Strings

Lecture 2

Strings in C++

- We've been using strings since the beginning

```
cout << "Hello World" << endl;
```

Everything between the double quotes is a string

- In general, a string is a sequence of characters (letters, digits, spaces, punctuation, etc.)

- In C++, there are two different string data types
  - C strings
  - The **`string`** class

C Strings

- C strings are arrays of characters, inherited from C
  - Example: `char str[16];`
- In C++, string values (characters between double quotes) are C strings
  - Example: `"Hello World"` is a C string, which means that it is treated as a `char` array
- Most of the time, C string variables are not completely full, so we need a special marker to denote where the actual string ends in the `char` array…

The Null Character

- The null character `'\0'` is used to tell C++ where the string ends
  - Note: like tab (`'\t'`) and new line (`'\n'`), the null character is a *single* character but requires the backslash to represent in C++ (it is actually ASCII value 0)

- For example, the C-string `"bob"` actually takes four characters to store: three for the three letters in the string plus one more for the `'\0'` null character at the end

```
char s[8] = "bob";
// s[0]='b', s[1]='o', s[2]='b', s[3]='\0', s[4]=?, s[5]=?, …
```

- Importantly, the `char` array must be large enough to store the extra null character

Declaring and Initializing C Strings

- When you declare a C string (`char` array), you can initialize it at the same time

```
char message[10] = "Hi there";
```

  - C++ automatically adds the null character at the end of the string (`message[8]` is set to `'\0'`)

- You can leave out the array size when you initialize a C string this way

```
char message[] = "Hi there";
```

  - C++ will size the array automatically, including the null character at the end, so the above is equivalent to…

```
char message[9] = "Hi there";
```

- However, the typical array initialization is **not** equivalent

```
char message[] = {'h','i',' ','t','h','e','r','e'};
```

  - C++ will make an array of size 8 without a trailing null (`'\0'`)

Accessing C String Contents

You can use C strings like other arrays by using the square brackets

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

int main()
{
   char str[] = "Wentworth Is Terrific!";
   cout << str[0] << str[10] << str[13] << str[21];
   return 0;
}
```

Exercise

In C++ you generally need to know the size of an array to iterate over all elements. This is not the case with C strings – because you know they end with a null character! Write a function named **`print_str`** that takes a single argument (a C string) and outputs the string to the screen.

Answer

```
void print_str(char str[])
{
  for ( int i=0; str[i]!='\0'; i++ )
  {
    cout << str[i];
  }
}
```

Exercise

Make a copy of **print_str** and name it **`str_size`** – modify this new function to *return* the length of the string.

Answer

```
int str_size(char str[])
{
   int i;
   for ( i=0; str[i]!='\0'; i++ );
   return i;
}
```

Do Not Overwrite Null!

- C strings must always end with the null character, so you have to be careful not to overwrite it

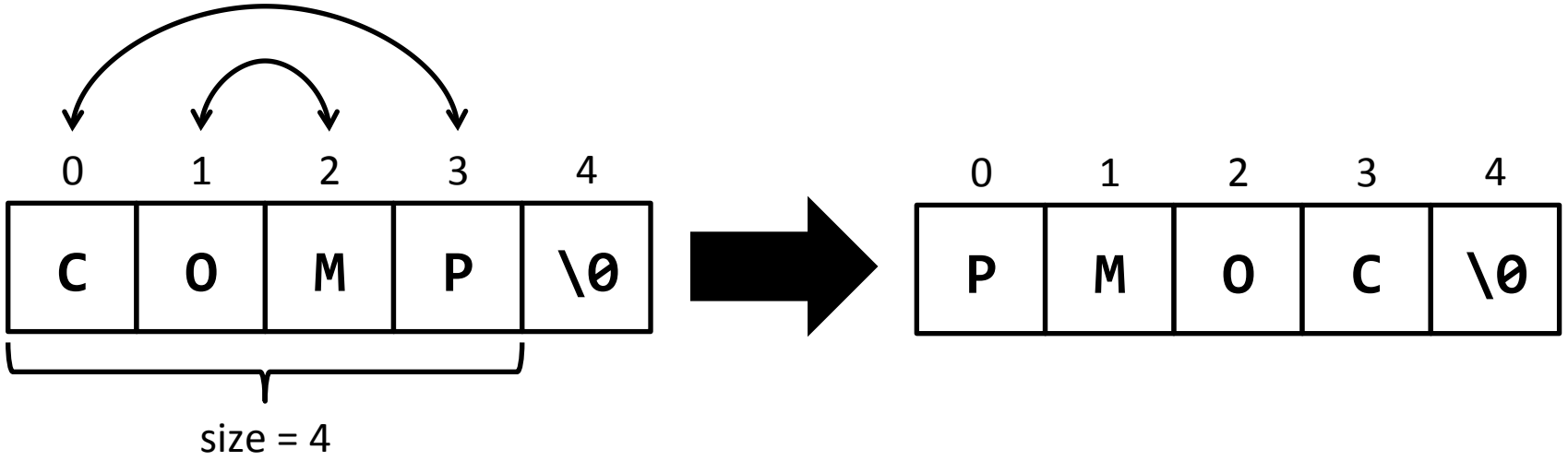
- If the `'\0'` is lost, then most C string manipulations will go out of bounds of the array and **Bad Things** (overwriting memory) might happen (think about the code you just wrote!)

