Тема: «Строковые данные»

Вопросы:

1. Нуль-терминальные строки. Функции для работы с нуль-терминальными строками.
2. Алгоритмы работы с нуль-терминальными строками.

Мы можем работать со строками в C++ в так называемом C-стиле как с массивами символов, которые оканчиваются на нулевой байт '0'. Однако, что если такой символ не будет найден или в процессе манипуляций со строкой будет удален, то дальшейшие действия с такой строкой могут иметь недетерминированный результат. По этой причине строки в C-стиле считаются небезопасными, и рекомендуется для хранения строк в C++ использовать тип std::string из модуля <string>.

Объект типа string содержит последовательность символов типа char, которая может быть пустой. Например, определение пустой строки:

```
std::string message;
```

Также можно инициализировать или присвоить переменной string конкретную строку:

```
std::string message {"Hello METANIT.COM!"};
// или так
std::string message2 = "Hello METANIT.COM!";
std::string message3("Hello METANIT.COM!");
```

В данном случае переменная message получит копию строкового литерала "Hello METANIT.COM!". В своем внутреннем представлении переменная message будет хранить массив символов, который также заканчивается на нулевой байт. Однако реализация типа string и предлагаемые им возможности делают работу с этим типом более безопасной.

И можно инициализировать объект string другим объектом string:

```
std::string hello{"hello world"};
std::string message {hello}; // message = "hello world"
// или так
// std::string message (hello);
// std::string message = hello;
```

Мы можем вывести подобную строку на консоль:

```
#include <iostream>
```

```
#include <string>

int main()
{
   std::string message {"Hello METANIT.COM!"};
   std::cout << "Message: " << message << std::endl; // Message: Hello METANIT.COM!
}
```

Получение и изменение символов строки

Подобно массиву мы можем обращаться с помощью индексов к отдельным символам строки, получать и изменять их:

```
std::string hello {"Hello"};
char c {hello[1]};      // e
hello[0]='M';
std::cout << hello << std::endl;    // Mello
```

Поскольку объект string представляет последовательность символов, то эту последовательность можно перебрать с помощью цикла for. Например, подсчитаем, сколько раз в строке встречается буква "l":

```
#include <iostream>
#include <string>

int main()
{
   unsigned count{};   // счетчик, сколько раз встречается символ
   std::string message{ "Hello World"};
   for(const char c: message)
   {
      if(c == 'l')
      {
         count++;
      }
   }
   std::cout << "Count: " << count << std::endl;   // Count: 3
}
```

Чтение строки с консоли

Для считывания введенной строки с консоли, как и для считывания других значений, можно использовать объект std::cin:

```
#include <iostream>
#include <string>

int main()
{
   std::string name;
   std::cout << "Input your name: ";
   std::cin >> name;
   std::cout << "Your name: " << name << std::endl;
}
```

Консольный вывод:

```
Input your name: Tom
Your name: Tom
```

Однако если при данном способе ввода строка будет содержать подстроки, разделенные пробелом, то std::cin будет использовать только первую подстроку:

```
Input your name: Tom Smith
Your name: Tom
```

Чтобы считать всю строку, применяется метод getline():

```
#include <iostream>
#include <string>

int main()
{
   std::string name;
   std::cout << "Input your name: ";
   getline(std::cin, name);
   std::cout << "Your name: " << name << std::endl;
}
```

Метод getline принимает два объекта - std::cin и переменную, в которую надо считать строку.

Консольный вывод:

```
Input your name: Tom Smith
Your name: Tom Smith
```

Терминальный ноль (terminal null) '\0' - символ конца строки в стиле C. Используется в строковых функциях (strlen, strcat), в функциях вывода (sscanf, printf). Терминальный ноль позволяет определить, где заканчивается обрабатываемая строка. Другими словами, ноль служит идентификатором конца строки. Без терминального нуля последовательность символов не сможет обрабатываться как строка, так как не будет известно, где она заканчивается.

Приведём несколько примеров использования терминального символа.

