Языки программирования

Лекция 5

ПМИ 2 курс

Демяненко Я.М.

ЮФУ 2025

Строки

- в стиле C
- в стиле C++

Строки в стиле С

В языке С строки определяются как одномерный массив типа char.
char str[] = "Hello world!!!";

Это строка из 14 символов, однако, размер массива будет равен 15, так как строки в С заканчиваются символом \0 — нуль терминатор (нулевой байт: байт, все биты которого равны 0). Иногда такие строки называют нуль-терминированными.

Недостаток такого подхода состоит в том, что для того чтобы узнать длину строки необходимо просканировать всю строку до конца, в поисках \0 — это может занять много времени.

Следовательно, и основные принципы работы с ними такие же, как и с массивами.

Описание и инициализация строк в стиле С

```
char s1[20] = "String";

char s2[6] = "String"; // Не отведено место под завершающий '\0’
//Если не отведено место под завершающий null-символ, то конец строки не определен
```

Чтобы избежать этой ошибки, рекомендуется не указывать размер массива при объявлении строки:

```
char s3[] = "String";

char s3_1[] = {'S', 't', 'r', 'i', 'n', 'g', '\0’};

char* s4 = "String"; //Не корректно, а в новых стандартах — не разрешает

const char* s4 = "String"; //Можно
```

Если в случае s3 объявляется константный указатель на массив, то объявленный указатель s4 ссылается на константную строку.

```
char s3[] = "String";
const char* s4 = "String";

//значение типа "const char *" нельзя использовать для инициализации сущности типа "char *"
// в последнем стандарте

s3[1]='A';
cout<<s3<<endl;

s4[1]='A'; – ошибка времени выполнения в старом стандарте
cout<<s4<<endl;
cout<<*(++s3)<<endl; – ошибка компиляции
cout<<*(++s4)<<endl;

В новом стандарте
const char * s4 = "String";
При таком объявлении оператор
s4[1]='A';
будет вызывать ошибку компиляции
's4': you cannot assign to a variable that is const
```

Создать строку, так же как и массив, можно динамически.
char* s5 = new char[10];

где s5 — указатель на строку, память для которой выделена динамически.
При таком описании строка не инициализирована, а значит не содержит символа конца строки.

Распространенная ошибка при попытке описания строки:
char* s6;//это не строка, а указатель на строку

В действительности переменная s6 является указателем на char или на строку (массив символов).

Ввод-вывод

```
char* str = new char[14];

cin>> str;
```

Распространённые ошибки при вводе строковых данных

Ошибка 1. Попытка ввести больше символов, чем вмещает в себя массив символов.

Ошибка 2. Попытка ввести строку, содержащую пробельные символы.

Ошибка 1. Попытка ввести больше символов, чем вмещает в себя массив символов

```
char word[10];
cin >> word;
```

При вводе строки "□□abracadabra□□" (пробел для наглядности изображен символом □) два начальных пробела будут пропущены, в массив word будут записаны символы "abracadabr", а символы 'a' и '\0' будут записаны в следующие за word ячейки памяти. Сообщение об ошибке при этом, как правило, не возникает.

Решение.

```
cin>>setw(10)>>word;          //iomanip
```

В этом случае в массив word попадут символы "abracadab"плюс завершающий нулевой символ, а символы 'ra' останутся в потоке ввода.

Ошибка 2. Попытка ввести строку, содержащую пробельные символы

```
char* str = new char[14];

cin>> str;
```

При попытке стандартным образом ввести с консоли значение "Hello world!" в str, будет введено только "Hello" (до первого пробела).

Решение.

