НРТК

C / C++

Тема 07. Строковый тип данных

Строки в C / C++

- Специального типа данных «строка» нет (в смысле набора значений и операций)
- Строки объявляются как массивы типа **`char`**

```
char str[15];
```

- Операции над строками обеспечиваются специальной библиотекой

Строки в C / C++

- Можно присваивать значение при инициализации:

```
char str[15] = "Вова+Лена";
```

- Массив при этом заполняется слева-направо, последним заносится нулевой байт (признак конца), далее — нули:

Строки в C / C++: ввод

- C++: потоковый ввод-вывод

```
cin >> str;
```

- C: форматный ввод-вывод (`stdio.h`)

```
scanf("%s", str);
```

- Вводят значения, разделенные пробелами (т. е. нельзя ввести строку, содержащую пробелы)
- Не контролируют длину строки
  - проблема переполнения буфера

Строки в C / C++: ввод

- C++: потоковый ввод-вывод:

```
cin.getline(char s[], size_t count);
```

  – пример:

```
const int str_len = 20 + 1; // +1 - под \0
char str[str_len];
cin.getline(str, str_len);
```

- C: форматный ввод-вывод (`stdio.h`)

```
char* gets(char s[]);
```

  – пример:

```
const int str_len = 20 + 1;
char str[str_len];
char* s = gets(st);
gets(st);
```

- В C остается возможность переполнить буфер

Строки в C / C++: вывод

- C++: потоковый ввод-вывод:

```
cout << str;
```

- C: форматный ввод-вывод (`stdio.h`)
  - вывод по-умолчанию

```
printf("%s", str);
```

  - вывод *n* символов строки

```
printf("%ns", str);
```

  - вывод без формата (никогда так не делайте!)

```
printf(str);
```

Пример: ввод текста

```
const MAXT = 100;
char txt[MAXT][80];
int nt = 0;
do
{
   cin.getline(txt[nt++], 80);
} while ( nt <= MAXT && txt[nt - 1] != “” );
```

Библиотека `cstring` (`string.h`)

Подробнее

- Определение длины строки `str`:

```
size_t strlen(const char str[]);
```

  – пример:

```
size_t l = strlen("Вова = ?");// l = 8
```

- Сравнение строк `s` и `d`:

```
int strcmp(const char s[], const char d[]);
```

  – пример:

```
r1 = strcmp("Волк", "Воля");  // r1 < 0
r2 = strcmp("Волк", "Валя");  // r2 > 0
r3 = strcmp("Волк", "Волк");  // r3 = 0
```

Библиотека cstring (string.h)

Подробнее

- Копирование строки s в d:

```
char* strcpy(char d[], const char s[]);
```

  – пример:

```
const int str_len = 20 + 1; // +1 - под \0
char str[str_len];
strcpy(str, "Лена");        // В str — «Лена»
```

  – проблема — переполнение буфера

- Копирование не более n символов строки s в d:

```
char* strncpy(char d[], const char s[],
size_t n);
```

  – пример:

```
strncpy(str, "Лена", str_len);
```

  – если s длиннее d, то функция «забывает» финальный 0

Библиотека `cstring` (`string.h`)

Подробнее

- Объединение (конкатенация) строк `s` и `d`:

```
char* strcat(char d[], const char s[]);
```

  – пример:

```
char str[str_len] = "Вова+";
strcat(str, "Лена");     // В s1 – «Вова+Лена»
```

  – проблема — переполнение буфера

- Объединение строки `s` и не более, чем `n` символов строки `d`:

```
char* strncat(char d[], char s[], size_t n);
```

  – пример:

```
char str[str_len + 1] = "Вова+";
strncat(str, "Лена", str_len); // «Вова+Лена»
```

  – проблема — переполнение буфера

Пример: StrLen

- Определение длины строки:

```
size_t StrLen(const char str[])
{
  size_t len = 0;
  while ( str[len++] ) ;
  return len - 1;
}
```

Пример: StrCopy

- Копирование символов строки `s` в `d`, с контролем длины строки `d` (не более `n` символов, включая 0):

```
bool StrCopy(char d[], size_t n,
  const char s[])
{
  // Количество скопированных символов
  size_t cnt = 0;
  while ( (cnt < n - 1) && (d[cnt] = s[cnt]) )
    cnt++;
  // В любом случае - запись 0
  d[n - 1] = '\0';
  return (cnt == StrLen(s));
}
```

Пример: StrCat

- Объединение строк `s` и `d`, с контролем длины строки `d` (не более `n` символов, включая 0):

```
bool StrCat(char d[], size_t n,
  const char s[])
{
  // Количество скопированных символов
  size_t cnt = 0;
  size_t len = StrLen(d);
  while ( (len + cnt < n - 1) &&
    (d[len + cnt] = s[cnt]) ) cnt++;
  // В любом случае - запись 0
  d[n - 1] = '\0';
  return (cnt == StrLen(s));
}
```

Выводы

- Программы с использованием строк в стиле C подвержены ошибкам, в C++ есть более безопасные средства для работы со строками
- Функции, «увеличивающие» строку обязательно должны знать максимальный размер буфера, используемого для этой строки

Пример: StrToInt

- Перевод строки str в число:

```
int StrToInt(const char str[])
{
  int num = 0;
  int i = 0;
  while(str[i] && str[i]>='0' && str[i]<='9')
    num = num * 10 + str[i++] - '0';
  return num;
}
```

- Библиотечные функции (**cstdlib**, **stdlib.h**):

```
int atoi(char str[]);
sscanf("строка", "формат", <список ввода>);
```

  - пример:

```
sscanf(str, "%i" , &i);
```

Пример: IntToStr

```
void IntToStr(char str[], int num)
{
  int len = 0, nr = num;
  // Определение длины строки
  do
    len++;
  while ( nr  /= 10 );

  str[len] = '\0';
  do
    str[--len] = '0' + num % 10;
  while ( num /= 10 );
}
```

Пример: удаление символа (версия 1)

```
int DelSymbol(char str[], char sym)
{
  int i = 0, j = 0;
  while(str[i])
    if(str[i] == sym)
      i++;
    else
      str[j++] = str[i++];

  str[j] = '\0';

  return i - j;
}
```

Пример: удаление символа (версия 2)

```
int DelSymbol(char str[], char sym)
{
  int i = 0, j = 0;
  do
    if(str[i] != sym )
      str[j++] = str[i];
  while(str[i++]);

  return i - j;
}
```

Пример: анализ палиндрома (версия 1)

```
DelSymbol(str, ' ');
size_t n = StrLen(str);
size_t i, j = n - 1;

bool res = true;
for ( i = 0; i < n; i++ )
{
  if ( str[i] != str[j] )
    {
      res = false;
      break;
    }
    else j--;
}
```

Пример: анализ палиндрома (версия 2)

```
DelSymbol(str, ' ');
size_t n = strlen(str), i, j;

bool res = true;
for (i = 0, j = n - 1; rez && i < j; i++, j--)
  if ( rez = (str[i] == str[j]) ) ;
```

Здесь есть ошибка. Где?

Демонстрация

Задачи

- Перестановка символов в строке в обратном порядке
- Определение количества вхождений подстроки в строку
- Задачи для самостоятельной работы