Пример первый.

```
void str_cpy(char* dst, char* src)
{
  while(*dst++ = *src++);
}
```

В этом примере в '*dst' записывается значение из '*src, до тех пор, пока не будет записан символ конца строки '\0'. Как только будет записан терминальный ноль '\0', цикл завершится.

В качестве ***второго примера*** приведём реализацию функции str_cat(char *to, const char *from), которая объединяет строку 'to' и 'from' и выводит результат в 'to'.

```
void str_cat(char *to, const char *from)
{
  while(*to++);
  to--;
  while (*to++ = *from++);
}
```

В этом примере будет установлен указатель на первый найденный терминальный символ в строке 'to'. После произведена запись символов из массива 'from' в массив 'to', до тех, пока из 'from' не будет прочитан символ терминального нуля.

В Windows API есть структуры, в которых указатели на строки должны заканчиваться парой терминальных символов. В качестве примера можно привести член lpstrFilter в структуре OPENFILENAME. lpstrFilter - указатель на буфер, в котором находятся пары нуль-терминированных (null-terminated) строк для фильтра. Первая строка пары содержит описание фильтра

(например, "Text Files"), а вторая - сам шаблон фильтра (например, "*.TXT"). Шаблоны отделяются друг от друга терминальным символом. Последняя строка в этом буфере должна заканчиваться двумя символами '\0'.

Пример строки для этого параметра из MSDN (2 шаблона):

ofn.lpstrFilter = "All\0*.*\0Text\0*.TXT\0"

Каждая подстрока заканчивается терминальным нулем. В том числе и последняя. Компилятор добавит ещё один ноль в конце всей строки. В результате в конце будет два терминальных нуля и функция с сможет понять, где заканчиваются передаваемые данные.

Нуль-терминированная строка или С-строка (от названия языка Си) или ASCIIZ-строка — способ представления строк в языках программирования, при котором вместо введения специального строкового типа используется массив символов, а концом строки считается первый встретившийся специальный нуль-символ (NUL из кода ASCII, со значением 0).

Функции для работы со строками в С++

К стандартным функциям библиотеки cstring относятся:

– strlen() – подсчитывает длину строки (количество символов без учета \0);
– strcat() – объединяет строки;
– strcpy() – копирует символы одной строки в другую;
– strcmp() – сравнивает между собой две строки.

strlen() (от слова length – длина)

Пример программы, которая подсчитает количество символов в строке:

```
#include <iostream>
using namespace std;
 int main()
{
 setlocale(LC_ALL, "rus");
  char ourStr[128] = ""; // для сохранения строки
  cout << "Введите текст латиницей (не больше 128 символов):\n";
 cin.getline(ourStr, 128);
  int amountOfSymbol = 0; // счетчик символов
 while (ourStr[amountOfSymbol] != '\0')
 {
```

```
 amountOfSymbol++;
 }
  cout << "Строка \"" << ourStr << "\" состоит из "
 << amountOfSymbol << " символов!\n\n";
  return 0;
}
```

Для подсчёта символов в строке неопределённой длины (так как вводит её пользователь), мы применили цикл while – строки 13 – 17. Он перебирает все ячейки массива (все символы строки) поочередно, начиная с нулевой. Когда на каком-то шаге цикла встретится ячейка ourStr [amountOfSymbol], которая хранит символ \0, цикл приостановит перебор символов и увеличение счётчика amountOfSymbol.

Так будет выглядеть код, с заменой нашего участка кода на функцию strlen():

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
 setlocale(LC_ALL, "rus");

 char ourStr[128] = ""; // для сохранения строки

 cout << "Введите текст латиницей (не больше 128 символов):\n";
 cin.getline(ourStr, 128);

 cout << "Строка \"" << ourStr << "\" состоит из "
 << strlen(ourStr) << " символов!\n\n";