Чтобы этого не происходило можно использовать следующую запись:

```
cin.getline(str, 13);       // 12 — количество вводимых символов + '\0’

cin.getline(str, 13,'!’);  // или до '!' но не больше 12
```

Как передавать в функции С-строки

С-строки

```
void p(char s[])          // s передается по ссылке
void p(const char s[])    // const запрещает изменение s
void p(char * s)          // s передается по ссылке
void p(const char *s)     // const запрещает изменение s
```

Обработка строк в стиле языка С

Стандартные функции для работы с С-строками

```
#include <cstring>

size_t strlen(const char* p);
// Возвращает длину строки p

char* strcpy(char* destination, const char* source);
// Копирует строку source в строку destination
// Возвращает указатель на первый символ скопированной строки

int strcmp(const char* s1, const char* s2);
// Сравнение строк в лексикографическом порядке
// < 0, s1 < s2
// = 0, s1 == s2
// > 0, s1 > s2
```

Стандартные функции для работы с С-строками

```
#include <cstring>


char* strcat (char* destination, const char* source);
// Добавляет source в конец destination
// Считается, что в строке destination достаточно памяти

char* strchr(const char* s, char c);
// Ищет в строке s символ c

char* strstr(const char* s, const char* s1);
// Ищет в строке s подстроку s1
```

_CRT_SECURE_NO_WARNINGS

Часто жалуются на «неработающие» коды,
особенно консольных приложений или CLR,
особенно тех, что работали без каких-либо замечаний в версиях 2010-2013
и вдруг «не работают» в 2015 и выше

Выдаются ошибки типа
Ошибка C4996 '**strcpy**': This function or variable may be unsafe.
Consider using **strcpy_s** instead. To disable deprecation, use
_CRT_SECURE_NO_WARNINGS. See online help for details.

_CRT_SECURE_NO_WARNINGS
для совместимости с классическими функциями

Можно заменить,
а можно оставить (НО тогда вы несете ответственность за безопасность).

Если оставить, то

ИЛИ

добавьте в самом начале до всех #include
#define _CRT_SECURE_NO_WARNINGS

ИЛИ

Настроить Управление предкомпилированными заголовками: меню Проект - Свойства C/C++ - Препроцессор (Preprocessor) - Определения препроцессора (Preprocessor Definitions).

Проверено на Visual Studio 2015 и более поздних

Контроль памяти при работе с этими функциями

char s1[] = "Hello";
char* s2;

strcpy(s2, s1); // Ошибка времени выполнения, потому что память под s2 не выделена

char s2[4];
strcpy(s2, s1); // Контроль памяти лежит на программисте.

// В случае возникновения ошибки она, поначалу, может себя никак не проявить.

strcpy_s(s2, s1);
//Или не работает вовсе – новый стандарт
//Run-Time Check Failure #2 - Stack around the variable 's2' was corrupted.
//или Отсутствуют экземпляры перегруженная функция "strncpy_s", соответствующие списку аргументов

Контроль памяти при работе с этими функциями

char s1[] = "Hello";
char s2[4];
strncpy(s2, s1, 4);
//в старом варианте Результат cout<<s2; HeII╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠Hello
char s2[4];
char s1[] = "Hello";
strncpy(s2, s1, 4);
// в старом варианте Результат cout<<s2 HeII╠╠╠╠╠╠╠╠(▌ ╬

//в новом в обоих случаях ошибка компиляции - This function or variable may be unsafe.
// Consider using //strncpy_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS.

strncpy_s(s2, s1, 4); //ошибка времени выполнения

Контроль памяти при работе с этими функциями

char s3[10];
strncpy(s3, s1, 4);
cout<<s3; //Результат Hell╠╠╠╠╠╠╠╠╠╠╠╠╠╠Hello

strncpy_s(s3, s1, 4);
cout<<s3; //Результат Hell

Реализация аналогичных функций

Копирование строки

подход по принципу массива

```
int i;
for (i=0; s1[i]!=0; ++i)     //for (i=0; s1[i]; ++i)
  s2[i] = s1[i];
s2[i] = 0;
```

++i vs i++

используя указатели

```
char *p=s1, *q=s2;
while (*p)                   //   (*p!=0)
  *q++=*p++;                 //*q++=*p++;
*q=0;                        равносильно
                             *q = *p;
                              q++;
                              p++;
```

или

```
while (*q++ = *p++);
```

Функция копирования

```
char * copy (char *p, const char *q) {
  char *res=p;
  while (*p++=*q++);
  return res;
}

char s1[20], s2[10]="Hello!";
char *s3=new char [21];
char *s=copy(s1,s2);
copy(s3,s1);
```

Рекурсивные функции

```
void printReverse (const char* s) {
  if (*s !=0 ){
    printReverse (s+1);
    cout <<*s;
  }
}
```

