Строковый тип данных

В языке C++ стоки могут быть разделены на C-строки, представляющие собой массивы символов с нулевым байтом (нуль-терминатором) в конце, и C++-строки, представленные специальным типом данных «*string*».

Строки с нуль-терминатором

Язык C++ поддерживает тип данных *char*, но в переменных этого типа могут храниться только одиночные символы, но не текстовые строки. Если в программе необходимо работать с текстовыми строками, их можно создавать как массивы символов. В таком массиве для каждого символа строки отводится одна ячейка типа *char*, а последний элемент содержит символ конца строки – \0 (символ с нулевым кодом; нуль-терминатор).

При работе со строками необходимо четко различать запись 'F' (представляет собой символ, в памяти выделяется один байт) и "F" (представляет собой строку, в памяти выделяется два байта – символ 'F' и символ конца строки '\0').

Для ввода и вывода строк можно использовать рассмотренные ранее потоки ввода/вывода «*cin*» и «*cout*» (Листинг 1).

Листинг 1 – Пример кода работы с символьными массивами

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

int main() {
    locale::global(locale("Russian_Russia.1251"));
    char str1[5];
    char str2[10];
    char str3[4] = "C++";
    char str4[] = "C++";
    char str5[4] = {'C','+','+','\0'};

    //в массив str1 занесем слово "язык"
    str1[0]='я';
    str1[1]='з';
    str1[2]='ы';
    str1[3]='к';
    str1[4]='\0';

    cout << "Введите имя: ";
```

```
    cin >> str2;    //  воспринимает  ввод  строки  до  первого
                //пробела, далее символы не считываются

    cout << "Привет " << str2 << endl;
    return 0;
}
```

Как было отмечено в листинге 1, поток «*cin*» использует пустые символы, такие как пробел, табуляция или возврат каретки, для определения, где заканчивается одно значение и начинается другое. Для ввода строки, состоящей из слов разделенных пробелами, нужно использовать функцию «*getline*» потока «*cin*», которая имеет следующий формат:

```
cin.getline(<строка>, <максимальная-длина>);
```

Первый параметр — это имя символьной строки, а второй – ее размер. Для определения размера строки можно использовать оператор «*sizeof*», как показано в листинге 2.

Листинг 2 – Пример ввода строки с помощью «*cin.getline*»

```
#include <iostream>
using namespace std;
int main( )
{
    char st[10];

    cin.getline(st,sizeof(st));
    cout << st << endl;

    return 0;
}
```

С помощью рассмотренной функции «*getline*» можно организовывать ввод строк, ограниченных определенным символом. Для этого такой символ должен быть передан в функции в качестве третьего параметра. В листинге 3 приведен пример ввода строки, ограниченной либо возвратом каретки, либо указанным количеством символов, либо пока не встретиться символ 's'.

Листинг 3 – Пример ввода строки с ограничением

```
#include <iostream>
using namespace std;
int main( )
{
    char st[10];

    cin.getline(st,sizeof(st),'s');
```

```
    cout << st << endl;

    return 0;
}
```

Кроме массива, строки можно объявлять с помощью указателей, например:

```
char *str = "Строка";
char *str5;
str5 = "Строка текста";
```

При работе со строками при помощи указателей нужно следить, чтобы под описанный указатель была выделена память, т.к. в противном случае при вводе могут быть повреждены значения других переменных, и работа программы после этого станет непредсказуемой.

`char *p;`
`cin >> p;`
 	
Ошибочный код
	`char str[10];`
	`char *p = str;`
	`cin >> p;`
	
	Правильный код

Если введенные строки содержат меньше символов, чем зарезервировано ячеек в массиве «*str*», программа работает правильно. Если ввести строки большей длины, чем позволяет размер массива «*str*», на экране могут появиться «непонятные» символы или даже произойдет нарушение работы программы. Данный вид ошибок называется «переполнением ошибок» и является чрезвычайно опасным. Размер вводимых данных должен в обязательном порядке контролироваться при написании программы, проверяя, что длина строк не превосходит размер, отведенного под них массива.

Стандартные функции работы со строками

Для работы с нуль-терминантными строками в языке C++ используются функции, которые, в большинстве случаев, принимают в качестве аргумента имя массива символов, т.е. указатель на начало строки. К таким функциям относят функции: «*gets*», «*puts*», «*fgets*», «*fputs*», «*sprint*», «*strcpy*», «*strcat*», «*strncmp*», «*strlen*» (см. Листинг 4).

Листинг 4 – Пример программы ввода/вывода строк

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

