedef* для типов указателя, структуры, объединения может быть объявлено, прежде чем эти типы будут определенны, но в пределах видимости объявителя.

```
typedef char FIO[40] // FIO - массив из сорока символов
FIO person; // Переменная person  -  массив из сорока символов
//  Это  эквивалентно объявлению
char person[40];
```

При объявлении переменной и типа здесь было использовано имя типа (FIO). Помимо этого, имена типов могут еще использоваться в трех случаях: в списке формальных параметров, в объявлении функций, в операциях приведения типов и в операции sizeof (операция приведения типа).