Ошибки

- строка не размещена в памяти

```
char s1[10]="Hello!";
char *s3;
copy(s3,s1);
```

- потерян завершающий строку символ

```
char a[2]= "ab";
char s[5]= "123";
```

- аналогичная ошибка при работе через индекс

```
char s []= "abc";
s[3]='s';
```

- попытка скопировать строку через операцию присваивания

```
char s [ ]= "fgfgf";
char  ns [10];
char  *p;
ns = s; //статические массивы нельзя присваивать
p  = s;
```

Пример

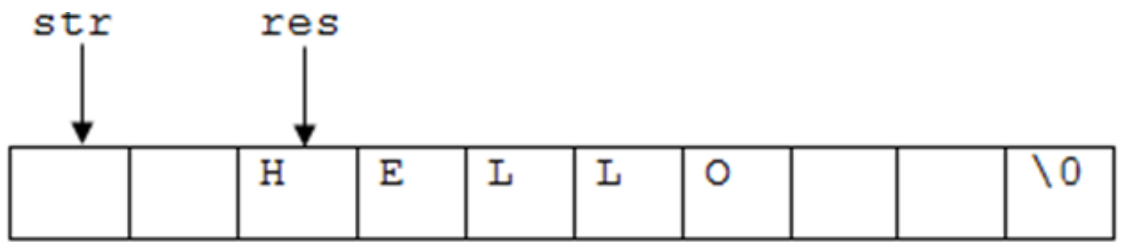
Описать функции TrimLeftC(s,c) и TrimRightC(s,c), удаляющие в строке s соответственно начальные и концевые символы, совпадающие с символом c.

Поскольку очень часто требуется удалить начальные пробелы, то параметр c сделать параметром по умолчанию (по умолчанию значение равно символу пробела).

```
char* TrimLeftC(char* s, char c=' ') {
   while (*s!=0 && *s==c)
   s++;
  return s;
}
```

```
void TrimRightC(char* s, char c=' ') {
  char *p=s;
  while (*p!=0)
    p++;
  if (p==s) return;
  --p;
  while (p!=s && *p==c)
    p--;
  p++;
  *p=0;
  return;
}
```

```
char *str=new char[101];
char *res;
cout<<endl<<"input string ";
cin.getline(str,100);
res=TrimLeftC(str);
cout<<endl<<"origin string ="<<str<<endl;
cout<<"result string ="<<res<<endl;
```



Так как исходная строка размещается в динамической памяти, то в какой-то момент память необходимо освободить:

```
delete[] str;
```

Функция TrimLeftCInPlace реализует алгоритм удаления ведущих символов путём сдвига влево.
Этот вариант работает дольше, но решение полностью соответствует постановке задачи.

```
void TrimLeftCInPlace(char* S, char C = ' '){
  char *p = S;
  while (*p != 0 && *p == C)
    p++;
  while (*S++ = *p++);
}
```

Удаление ведущих пробелов в строковой константе

```
int main() {
  char *s;

  //удаление ведущих пробелов в строковой константе
  s=TrimLeftC("   Hello       !");
  cout<<endl<<"string3="<<s<<endl;

  return 0;
}
```

```
char* TrimLeftC(const char* S, char C=' ') {
  char *q=const_cast<char*>(S);
  while (*q!=0 && *q==C)
    q++;
  return q;
}
```

Приведение типов (снятие константности)

```
const char *q;
char *p;

p=(char*)q;                 //устаревшая форма

p=const_cast <char*>(q);        //рекомендуемая форма
```

Пример

Найти индекс последнего вхождения символа ch в строке s, используя функцию strchr

```
int last_ind(char *s, char ch){
  int ind = -1;
  char *p = strchr(s,ch);
  while (p) {
    ind = p - s;
    p = strchr(p+1, ch);
  }
  return ind;
}
```

Пример

Вычислить количество вхождений строки s2 в строку s1, используя стандартные функции strstr, strlen библиотеки **cstring**

```
int count(char *s1,char *s2){
  int c = 0;
  int len = strlen(s2);
  char *p = strstr(s1, s2);
  while (p){
      c++;
      p = strstr(p + len, s2);
  }
  return c;
}
```