C++ Strings

Two types of strings in C++ :

- **C-strings**
- **The Standard `string` Class**
  (will be discussed in cmpt135)

What is a C-string?

- A string is an ordered sequence of characters
- A C-string is
  - **a character array**
  - **ended with null character (‘\0’)**
- E.g. the following is a C-string named **s** that is size 10 and represents string **“Hi you!”**

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]
H	i		y	o	u	!	\0	?	?

Declaring C-strings

- To declare a C-string variable, declare an array of characters, for example:

```
char s[10];
```

  - **Declaring a C-string as `char s[10]` creates space for a string that can only contain maximum nine characters**
  - **The null character terminator requires one space**

Initializing a C-string

- You may initialize the C-string character by character after it is declared (Remember to include the null character!)
- More often, initialize a C-string during declaration:

```
char my_message[20] = "Hi there.";
```

- Another alternative:

```
char short_string[] = "abc";
```

- For each of the above, the null character is automatically added for you
- but not this -- '\0' is not inserted in this array:

```
char short_string[] = {'a','b','c'};
```

Don't Change '\0'

- "size" variable may not be needed when processing C-strings
  - Just use the null character

```
int index = 0;
while (our_string[index] != '\0')
{
    our_string[index] = 'X';
    index++;
}
```

    - The termination of the loop depends on the null character!
  - If the null character is lost, the array cannot act like a C-string.

Safer Processing of C-strings

- The loop on the previous slide depended on finding the '\0' character
- It would be wiser to use this version in case the '\0' character had been removed

```
int index = 0;
while(our_string[index] != '\0' && index < SIZE)
{
     our_string[index] = 'X';
     index++;
}
```

Library

- Declaring c-strings
  - Requires no C++ library
  - Built into standard C++
- Using C-string functions
  - Require library `<cstring>`
  - Typically included when manipulating c-strings

C-string Arguments and Parameters

- Recall: **c-string is array**
- So c-string parameter is array parameter
  - *C-strings passed to functions can be changed by receiving function!*
- Like all arrays, you may send size into functions as well
  - Function could also use '\0' to find end, or use `strlen`
  - So size parameter is not a must in most cases
  - Use "`const`" modifier to protect c-string arguments when applicable

C-string Functions: `strlen`

- Defined in `<cstring>` library

- returns the length/size of the string

```
char myString[10] = "dobedo";
cout << strlen(myString); // 6
```

  - Returns the number of characters
    - Not including the null character

  - How can we define our own strlen function?

= and == with C-strings

- Recall: **C-string is array!**
- C-string variable cannot be assigned:

```
char aString[10];
aString = "Hello";  // illegal!
```

  - **Can ONLY use "=" at declaration of the C-string!**

- Using **`operator ==`** is only to compare pointers, not the content of the arrays :

```
char s1[10] = "Hello";
char s2[10] = "Hello";
s1 == s2;  // returns false
```

Make a copy of a C-string

- How to make a deep copy of a C-string?

```
strcpy (copy, original);
```

  - `strcpy` is a built-in function in <cstring>
  - both `copy` and `original` are C-strings
  - Copy `original`'s characters to `copy`
  - `copy` should have memory space allocated already!
  - Be careful, this function does not check whether `copy` has enough size for `original`
    - programmer's responsibility
  - How can we define our own strcpy function?

Comparing C-strings

- We know that

```
char s1[10] = "Hello";
char s2[10] = "Hello";
s1 == s2; // will give us false
```

- How can we compare whether two strings are the same?
  -- We can use <cstring> function strcmp

```
if (strcmp(s1, s2))
    cout << "Strings NOT same.";
else
    cout << "Strings are same.";
```

  -- How can we define our own strcmp function?

Returning a dynamic c-string

- **Exercise 1:**

  Define a function called *initials_1* that takes two names (*a first name and a last name, both are cstrings*) as its parameters and returns the initials of the two names in the form of "A.B."

  E.g. `initials_1("Donald", "Knuth")` returns "D.K."

- **Exercise 2:**

  Define a function called *initials_2* that takes a full name (*a single string consisting of a first name and a last name separated by a space character*) as its parameter and returns the initials

  E.g. `initials_2("Donald Knuth")` returns "D.K."

C-String Output

- Can output with insertion operator **<<** because **<<** operator is overloaded for C-strings

  E.g.

```
char news[] = "Earthquake!";
cout << news << endl << "When?";
```

C-String Input

- Can input with extraction operator >>
  - Issues exist, however
- Whitespace is "delimiter"
  - Tab, space, line breaks are "skipped"
  - Input reading "stops" at delimiter
- Watch size of c-string
    - Must be large enough to hold entered string!
    - C++ gives no warnings of such issues!

C-String Input Example

```
char s1[80], s2[80];
cout << "Enter input: ";
cin >> s1 >> s2;
cout << s1 << s2 << "END OF OUTPUT\n";
```

- Sample run:

```
Enter input: Do be do to you!
DobeEND OF OUTPUT
```

  - Underlined portion is user input
  - C-string **s1** receives "**Do**"
  - C-string **s2** receives "**be**"

C-String Line Input

- Can receive entire line into c-string
- Use `getline()`, a predefined member function:

```
char str[80];
cout << "Enter input: ";
cin.getline(str, 80);
cout << str << "END OF OUTPUT\n";
```

  - Dialogue:

```
Enter input: Do be do to you!
Do be do to you!END OF OUTPUT
```

The Standard `string` Class

- The `string` class allows the programmer to treat strings as a basic data type
  - No need to deal with the string at array level
- The `string` class is defined in the `string` library and the names are in the standard namespace
  - To use the `string` class you need to include it:

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

We will discuss `string` class in more detail in CMPT135