```
int main() {
    locale::global(locale("Russian_Russia.1251"));
    char str[20];

    fputs("Введите первую строку ",stdout);
    gets(str);
    fputs("Вы ввели: ", stdout);
    puts(str);

    fputs("Введите вторую строку ",stdout);
    fgets(str, 20, stdin);
    fputs("Вы ввели: ", stdout);
    fputs(str, stdout);

    sprintf(str,"Это была %s проверка ","только");
    fputs("Функция sprintf создала: ", stdout);
    fputs(str, stdout);

    return 0;
}
```

Функция «*gets*» принимает символы от стандартного устройства ввода (в большинстве случаев это клавиатура) и помещает их в массив, указанный в качестве аргумента. Когда нажимается клавиша Enter для завершения ввода, генерируется символ новой строки ('\n'). Функция «*gets*» преобразует его в нулевой символ ('\0'), который служит признаком конца строки. Эта функция не позволяет определить превысило лАи количество введенных символов размер массива.

Функция «*puts*» выводит на экран текстовую строку, дополняя ее символом новой строки ('\n'), так что вызов данной функции два раза подряд, выведет на экран как минимум две строки текста.

Функция «*fgets*» аналогична функции «*gets*», но позволяет контролировать соответствие числа введенных символов установленным размерам массива. Символы читаются из указанного потока, в примере – это стандартное устройство ввода (stdin). Введенных символов должно быть на единицу меньше, чем размер массива: в последнюю позицию автоматически добавляется символ конца строки.

Совместно с функцией «*fgets*» используется функция «*fputs*», которая направляет символы в указанный поток, в листинге 2А3 – это устройство стандартного вывода (stdout).

Используемая в примерах выше функция «*strlen*» возвращает число символов в строке, указанной в качестве аргумента, не считая последнего, нулевого символа.

Функция «*strncmp*» сравнивает текстовую строку, заданную в первом аргументе, со второй строкой, начиная с первого символа. Длина первой строки задается в третьем аргументе. Если строки одинаковы, функция возвращает нулевое значение. Когда строки не одинаковы, функция возвращает отрицательное значение, если первая строка меньше второй, и положительное значение, если первая строка больше.

Функции стандартной библиотеки для работы со строками приведены в таблице 1.

Таблица 1 – Функции для работы со строками

Функция	Описание
strcat	*char *strcat(char *s1, char *s2);* Добавляет строку s2 в конец строки s1. В конце результирующей строки добавляя символ нуль-терминатора. Функция не производит выделение памяти, поэтому параметр s1 должен указывать на выделенную область памяти достаточного для совершения операции объема.
strchr	*char *strchr(char *s, int ch);* Ищет символ, заданный параметром *ch*, в строке *s*, возвращая указатель на первый найденный символ или NULL, если в строке нет заданного символа.
strcmp	*int *strcmp(char *s1, char *s2);* Сравнивает строки *s1* и *s2*, возвращает отрицательное значение, если строка s1 меньше s2, нулевое значение, если s1 и s2 являются одинаковыми строками, и положительное значение, если s1 больше s2.
strcpy	*char *strcpy(char *s1, char *s2);* Копирует значение строки s2 в память, заданную указателем s1. Функция не производит выделение памяти, поэтому параметр s1 должен указывать на выделенную область памяти достаточного для совершения операции объема.
strlen	*size_t strlen(char *s);* Возвращает длину строки (без учёта нуль-терминатора).

Функция	Описание
strncat	*char *strncat(char *s1, char *s2, size_t n);* Добавляет в конец строки s2 не более чем n символов из s1. Если длина s1 меньше n, то добавляется вся строка. Функция не производит выделение памяти, поэтому параметр s1 должен указывать на выделенную область памяти достаточного для совершения операции объема. Пример использования функции приведен в листинге 5.
strncmp	*int *strncmp(char *s1, char *s2, size_t n);* Сравнивает строки аналогично функции strcmp. Однако в сравнении принимает участие не более n символов. Пример использования функции приведен в листинге 6.
strncpy	*char *strncpy(char *s1, char *s2, size_t n);* Копирует значение строки s2 в память, заданную указателем s1. Длина скопированной части не будет превышать n символов в длину. Функция не производит выделение памяти, поэтому параметр s1 должен указывать на выделенную область памяти достаточного для совершения операции объема. Пример использования функции приведен в листинге 7.
strstr	*char *strstr(char *s1, char *s2);* Ищет подстроку s2 в строке s1 и возвращает указатель на элемент из s1, с которого начинается подстрока. Если заданной подстроки в s1 не найдено, возвращает NULL.
strlwr	*char *strlwr(char * s);* Преобразует строчные символы строки в прописные (обрабатывает только буквы латинского алфавита).
strupr	*char *strupr(char * s);* Преобразует прописные символы строки в строчные (обрабатывает только буквы латинского алфавита).
strset	*char *strset(char * s, char c);* Заполняет строку (до нуль-терминатора) указанным при вызове функции символом.

Замечание. Т.к. были введены в C++ дополнительные ограничения на исторически старые функции, чтобы избежать типичных случайных ошибок и учесть многобайтовые кодировки строк типа UTF-8 И Unicode, то просто вставьте в начало программы «**#define _CRT_SECURE_NO_WARNINGS**». Это отключит эти проверки.

Листинг 5 – Пример программы объединения строк функцией «*strcat*»

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
```

```
#include <string.h>

using namespace std;

#define size 64

int main() {
    locale::global(locale("Russian_Russia.1251"));
    char str1[] = “язык”,
         str2[] = “программирования”,
         str3[size];
    strcpy(str3, str1); // копирование строк
    strcat(str3, str2); // объединение строк
    strcat(str3, “C++”); // объединение строк
    cout<<str3<<”\n”; // вывод на экран
    return 0;
}
```

