1. С-строки

В С++ имеется два типа строк: встроенный тип, унаследованный от языка С (строки данного типа мы будем называть С-строками), и класс `string` из стандартной библиотеки С++. Класс `string` появился в стандарте языка в августе 1998 г. и может быть не реализован в устаревших компиляторах. Мы рассмотрим лишь С-строки, поскольку они тесно связаны с указателями и используются для получения эффективного кода.

С-строка – это массив символов, оканчивающийся символом с кодом 0, или *нулевым символом* (`'\0'`). К строковым константам нулевой символ добавляется автоматически:

```
sizeof("LoveC++")==8
```

Доступ к строке обычно осуществляется с помощью указателя `char*`, поэтому, как правило, тип `char*` ассоциируется именно со строкой. Так, если переменная `s` имеет тип `char*`, то при выполнении оператора

```
cout<<s;
```

в поток вывода записываются символы, на которые указывает `s`, до тех пор, пока не будет встречен нулевой символ `'\0'`.

Всюду далее, если не оговорено противное, под строкой будем понимать именно С-строку.

1.1. Описание и инициализация строк

Для строки резервируется массив символов:

```
char s[20];
```

Длина массива должна быть достаточной для хранения всех возможных строковых значений, которые могут встретиться при работе с данной строковой переменной (напомним, что контроль выхода за границы массива в С++ отсутствует).

При описании строка может быть инициализирована строковой константой:

```
char s1[20]="String";
char s2[5]="Hello"; // ошибка: не отведено место
                    // под завершающий '\0'
```

Если требуется описать именованную строковую константу, то используются следующие варианты инициализации:

```
const char s3[]="Hello";
const char* s4="Good Bye";
```

Под массив `s3` при этом отводится память из шести элементов типа `char`. Заметим, что, в отличие от `s3`, указатель `s4` может менять свое значение в процессе работы.

Можно также встретить аналогичную инициализацию без `const`:

```
char s3[]="Hello";
char* s4="Good Bye";
```

Она не вполне корректна, поскольку позволяет модифицировать константные по смыслу данные, однако ее можно встретить в реальных программах.

1.2. Ввод строк

Оператор >> позволяет ввести слово:

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

При этом в потоке ввода вначале пропускаются все символы-разделители (пробелы, символы перехода на новую строку и символы табуляции), затем в переменную `word` считываются символы до символа-разделителя и дописывается нулевой символ. Основная ошибка здесь – это попытка ввести больше символов, чем вмещает в себя массив символов `word`. Например, при вводе строки `"  abracadabra  "` два лидирующих пробела будут пропущены, в массив `word` будут записаны символы `"abracadabr"`, а символы `'a'` и `'\0'` попадут в следующие за `word` ячейки памяти. Сообщение об ошибке при этом, как правило, не возникнет.

Для решения проблемы выхода за границы массива-строки в приведенном выше примере следует использовать манипулятор `setw`, устанавливающий максимальную ширину поля ввода:

```
#include <iomanip.h>
...
cin>>setw(10)>>word;
```

В этом случае в массив `word` попадут символы `"abracadab"` плюс завершающий нулевой символ, а символ `'a'` останется в потоке ввода. Заметим, что использование манипулятора `setw` влияет только на непосредственно следующую за ней операцию ввода.

Если требуется ввести не отдельное слово, а строку целиком, то используется функция-член `getline` класса `istream`, к которому принадлежит поток `cin`:

```
cin.getline(word,10);
```

В этой ситуации ввод будет осуществляться до символа перехода на новую строку `'\n'`, но максимально будет считано 9 символов, после чего к строке `word` будет добавлен `'\0'`.

Функция `getline` имеет также третий параметр – символ-разделитель, до которого осуществляется считывание. По умолчанию он равен `'\n'`, но может быть явно изменен. В следующем примере считывание осуществляется до символа `'!'`, но не более 9 символов:

```
cin.getline(word,10,'!');
```

Отметим, что сам символ-разделитель удаляется из потока ввода.

1.3. Копирование строк

Поскольку строка – это массив символов, то копирование строк невозможно осуществить с помощью оператора присваивания. Действительно, если имеются следующие описания

```
char s1[10]="Hello",s2[10],*s3;
```

то присваивание s1=s2 вызовет ошибку компиляции, поскольку имя массива s1 является константным указателем. Присваивание же s3=s1 вполне законно, но оно не копирует данные из одной строки в другую, а лишь инициализирует указатель s3 адресом начала строки s1. В этом случае любое изменение строки через указатель s3 меняет исходную строку s1.

Поскольку строка завершается нулевым символом, ее копирование имеет специфику. Рассмотрим вначале несколько способов копирования без привлечения библиотечных функций.

Следующий цикл производит посимвольное копирование из строки s1 в строку s2 до того момента, как в строке s1 встретится нулевой символ:

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

После цикла нулевой символ дописывается в конец строки s2 (число 0 неявно преобразуется в символ '\0').

Посимвольное копирование можно также осуществить, используя указатели:

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

Если *p становится равным нулю, то достигнут конец строки s1, и цикл завершается. Завершающий нулевой символ также приходится дописывать «вручную». Обратите внимание, что после цикла длина строки s1 может быть вычислена как p-s1.