```
char my_string[4] = "abc"; // my_string[3] = '\0'
my_string[3] = 'd';        // overwrite the '\0'

print_str( my_string );    // prints out abcd and
                           // then garbage until '\0'
                           // is reached
```

Exercise

Write a function named **`reverse`** that takes as input a C string and reverses it. For example, if "COMP" were the input, the C string should be changed to "PMOC" – you may wish to use your **`str_size`** function…



Answer

```
void reverse(char str[])
{
   int size = str_size( str );
   for ( int i=0; i<( size/2 ); i++ )
   {
      char temp = str[i];
      str[i] = str[ size-i-1 ];
      str[ size-i-1 ] = temp;
   }
}
```

Swap!

Output with C Strings

- You can use the `<<` operator directly with a C string – it will send characters until the null character is reached

- You wrote the following functionality with the **`print_str`** function

```
char msg[] = "howdy";
cout << msg;
```

Input with C Strings (1)

You can use the >> operator directly with C strings, but be careful!

- It reads only the first "word" (stops at tab, space, new line, any whitespace)
- It does **not** respect the length of the array

```
char target[3];
cin >> target;
cout << target;
```

```
> 5 is my lucky number
5

> five is my lucky number
five
```

Input with C Strings (2)

To get an entire "line" of input (up to a maximum number of characters) in a C string, use istream.getline:

```
char target[5];
cin.getline( target, 5 );
cout << target;
```

```
> 5 is my lucky number
5 is

> five is my lucky number
five
```

Buffer Overflow

- When a program writes more data to a variable than is memory-allocated
- Sometimes innocuous, usually leads to hard-to-find bugs, can breach security
- **Bounds checking** can prevent buffer overflows!

Simple Buffer Overflow Example

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

int main()
{
     char a[] = "foo";
     char b[] = "bar";

     cout << a << " " << &a << endl;
     cout << b << " " << &b << endl;

     cin >> b;

     cout << a << endl;
     cout << b << endl;

     return 0;
}
```

```
foo 0x7fff50455a68
bar 0x7fff50455a64

> howdy!

y!
howdy!
```

Working with C Strings

- You can only use = with C strings during initialization (as part of the declaration)
  - If you try to assign a new value to C-string after declaration you will get a build error:

```
char str[] = "something";
str = "something else";   // build error
```

- You can not use == with C strings
  - You will not get an error but you will not get the result you expect because it does not compare the actual values in the strings

The `cstring` Library

The `cstring` library has functions that can assign, compare, and manipulate C strings, for example…

Copy: **`strncpy( target, source, limit );`**

- Watch out for **`strcpy`**: risk of buffer overflow!

Concatenate: **strncat(target, source, limit);**

- Watch out for **strcat**: risk of buffer overflow!

Compare: **strcmp(str1, str2);**

- =0 if same
- <0 if **str1 < str2**
- >0 if **str1 > str2**

– Length: **`strlen( str );`**

- Returns an integer

Exercise

Write a program that reads two words from the user, compares them, and prints out whether or not they are the same

Answer

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

int main()
{
	char a[100];
	char b[100];

	cin >> a >> b;

	if ( strcmp( a, b ) == 0 )
	{
		cout << "Same" << endl;
	}
	else
	{
		cout << "NOT the same" << endl;
	}

	return 0;
}
```

C String -> Numbers

The `cstdlib` library has functions to convert a C string to a number

```
int i = atoi( str );
double d = atof( str );
```

– These functions return `0` if unsuccessful

Program Arguments

- One reason you will have to use C strings is for *program arguments*
- Just like functions, your program can take arguments from the command line
- These are represented as arguments to the **`main`** function

Changing `main`: `argc`/`argv`

- To gain access to program arguments, change your main function

```
int main(int argc, char* argv[])
```

- The **`argv`** argument is an array of C strings (we will discuss the * next lecture)

- The **`argc`** argument tells you the array size
  - The size is at least 1 and the first element of **`argv`** is how the program was invoked

Example

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

int main(int argc, char* argv[])
{
   int i;
   for ( i=0; i<argc; i++ )
      cout << i << " = "
           << argv[i]
           << endl;

   return 0;
}
```

```
> ./a hello friend
0 = ./a
1 = hello
2 = friend
```

Exercise

Write a program named **`mult`** that requires two command line arguments. Convert them to integers and output the result of multiplying them together.

