CSC 270 – Survey of Programming Languages

C++ Lecture 2 – Strings

Predefined Functions in `<cstring>`

Function	Description	Caution
`strcpy(s, t)`	Copies s into t	No bounds checking
`strncpy(s, t, n)`	Copies s into t but no more than n characters are copies	Not implemented in all versions of c++
`strcat(s, t)`	Concatenates t to the end of s	No bounds checking
`strncat(s, t, n)`	Concatenates t to the end of s but no more than n characters	Not implemented in all versions of c++
`strlen(s)`	Returns the length of s (not counting '\0')	
`strcmp(s, t)`	Returns 0 if s == t < 0 if s < t > 0 if s > t	No bounds checking
`strncmp(s, t, n)`	Same as strcmp but compares no more than n characters	Not implemented in all versions of c++

C-String: Input and Output

- In addition to **`cin >>`** and **`cout <<`**, there are other input and output methods available when working with strings:
  - **`getline()`**
  - **`get()`**
  - **`put()`**
  - **`putback ()`**
  - **`peek()`**
  - **`ignore()`**

`getline()`

- **`getline()`** allows the user to read in an entire line of text at a time, or no more than *n* characters:

```
char    a[80], s[5];
cout << "Enter a line:"
cin.getline(a, 80);
cout << "Enter a short word:";
cin.getline(s, 5);
cout << "\'" << a  << "\'\n\'" << s
                          << "\'" << endl;
```

- In both cases, one character less is actually read in to leave room for **`'\0'`**

`getline()` – An Example

Enter a line:

Do be do to you!

Enter a short word

Do be Do to you!

Do be Do to you!Do b

`get()`

- The function `get()` allows the user to read in every character typed, including whitespace characters.
- Use:

```
char     nextChar;
cin.get(nextSymbol);
```

- get() reads blanks and newlines as well as other characters:

```
char c1, c2, c3
cin.get(c1); cin.get(c2); cin.get(c3);
```

- If you had entered "AB\nCD", c3 would contain the newline.

CheckInput.cpp

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

void newLine(void);
// Discards all the input remaining on the current
input line.
// Also discards the '\n' at the end of the line.

void getInt(int & number);
// Sets the variable number to a
// value that the user approves of
```

```
int main(void)
{
      int n;

      getInt(n);
      cout << "Final value read in == " << n << "\n"
      << "End of demonstation." << endl;

      return(0);
}
```

```
// Uses iostream:
void newLine(void)
{
      char symbol;

      do {
            cin .get(symbol);
      } while (symbol != '\n');
}
```

```
//Uses iostream
void getInt(int &number)
{
      char ans;

      do {
            cout << "Enter input number: ";
            cin >> number;
            cout << "You entered " << number
                  << " Is that correct(yes/no): ";
            cin >> ans;
            newLine();
      } while ((ans == 'N') || (ans == 'n'));
}
```

`put()`

- **`put()`** allows the program to print a single character.
- It does not do anything that cannot be done using **`<<`**.
- Example

```
cout.put('a');
```

`putback ()`

- Sometimes your program needs to know what the next character in the input stream is going to be, but it may not be needed here. Therefore your program needs to be able to "put back" that next character.
- putback() allows your program to return a character to the input stream.

```
  if ( (c >= '0') && (c <= '9') )
  {
    cin.putback (c);
    cin >> n;
    cout << "You have entered number " << n << endl;
  }
  else
  {
    cin.putback (c);
    cin >> str;
    cout << " You have entered word " << str
         << endl;
  }

  return 0;
}
```

peek()

- peek() returns the next character in the input stream without actually removing it from the input steam – it allows you a “peek” at what comes next.

`peek()` – An Example

```
// istream peek
#include <iostream>
using namespace std;

int main () {
  char c;
  int n;
  char str[256];

  cout << "Enter a number or a word: ";
  c=cin.peek();
```

```
  if ( (c >= '0') && (c <= '9') )
  {
    cin >> n;
    cout << "You have entered number " << n << endl;
  }
  else
  {
    cin >> str;
    cout << " You have entered word " << str
         << endl;
  }

  return 0;
}
```

`ignore()`

- **`ignore()`** skips up to n characters, or until it encounters a particular character of the programmer's choosing, which ever comes first.

`ignore()` – An Example