Листинг 6 – Пример программы сравнения строк функцией «*strncmp*»

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string.h>

using namespace std;

int main() {
    locale::global(locale("Russian_Russia.1251"));
    char str1[] = “язык ”, str2[] = “C++ ”;
    int length, result = 0;
    length = strlen(str1);
    if (strlen(str2)>=strlen(str1))
        result = strncmp(str1, str2, length);
    if (result==0) cout<<"Строки равны"<<endl;
    else
        cout<<"Строки не равны"<<endl;
    return 0;
}
```

Листинг 7 – Пример программы копирования строки функцией «*strcpy*»

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string.h>

using namespace std;

#define size 20

int main() {
```

```
    locale::global(locale("Russian_Russia.1251"));
    char str1[size] = "исходная строка",
          str2[size];
    strcpy(str2, "новая строка");
    cout<<"\n"<<str2;                // вывод на экран
    strcpy(str2,str1);
    cout<<"\n"<<str2;
    return 0;
}
```

Тип данных «string»

В предыдущих пунктах строка рассматривалась как массив символов оканчивающихся нуль-терминатором. Такой низкоуровневый подход работы с текстом дает в руки программиста большие возможности по управлению памятью и данными, но одновременно требует тщательного контроля над ошибками при работе с индексами массивов и указателями.

Чтобы избежать наиболее часто возникающих ошибок распространенной практикой в коллективах разработчиков было создание собственного типа данных, реализующего основные операции над строками. В современных версиях языка C++ такой высокоуровневый тип данных «*string*» (строка) стандартно описан в одноименном заголовочном файле.

Описание переменных и присвоение им значений может быть осуществлено так, как показано в листинге 8.