Наконец, учитывая то, что оператор присваивания возвращает значение левой части после присваивания, мы можем записать алгоритм копирования максимально компактно:

```
char *p=s1,*q=s2;
while (*q++=*p++);
```

На последней итерации цикла *p становится равным нулю, это значение присваивается *q и возвращается как результат оператора присваивания, что и приводит к завершению цикла.

1.4. Стандартные функции работы со строками

Функции для манипулирования С-строками объявлены в заголовочном файле <string.h> (или по стандарту 1998 г. в <cstring>). Наиболее часто используются функции strlen, strcpy, strcmp, strcat.

Функция strcpy имеет следующий прототип:

```
char *strcpy(char *p, const char *q);
```

Она копирует строку q в строку p, включая нулевой символ, и возвращает указатель на строку p. При этом выход за границы массива p не контролируется:

```
char s1[6]=”Hello”, s2[20]=”Good bye”;
strcpy(s2,s1); // верно
strcpy(s1,s2); // ошибка!
```

Возможные реализации функции `strcpy` приведены в предыдущем пункте.

Для определения длины строки служит функция `strlen` с прототипом

```
size_t *strlen(const char *p);
```

Использовать ее предельно просто:

```
char s3[20]=”abracadabra”;
size_t sz=strlen(s3); //  sz==11
```

Очевидно, что для вычисления длины строки функция `strlen` должна просканировать строку до конца. Возможная реализация `strlen` выглядит так:

```
size_t *strlen(const char *p)
{
  size_t len;
  while (*p++) len++;
  return len;
}
```

Для добавления одной строки к другой используется функция `strcat` с прототипом:

```
char *strcat(char *p, const char *q);
```

Данная функция добавляет содержимое строки `q` в конец строки `p` и возвращает полученную строку. Например:

```
char s[20]; s1[]=”love”;
strcpy(s,”I ”);
strcat(s,s1);
strcat(s,” C++”); // s==”I love C++”
```

Возможная реализация `strcat` приведена ниже:

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

Отметим, что если длина исходной строки известна заранее, то вместо `strcat` можно воспользоваться `strcpy`:

```
strcpy(s,”I ”);
strcpy(s+2,s1);
strcpy(s+7,” C++”);
```

Такой алгоритм эффективнее, так как строка, к которой происходит добавление, не сканируется в поисках завершающего нуля.

Функция `strcmp` предназначена для лексикографического сравнения строк. Она имеет прототип

```
int strcmp(const char *p, const char *q);
```

и возвращает 0 если строки совпадают, положительное число, если первая строка больше второй, и отрицательное – если вторая больше первой. Более точно: производится посимвольное сравнение строк до обнаружения пары различных символов или до конца одной из строк и возвращается разность кодов первых не совпавших символов или 0.

Возможная реализация `strcmp` имеет вид:

```
int strcmp(const char *p, const char *q)
{
  while(*p==*q && *p)
  {
    p++; q++;
  }
  return *p-*q;
}
```

В заключение данного пункта приведем менее употребительные функции работы с С-строками:

```
char *strncpy(char *p, const char *q, int n);
```

– то же, что и `strcpy`, но копируется максимум `n` символов.

```
char *strncat(char *p, const char *q, int n);
```

– то же, что и `strcat`, но добавляется максимум `n` символов.

```
int strncmp(const char *p, const char *q, int n);
```

– то же, что и `strcmp`, но сравнивается максимум `n` символов.

```
char *strchr(const char *p, char c);
```

возвращает адрес первого вхождения символа `c` в строку `p` (или 0, если символ не найден).

```
char *strrchr(const char *p, char c);
```

возвращает адрес последнего вхождения символа `c` в строку `p` (или 0, если символ не найден).

```
char *strstr(const char *p, const char *q);
```

возвращает адрес первого вхождения подстроки `q` в строку `p` (или 0, если подстрока не найдена).

```
char *strpbrk(const char *p, const char *q);
```

возвращает адрес первого вхождения в строку `p` какого-либо символа из строки `q` (или 0, если совпадений не обнаружено).

```
size_t strspn(const char *p, const char *q);
```

возвращает число начальных символов в строке `p`, которые не совпадают ни с одним из символов из строки `q`.

```
size_t strcspn(const char *p, const char *q);
```

возвращает число начальных символов в строке p, которые совпадают с одним из символов из строки q.

```
char *strtok(char *p, const char *q);
```

– последовательность вызовов функции strtok разбивает строку p на лексемы, разделенные символами из строки q. При первом вызове в качестве p передается указатель на строку, которую надо разбить на лексемы, при всех последующих – нулевой указатель. При этом значение указателя, с которого должен начинаться поиск следующей лексемы, сохраняется в некоторой системной переменной. Функция strtok возвращает указатель на найденную лексему или 0, если лексем больше нет. Она также модифицирует исходную строку, вставляя нулевой символ после найденной лексемы. Далее в пункте 10.6 будет дан пример использования функции strtok.