Lecture 13: C++ string class

string processing and character processing

Strings

- There are 2 types of strings in C++:
  - One inherited from the C language, we call them `cstrings`
  - The other is defined in the ANSI Standard Library `<string>`
- Any sequence of characters enclosed with a pair of " " is a cstring value, e.g.

```
cout << "Hello world\n";
```

- A cstring can be stored in an array of characters
- The following array can store a cstring of length at most 9 (not 10): `char s[10];`

- By the following array declaration and initialization:

```
char s[10] = "Hi Mom!";
```

  array `s` contains:

0	1	2	3	4	5	6	7	8	9
H	i		M	o	m	!	\0	\0	\0

- The null character `'\0'` is used to mark the end of a cstring. ASCII value of `'\0'` is 0.

Initializing a cstring variable

- During declaration:

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

- The cstring assigned to the variable need not fill the entire array
- You can omit the size if you initialize:

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

  The system will declare it as an array of 4 chars.

- This is different from :

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

  The system will declare it as an array of 3 chars.

Inputting and Outputting a Cstring

```
void main()
{
   char message[20];
   cin >> message;
   cout << message;
}
```

- In the statement:

```
   cin >> message;
```

  a null character will be appended at the end of the input string.

Passing a Cstring into a function

```
int str_length(char s[])
{  int i=0;
   while (s[i]!='\0') i++;
   return i;
}
void main()
{
   char message[20]="Hi Mom!";
   cout << str_length(message);
   cout << str_length("How are you");
}
```

- No need to pass the length of a cstring into a function. The null character can be used to tell the end of the cstring.

C++ standard `string` class

- Defined in standard library `<string>`
- Provides a lot more functions than `<cstring>` in a much more natural way
- E.g.,

```
  string phrase, word1("hot"), word2("dog");
  phrase = word1 + word2;
```

declares 3 string objects, `phrase`, `word1` and `word2`; concatenate `word1` with `word2`; and copy to `phrase`

String constructors

- Create empty strings:

```
string str;
```

- Create string objects from cstrings:

```
string str("abc");
```

  Internally, `str` stores the 3 characters `'a'`, `'b'`, `'c'` and the number 3

- Create string objects from another string:

```
string str(another_str);
```

Accessing strings

- `[]` is overloaded so that you can access individual characters as if using an array. (`[]` is regarded as an operator in C++.)
- E.g.,

```
for (i=0; i<surname.length(); ++i)
        cout << surname[i] << " ";
```

- The `[]` operator does not do index range checking

- The member function **`at`** performs <u>range checking</u>
- E.g.
```
string str("Mary");
cout << str[6];       // no complain
cout << str.at(6);  // error
```
- Write is also possible: `str.at(2) = 'X';`
- Extract a (read-only) substring:
```
str.substr(start_pos, length);
```
- E.g.
```
string str("computer");
cout << str.substr(3,3);  // put
str.substr(3,3)="mut";  // error
```

String assignment and modifiers

- Copy one string to another:

```
str1 = str2;
```

- Concatenation:

```
str1 + str2
```

  returns a string with `str2` appended to `str1`

- Test for empty string:

```
str.empty();
```

- Insert and remove substrings:

```
str.insert(start_pos, str2);
str.erase(start_pos, length);
```

String comparison

- Equality and inequality:

```
str1 == str2
str1 != str2
```

- Lexicographical comparisons:

```
str1 < str2
```

  `>` `<=` `>=` are similar

- Finding a subtring:

```
str.find(str1)
```

  returns index of first occurrence of `str1` in `str`
  or `string::npos` if `str1` is not found

String expressions

- Automatic type conversion is done by constructor in the following:

```
phrase = word1 + " " + word2;
```

- Note: `" "` is a cstring, not a string
- Parentheses are not needed, + operators applied from left to right

String input/output

- << and >> are also overloaded for strings; the operator >> reads a word (of non-whitespace characters)
- To read an entire line (up to the newline character), use the `getline` function which is an ordinary (non-member) function with 2 or 3 parameters:
  - 1st parameter: an input stream
  - 2nd parameter: a string
  - 3rd parameter: terminating character, default to `'\n'`

- E.g.

  ```
  string str1;
  getline(cin,str1);
  ```

  inserts into `str1` all that is typed up to `'\n'`; the `'\n'` is removed from `cin` and discarded
- Note: There is another `getline` function which is for cstrings and is a member function of all input streams. E.g.:

  ```
     char input[500];
     cin.getline(input,500);
  ```

  It will read at most 500 characters (including `'\0'`)

Ignore member function

- Consider `cin >> x;  // x is integer`

  When user types in some characters after a number, these extra characters will be left in `cin` and may corrupt the next extraction

- To skip extra inputs:

  ```
  cin.ignore(count,delimiter);
  ```

  Read up to `count` characters, or until `delimiter` is reached, whichever is first, and discard these characters. If `delimiter` is found, it is removed from the input stream

Predefined character functions

- Defined in `<cctype>`
- `toupper('a')` returns the ascii of `'A'`
- `tolower('A')` returns the ascii of `'a'`
- `isupper(sym)` returns true if `sym` contains an upper case letter
- `islower(sym)` similar
- `isspace(sym)` returns true if `sym` contains a whitespace: blank, tab or newline
- `isalpha(sym)` returns true if `sym` contains a letter
- `isdigit(sym)` returns true if `sym` contains a digit

Lower-case to upper-case conversion

```
#include <iostream>  // for I/O
#include <fstream>   // for file I/O
#include <cctype>    // for character handling
using namespace std;

void main() {
   char ifile[20], ofile[20], c;
   cin >> ifile >> ofile;
   ifstream ins;
   ofstream outs;
   ins.open(ifile);
   outs.open(ofile);
   ins.get(c);
   while (!ins.eof())
   {
      if (islower(c))
         outs << char(toupper(c));
      else
         outs << c;
      ins.get(c);
   }
   ins.close();
   outs.close();
}
```

String Iterators

```
#include <iostream>
#include <string>
using namespace std;
int main () {
 string str ("Test string");
 for (string::iterator it = str.begin();
     it != str.end(); ++it)
   cout << *it;
   cout << '\n' ;
   return 0;
}
```