```
// istream ignore
#include <iostream>
using namespace std;

int main () {
  char first, last;

  cout << "Enter your first and last names: ";

  first=cin.get();
  cin.ignore(256,' ');
```

```
  last=cin.get();

  cout << "Your initials are " << first << last;

  return 0;
}
```

Character-manipulating Functions

- There are several operations that you may need for basic text manipulation and are most commonly performed character by character.
- These functions have their prototypes in the cctype header file.
- Using these methods requires that

  `#include <cctype>`

  be included in the program using them

Functions in <cctype>

Function	Description	Example
`toupper(c)`	Returns the upper case version of the character	`c = toupper('a');`
`tolower(c)`	Returns the lower case version of the character	`c = tolower('A');`
`isupper(c)`	Returns true if c is an upper case letter	`if (isupper(c))` `  cout << 'upper case';`
`islower(c)`	Returns true if c is an lower case letter	`if (islower(c))` `  cout << 'lower case';`
`isalpha(c)`	Returns true if c is a letter	`if (isalpha(c))` `  cout << "it's a letter";`
`isdigit(c)`	Returns true if c is a digit (0 through 9)	`if (isalpha(c))` `  cout << "it's a number";`

Functions in <cctype> (continued)

Function	Description	Example
`isalnum(c)`	Returns true if c is alphanumeric	`if (isalnum('3'))` `  cout << "alphanumeric";`
`isspace(c)`	Returns true if c is a white space character	`while (isspace(c))` `  cin.get(c);`
`ispunct(c)`	Returns true if c is a printable character other than number, letter or white space	`if (ispunct(c))` `  cout << "punctuation";`
`isprint(c)`	Returns true if c is a printable character	
`isgraph(c)`	Returns true if c is a printable character other an white space	
`isctrl(c)`	Returns true if c is a control character	

Pitfall: toupper and tolower return int value

- In many ways, C and C++ consider characters to be 8-bit unsigned integers. For this reason, many string functions return an **`int`** value.
- Writing **`cout << toupper('a');`** will not write '**`A`**' but the numeric code that represents '**`A`**'.
- To get the desired result write

```
char c = toupper('a');
cout << c;
```

The **`string`** class

- Up until now, we have been using C-strings, which are arrays of characters ended with a null byte.
- The class **`string`** is defined in the library **`<string>`** and allows you to use strings in a somewhat more natural way.
- You can use **`=`** as an assignment operator and **`+`** as a concatenation operator.

ants.cpp

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

int   main(void)
{
      string phrase; //uninitialized

      // The following ARE BOTH initialized
      string adjective("fried"), noun("ants");
      string wish = "Bon appetite";
```

```
      // + is used for concatenation
      phrase = "I love " + adjective + " " + noun
                         + "!";
      cout << phrase << endl;
      cout << wish << endl;

      return 0;
}
```

Output

```
I love fried ants!
Bon appetite
```

I/O with string

- You can use the insertion operator **>>** and **cout** to print string objects just as you would do with any other data item.
- You can use the extraction operator **<<** and **cin** to read string objects, but **<<** will skip initial whitespace and then read only until the next whitespace character.
- If you wish to read input including the whitespace, you need to use the method **cin.get()**

motto.cpp

```
// Demonstrates getline and cin.get
#include    <iostream>
#include    <string>
using namespace std;

void  newLine();

int   main(void)
{
      string firstName, lastName, recordName;
      string motto
             = "Your records are our records.";
```

```
        cout << "Enter your first and last name:";
        cin >> firstName >> lastName;
        newLine();

        recordName = lastName + ", " + firstName;
        cout << "Your name in our records is: ";
        cout << recordName << endl;

        cout << "Our motto is\n"
              << motto << endl;
        cout << "Please suggest a better "
             << "(one line) motto:\n";
```

```
        getline(cin, motto);
        cout << "Our new motto will be:\n";
        cout << motto << endl;

        return(0);
}

// Uses iostream
void newLine(void)
{
        char nextChar;

        do    {
              cin.get(nextChar);
        } while (nextChar != '\n');
}
```

more Versions of getline

- `getline(cin, line);` will read until the newline character.
- `getline(cin, line, '?');` will read until the `'?'`.
- `getline(cin, s1) >> s2;`

  will read a line of characters into `s1` and then store the next string (up to the next whitespace) in s2.

Mixing cin << variable with getline

- Consider

```
int n;
string line;
cin >> n;
getline(cin, line);
```

  will read a value into n but nothing in line because it is holding the remainder of the line from which n's value comes for the next use of `cin`.

String Processing with **`string`**

- The string class lets you use the same operations that C-string allow and then some.
- E.g.

  `string s1;`

  `s1.length` - returns the length of the string s1.

  `lastName[i]` is the **`i`**th character in the string.

`NameArray.cpp`

```
// Demonstrates using a string object as if it were
// an array
#include    <iostream>
#include    <string>
using namespace std;

int   main(void)
{
      string firstName, lastName;

      cout << "Enter your first and last name:\n";
      cin >> firstName >> lastName;
```

```
        cout << "Your last name is spelled:\n";
        unsigned    i;
        for (i = 0;  i < lastName.length();  i++)  {
              cout << lastName[i] << " ";
              lastName[i] = '-';
        }

        cout << endl;
        for (i = 0;  i < lastName.length();  i++)
              // Places a "-" under each letter
              cout << lastName[i] << " ";
        cout << endl;
```

```
        cout << "Good day, " << firstName << endl;

        return(0);
}
```