Листинг 8 – Описание и инициализация переменных типа «*string*»

```
//Описание и присвоение текстового значения
string str1;
str1 = "Первая строка";

//Описание с одновременной инициализацией
string str2 = "Вторая строка";

//Инициализация через вызов конструктора строки
string str3("Третья строка");
```

Очень важным отличием типа «*string*» от ранее рассмотренных нуль-терминантных строк является копирование содержимого строки при присвоении текстового значения переменной. Это справедливо как при показанной выше

инициализации, так и при присваивании значения переменной типа «*string*» другой переменной (см. Листинг 9).

Листинг 9 – Различие поведения при присвоении строк

```
locale::global(locale("Russian_Russia.1251"));

char *ns1 = new char[25]{ "Нуль-терминантная строка" };
char *ns2 = ns1;

string s1 = "Строка типа string";
string s2 = s1;

ns2[0] = '*';
s2[0] = '*';

cout << "ns1: " << ns1 << endl;
cout << "ns2: " << ns2 << endl;
cout << "s1: " << s1 << endl;
cout << "s2: " << s2 << endl;
```

Такое отличие объясняется тем, что нуль-терминантные строки являются указателями на область памяти (их формальный тип «*char**»), а переменные типа «*string*» являются отдельными объектами в памяти, обладающими собственными, управляемыми самостоятельно данными, и набором операций нам ними. Среди таких операций, в том числе есть и обязательное копирование данных при присвоении текстовых значений.

Для ввода значений типа «*string*» можно использовать рассмотренный ранее поток ввода «cin» и функцию «getline».

Для описания операций с типом «*string*» предположим, что в нашей программе описаны две переменные *s1* и *s2* данного типа. Тогда основные операции с ними можно описать с помощью таблицы 2.

Таблица 2 – Основные операции над переменной типа «*string*»

Операция, функция	Описание
s1.c_str()	Преобразует строку к типу данных «*const char **», возвращая ссылку на начало текстовой строки с нуль-терминатором. Изменение строки через полученный указатель невозможно (т.к. он ссылается на константную строку), однако данная функция может быть использована при работе с кодом, рассчитанным на «старый» формат описания строк.

Операция, функция	Описание
s1.empty()	Возвращает логические признак пустоты строки (true – если строка пуста, false – если в строке есть хотя бы один символ). Альтернативным способом проверки на пустоту может служить проверка длины строки на равенство нулю «*s1.length() == 0*».
s1.find(s2, pos)	Возвращает номер первого вхождения строки *s2* в строку *s1*. Поиск начинается с номера *pos*. Параметр *pos* может отсутствовать, в этом случае поиск идет с начала строки.
s1.insert(p, s2)	Вставляет строку (или символ) *s2* в строку *s1*, начиная с позиции *p*.
s1.length()	Получение длины строки (количество символом в строке).
s1.remove(p, len)	Удаляет из строки *s1* подстроку длиной *len*, начиная с позиции *p*.

s1.substr(p, len)	Возвращает копию подстроки строки *s1*, начиная от символа с номером *p* и длиной *len*.
	`string s1 = "Строка №1";`
	`auto s3 = s1.substr(7, 2);  //"№1"`

s1 += s2	Добавляет в конец строки *s1* текст строки *s2*. Результат операции сохраняется в строке *s1*.
s1 + s2	Добавляет в конец строки *s1* текст строки *s2*. Результат операции возвращается в виде новой строки.
s1 > s2 s1 < s2 s1 >= s2 s1 <= s2	Сравнивает две строки между собой на соответственно больше, меньше, больше или равно, меньше или равно, возвращая результат в виде логического значения. Сравнение производится в лексикографическом порядке.
s1 == s2 s1 != s2	Сравнивает две строки на равенство/неравенство друг другу, возвращая результат в виде логического значения. Сравнение производится в лексикографическом порядке.
s1 = s2	Копирует значение текстовой строки *s2* в строку *s1*. Исходный текст, содержащийся в строке *s1* до операции, теряется.
s1[index]	Возвращает символ (тип *char*) строки находящийся в позиции *index*. Как и в нуль-терминантных строках нумерация символов начинается с нуля.
s1[index] = c	Заменяет в строке *s1* символ в позиции *index* на символ, содержащийся в переменной *c*.