 return 0;
}
```

Как видите, этот код короче. В нем не пришлось объявлять дополнительные переменные и использовать цикл. В выходном потоке cout мы передали в функцию строку – strlen(ourStr). Она посчитала длину этой строки и вернула в программу число. Как и в предыдущем коде-аналоге, символ \0 не включен в общее количество символов.

Результат будет и в первой программе и во второй аналогичен:

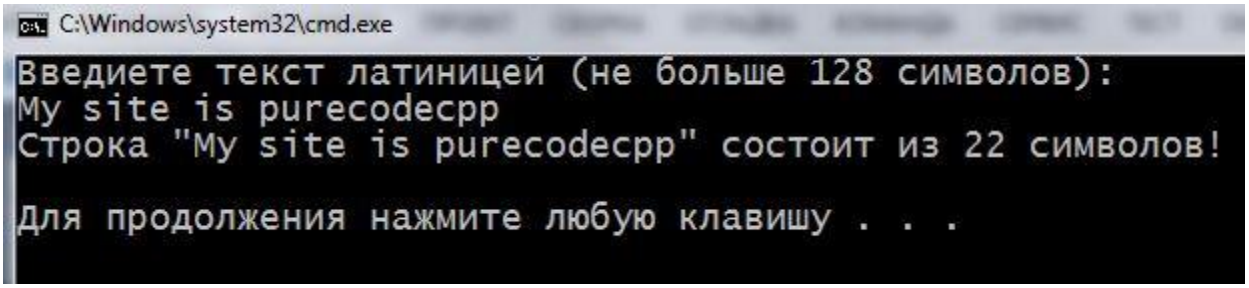

Рисунок 1

strcat() (от слова concatenation – соединение)

Программа, которая в конец одной строки, дописывает вторую строку. Другими словами – объединяет две строки.

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

int main()
{
 setlocale(LC_ALL, "rus");

 char someText1[64] = "Сайт purecodecpp.com!";
 char someText2[] = "Учите C++ с нами!";

 cout << "Строка someText1- \"" << someText1 << "\" \n";
 cout << "Строка someText2- \"" << someText2 << "\" \n\n";

 int count1 = 0; // для индекса ячейки где хранится '\0' первой строки
 while (someText1[count1] != 0)
 {
 count1++; // ищем конец первой строки
 }

 int count2 = 0; // для прохода по символам второй строки начиная с 0-й ячейки
 while (someText2[count2] != 0)
 { // дописываем вконец первой строки символы второй строки
 someText1[count1] = someText2[count2];
 count1++;
 count2++;
```

```
 }

 cout << "Строка someText1 после объединения с someText2 -\n\"" << someText1 << "\" \n\n";

 return 0;
}
```

По комментариям в коде должно быть всё понятно. Ниже напишем программу для выполнения таких же действий, но с использованием strcat(). В эту функцию мы передадим два аргумента (две строки) – strcat(someText1, someText2); . Функция добавит строку someText2 к строке someText1. При этом символ '\0' в конце someText1 будет перезаписан первым символом someText2. Так же она добавит завершающий '\0'.

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
 setlocale(LC_ALL, "rus");

 char someText1[64] = "Сайт purecodecpp.com!";
 char someText2[] = "Учите C++ с нами!";

 cout << "Строка someText1 - \"" << someText1 << "\" \n";
 cout << "Строка someText2 - \"" << someText2 << "\" \n\n";
 strcat(someText1 , someText2); // передаём someText2 в функцию

 cout << "Строка someText1 после объединения с someText2 -\n\"" << someText1 << "\" \n\n";

 return 0;
}
```

Реализация объединения двух строк, используя стандартную функцию, заняла одну строчку кода в программе – 14-я строка.

Запустите программу и самостоятельно проверьте работоспособность.

На что следует обратить внимание и первом и во втором коде – размер первого символьного массива должен быть достаточным для помещения

символов второго массива. Если размер окажется недостаточным – может произойти аварийное завершение программы, так как запись строки выйдет за пределы памяти, которую занимает первый массив. Например:

```
char someText1[22] = "Сайт purecodecpp.com!";
strcat(someText1, "Учите C++ с нами!");
char someText1[22] = "Сайт purecodecpp.com!";
strcat(someText1, "Учите C++ с нами!");
```

В этом случае, строковая константа “Учите C++ с нами!” не может быть записана в массив someText1. В нём недостаточно места, для такой операции.

Если вы используете одну из последних версий среды разработки Microsoft Visual Studio, возможно возникновение следующей ошибки: «error C4996: ‘strcat’: This function or variable may be unsafe. Consider using strcat_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.» Так происходит потому, что уже разработана новая более безопасная версия функции strcat – это strcat_s.

Она заботится о том, чтобы не произошло переполнение буфера (символьного массива, в который производится запись второй строки). Среда предлагает вам использовать новую функцию, вместо устаревшей. Почитать больше об этом можно на сайте msdn. Подобная ошибка может появиться, если вы будете применять функцию strcpy, о которой речь пойдет ниже.

strcpy() (от слова copy – копирование)

Реализуем копирование одной строки и её вставку на место другой строки.

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

int main()
{
 setlocale(LC_ALL, "rus");

 char someText1[64] = "Сайт purecodecpp.com!";
 char someText2[] = "Основы C++";

 cout << "Строка someText1 - \"" << someText1 << "\" \n";
 cout << "Строка someText2 - \"" << someText2 << "\" \n\n";
```

```
int count = 0;
while (true) // запускаем бесконечный цикл
{
someText1[count] = someText2[count]; // копируем посимвольно

if (someText2[count] == '\0') // если нашли \0 у второй строки
{
break; // прерываем цикл
}

count++;
}

cout << "Строка someText1 после копирования someText2 -\n\"" << someText1 << "\" \n\n";

return 0;
}
```

Применим стандартную функцию библиотеки cstring:

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
setlocale(LC_ALL, "rus");

char someText1[64] = "Сайт purecodecpp.com!";
char someText2[] = "Основы C++";

cout << "Строка someText1 - \"" << someText1 << "\" \n";
cout << "Строка someText2 - \"" << someText2 << "\" \n\n";

strcpy(someText1, someText2); // передаём someText1 и someText2 в функцию

cout << "Строка someText1 после копирования someText2 -\n\"" << someText1 << "\" \n\n";
```

```
 return 0;
}
```

Пробуйте компилировать и первую, и вторую программу и сравните результаты.

strcmp() (от слова compare – сравнение)

Эта функция устроена так: она сравнивает две Си-строки символ за символом. Если строки идентичны (и по символам и по их количеству) – функция возвращает в программу число 0. Если первая строка длиннее второй – возвращает в программу число 1, а если меньше, то -1. Число -1 возвращается и тогда, когда длина строк равна, но символы строк не совпадают.

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

int main()
{
 setlocale(LC_ALL, "rus");

 char someText1[] = "Сайт purecodecpp.com!";
 char someText2[] = "Сайт purecodecpp.com/";

 cout << "Строка someText1 - \"" << someText1 << "\" \n";
 cout << "Строка someText2 - \"" << someText2 << "\" \n\n";

 int compare = 0; // для сравнения длины строк

 int count = 0;
 while (true)
 {
 if (strlen(someText1) < strlen(someText2))
 {
 cout << "Строки НЕ равны: " << --compare << endl;
 break;
 }
 else if (strlen(someText1) > strlen(someText2))
 {
 cout << "Строки НЕ равны: " << ++compare << endl;
 break;
```

```
 }
 else // если по количеству символов строки равны
 {
 if (someText1[count] == someText2[count]) // сравниваем посимвольно включая \0
 {
 count++;
 if (someText1[count] == '\0' && someText2[count] == '\0')
 {
 cout << "Строки равны: " << compare << endl;
 break;
 }
 }
 else // если все же где-то встретится отличный символ
 {
 cout << "Строки НЕ равны: " << --compare << endl;
 break;
 }
 }
 }
 return 0;
}
```

Программа с strcmp():

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
 setlocale(LC_ALL, "rus");

 char someText1[] = "Сайт purecodecpp.com!";
 char someText2[] = "Сайт purecodecpp.com/";

 cout << "Строка someText1 - \"" << someText1 << "\" \n";
 cout << "Строка someText2 - \"" << someText2 << "\" \n\n";

 cout << strcmp(someText1, someText2) << endl << endl;

 return 0;
```

}
Запускаем и сравниваем результат.