Output

```
Enter your first and last name:
Robert Siegfried
Your last name is spelled:
S i e g f r i e d
- - - - - - - - - -
Good day, Robert
```

Member Functions of the string class

Example	Remarks
Constructors	
`string str`	Default constructor – creates empty string object **str**
`string str("string");`	Creates a string object with data **"string"**
`string str(aString);`	Creates a **string** object that is a copy of aString, (which is a **string** object)
Element Access	
`str[i]`	Returns read/write reference to character in str at index **i**
`str.at(i)`	Returns read/write reference to character in str at index **i**
`str.substr(position, length)`	Return the substring of the calling object starting at **position** and having **length** characters

Member Functions of the string class

Example	Remarks
Assignment/Modifiers	
`string str1 = str2;`	Allocates space and initializes it to **str1**'s data, releases memory allocated to **str1** and sets **str1**'s size to that of **str2**.
`str1 += str2;`	Character data of **str2** is concatenated to the end of **str1**; the size is set appropriately
`str.empty();`	Returns true if **str** is an empty string; returns false otherwise
`str1 + str2`	Returns a string that has **str2**'s data concatenated to the end of **str1**'s data. The size is set appropriately
`str.insert(pos, str2)`	Inserts **str2** into **str** beginning at position **pos**
`str.remove(pos, length)`	Removes a substring of size **length** beginning at position **pos**

Member Functions of the string class

Example	Remarks
Comparisons	
`str1 == str2` `str1 != str2;`	Compare for equality or inequality; returns a Boolean value.
`str1 < str2  str1 > str2` `str1 >= str2  str1 <= str2;`	Four comparisons. All are lexicographical comparisons
`str.find(str1)`	Returns index of the first occurrence of **`str1`** in **`str`**.
`str.find(str1, pos)`	Returns index of the first occurrence of **`str1`** in **`str`**; the search starts at position **`pos`**.
`str.find_first_of(str1, pos)`	Returns index of the first instance of any character in **`str1`**; the search starts at position **`pos`**.
`str.find_first_not_of(pos, length)`	Returns index of the first instance of any character ***not*** in **`str1`**; the search starts at position **`pos`**

palindrome.cpp

```
// Test for palindrome property
#include    <iostream>
#include <string>
#include <cctype>
using namespace std;

// Interchanges the values of v1 and v2
void swap(char &v1, char &v2);

// Returns a copy of s but with characters in
// reverse order
string reverse(const string &s);
```

```
// Returns a copy of s with any occurrences of
// characters in the string punct removed.
string removePunct(const string &s,
                   const string &punct);

// Returns a copy of s that has all uppercase
// characters changed to lowercase, with other
// characters unchanged
string makeLower(const string &s);

// Returns true if s is a palindrome;
//         false otherwise
bool isPal(const string &s);
```

```
int main(void)  {
      string str;

      cout << "Enter a candidate for palindrome "
           << "test followed by press Return."
           << endl;
      getline(cin, str);

      if (isPal(str))
            cout << "\"" << str
                + "\" is a palindrome." << endl;
      else
            cout << "\"" << str
                + "\" is not a palindrome." << endl;
```

```
        cin >> str;
        return(0);
}

void swap(char &v1, char &v2) {
        char temp = v1;
        v1 = v2;
        v2 = temp;
}
```

```
string reverse(const string &s)  {
        int start = 0;
        int end = s.length();
        string temp(s);

        while (start < end) {
                --end;
                swap(temp[start], temp[end]);
                start++;
        }
        return temp;
}
```

```
// Uses <cctype> and <string>
string makeLower(const string &s) {
      string temp(s);

      for (int i = 0;  i < s.length();  i++)
            temp[i] = tolower(s[i]);

      return temp;
}
```

```
string removePunct(const string &s,
                   const string &punct) {
      string noPunct; //Initialized to empty string
      int sLength = s.length();
      int   punctLength = punct.length();

      for (int i = 0;  i < sLength;  i++)  {
            // A one-character string
            string aChar = s.substr(i, 1);

            // Find location of successive
            // characters of src in punct
            int location = punct.find(aChar, 0);
```

```
            //  aChar is not in punct, so keep it
            if (location < 0 ||
                    location >= punctLength)
                noPunct = noPunct + aChar;
      }
      return noPunct;
}
```

```
// Uses functions makeLower, removePunct
bool isPal(const string &s)  {
      string punct(",;:.?!'\" "); // includes a
blank
      string str(s);
      str = makeLower(str);
      string lowerStr = removePunct(str, punct);

      return (lowerStr == reverse(lowerStr));
}
```

Converting string objects and C-Strings

```
//Legal
char aCString[] = "This is my C-string.";
string stringVariable;
stringVariable = aCString;

//ILLEGAL
aCString = stringVariable;
Strcpy(ACString, stringVariable);

//Legal
Strcpy(aCString, stringVariable.c_str());

//ILLEGAL
aCString = stringVAriable.c_str();
```