Answer

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

int main(int argc, char* argv[])
{
    if ( argc != 3 )
    {
        cout << "Usage: " << argv[0] << " <int 1> <int 2>" << endl;
        return 0;
    }

    int x = atoi( argv[1] );
    int y = atoi( argv[2] );

    cout << argv[1] << " * " << argv[2]
         << " = " << ( x * y ) << endl;

    return 0;
}
```

Motivation for the `string` Class

- C strings in C++ were inherited from the C language
  - They are minimal and fast
  - Some functions require their usage
  - They can be awkward to use, and expose your code to risk of bugs and security breach

- In C++, the **`string`** class makes it easier and safer to input and manipulate strings
  - You do NOT need to know the size ahead of time, and you can easily grow the string at will
  - Remember you must include the `string` library

Declaring and Initializing Strings

In COMP128 we primarily used the *default constructor* for declaring string variables, but there are other options – all the following statements are equivalent ways to declare the variable **`str`** and initialize its value to the string `"foo"`)

```
string str;
str = "foo";

string str = "foo";

string str( "foo" );
```

Accessing String Contents

- To get/set individual characters within a string, use either the `[]` operator or the **at** function
  - Note: the `[]` operator will NOT check to see if the supplied index is valid

- The **`length`** function returns the number of characters

```
string str( "abc" );
cout << str.length() << endl; // 3

cout << str[0] << endl; // a
cout << str[3] << endl; // ?

cout << str.at(0) << endl; // a
cout << str.at(3) << endl; // runtime error
```

Exercise

Write a new **`reverse`** function whose argument is a string and *returns* a string whose characters are in reverse order

Answer

```
string reverse(string str)
{
   string rstr = str;

   for ( int i=0; i<str.length(); i++ )
   {
      rstr.at( i ) = str.at( str.length() - i - 1 );
   }

   return rstr;
}
```

I/O with Strings

- Output works directly with the `<<` operator

```
cout << str;
```

- Input with the `>>` gets a string up to the first whitespace

```
cin >> str; // if input="hi there"
               str="hi"
```

- The getline function reads an entire line

```
getline( cin, str );
```

Exercise

Write a program that reads a line of text from the keyboard and outputs only characters that are uppercase, or not letters. For example:

```
> Hi There Terrific Programmer!
H T T P!
```

Answer

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

int main()
{
     string str;
     getline( cin, str );

     for ( int i=0; i<str.length(); i++ )
     {
          char c = str.at( i );

          if ( !( c >= 'a' && c <= 'z' ) )
          {
               cout << c;
          }
     }

     return 0;
}
```

Working with Strings

The **`string`** class has many additional useful functions, for example…

- Comparison
  - **str1 == str2; // returns true if contents are the same**
    - Also: **!=, <, >, <=, >=**
- Assignment
  - **str1 = str2;**
  - **str1 = `"stuffs"`;**
- Concatenation
  - **str1 + str2;**
  - **str1 = str2 + str3;**
  - **str1 += str2;**
- Substring
  - **str1.substr(position, length);**

Exercise

Write a function named **`palindromeflip`** that takes as input a string and returns a palindrome (i.e. one that reads the same front to back as back to front) in a very specific way: append to the original string a copy in reverse order. For example, if the input were “abc” then the returned string would be “abccba”

Hint: make use of your **`reverse`** function!

Answer

```
string palindromeflip(string str)
{
  return ( str + reverse( str ) );
}
```

Strings -> C Strings

- Some functions require a C string – if you are using the **string** class and need to get access to its equivalent C string representation, use the **c_str** function

- For example, in GCC 4.8, opening files must be done with a C string

```
string fname;
ifstream inf;
cin >> fname;
inf.open( fname.c_str() );
…
```

Wrap Up

- In C++, strings can be represented either by C strings or the **`string`** class
- C strings are a char array with a null character (**`'\0'`**) indicating the end of the string
- There are useful functions in the **`cstring`** and **`cstdlib`** libraries for C strings, but be wary of buffer overflows
- Program arguments are accessed via the **`argc`** and **`argv`** arguments of the **`main`** function, where **`argv`** is an array of C strings and **`argc`** is the size of the array
- Using the **`string`** class is generally preferred both for ease and safety