CSIS/COMP 1117B
Computer Programming

Strings and Vectors

Strings

- A **string** is a sequence of characters.
- There are two ways to represent strings in C++: C-style strings and the string class.
- C-style strings are NULL (‘\0’) terminated char arrays
  - note that a char array of size *n* can only store a string of at most *n* – 1 characters; for example:

    char x[6] = { ‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};

    char x[6] = “Hello”; // equivalent to the above
  - a literal (constant) string is stored in a const char array; for example, “No way” is stored as:

N	o		w	a	y	\0

  - operations on C-style strings are only those that are appropriate for char arrays

The cstring Library

- Some operations on C-style strings are available in the **cstring** library:

  #include <cstring> // no need to use std namespace
- **Assignment**: strcpy(t, s) copies the C-style string value s to the C-style string variable t; no check on whether or not t is large enough to store s.
- **Concatenation**: strcat(t, s) concatenates the C-style string value s to the end of the C-style string variable t; no check on whether or not t is large enough to hold the result of the concatenation.
- **Compare**: strcmp(t, s) returns 0 if t and s are the same, returns a value < 0 if t is less than s, returns a value > 0 if t is greater than s; the order is lexicographic. ***(Note that the return value for s equals t represents false!)***

Input of C-style Strings

- The getline method of the cin object can be used to read in ***one line*** of characters from a keyboard into a given char array ***without skipping*** any white space characters, e.g.,

```
char        s[100];
cin.getline(s, 100);   // up to 99 chars will be read
```

- Input stream objects also has getline as a member function:

```
ifstream  inf;
inf.getline(s, 80);     // only read up to 79 characters
```

Command Line Arguments

- In a non-window based environment, we run a program by typing its name on a command line, for example:

  rename old new // rename file old to new

- For convenience, sometimes we would like to provide input values to a program in the same command line; for example:

  copy file1 file2 // create a copy of file1 in file2

  – it is possible to modify our file copying program to accept filenames in the command line

The Copy File Program Again

```
#include  <fstream>
#include  <iostream>
#include  <cstdlib>
using namespace std;
// program invoked as:  argv[0]  argv[1]  argv[2]  . . .  argv[argc]
int main(int argc, char *argv[ ]) {
    ifstream   in;
    ofstream  out;
    if (argc < 2) {
        cout << argv[0] << “: Too few arguments\n”;
        exit(-1)
    }
    setup_source(in, argv[1]);
    setup_destination(out, argv[2]);
    copy(in, out));
    in.close();
    out.close();
    exit(0);
}
```

Setup File Streams

```
// setup source input stream
void setup_source(ifstream &in, char f[ ]) {
    in.open(f);
    if (in.fail()) {
        cout << “Cannot open \”” << f << “\”.\n”;
        exit(-1);
    }
}
// setup destination output stream
void setup_destination(ofstream &out, char f[ ]) {
    out.open(f);
    if (out.fail()) {
        cout << “Cannot open \”” << f “\”.\n”;
        exit(-1);
    }
}
```

Copying C-style Strings

- Many operations on C-style strings relies on the fact that there is a terminating ‘\0’:

```
void c_str_copy(char a[ ], char b[ ]) {
    for (int k = 0; a[k] = b[k]; k++) ;
}
```

- If for some reasons the ‘\0’ in b got changed to something else; the function above could cause a lot of damage to the data area since C++ does not perform index checking, a safer version:

```
void c_str_ncopy(char a[ ], char b[ ], int n) {
    for (int k = 0; a[k] = b[k]; k++)
        if (k >= n) return;
}
```

The string Class

- An object of the **string class** contains a char array (for storing the characters that make up the string) as well as functions and operators for manipulating strings.
- To use the string class:

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

- Declaration and initialization of string objects:

```
// variables are initialized to 0 by default
string phrase, phrase2, name;    // initialize to empty string
string last_name = “Chan”;
string first_name(“Tai Man”);
```

Common string Operations (1)

- **Assignment**: a character, a literal string, a C-style string or the value of another string variable can be assigned to a string variable:

```
name = 'I';            // automatic casting, char -> C-style string
phrase = " like  ";    // automatic casting C-style string -> string
phrase2 = phrase;
```

- **Concatenation** (+): a string variable and a character, or a literal string, or a C-style string, or another string variable can be *joined together* to form a new string:

```
name + phrase          // result -> "I like  "
'I' + phrase + "HKU"   // result -> "I like  HKU"
phrase + phrase        // result -> " like   like  "
```

Common string Operations (2)

- **Comparison**: the 6 relational operators are applicable to a pair of string variables or a string variable and a literal string (comparison is based on lexicographical order):

  phrase == phrase2 // true since they are identical

  last_name > “Cheng” // false since ‘e’ > ‘a’

- Note that ***at least one*** of the operands of the comparison operators and the concatenation operator must be a ***string variable***, they are not applicable to a pair of C-strings, or to a pair of literal strings, or to a C-string and a literal string:

  “no “ + “way” // illegal

  “high” < “low” // illegal

Operations on a string Object (1)

- The number of characters in a string s (excluding the trailing '\0') can be retrieved by calling s.length()
- The *i*th character of a string s can be referred to using the subscript operator [ ], e.g., s[i], where $0 \leq i < s.length()$

```
string s("CS HKU");     // s.length() -> 6
cout << s[1] << endl;   // prints s
s[1] = 'E';             // s[1] is a char variable
cout << s << endl;      // prints CE HKU
```

- s.at(i) is similar to s[i] except that range check is applied to i

Operations on a string Object (2)

- Suppose s is a string variable

  string s = “abcdefgh”;
- s.empty() checks whether or not s is an empty string; it returns a bool value
- s.find(t) returns the index of the first occurrence of string t in s; if not found, it returns the constant string::npos, e.g., s.find(“ef”) returns 4
- s.substr(pos,n) returns the substring of s of length n starting at position pos; e.g., s.substr(2,4) returns a string with the value “cdef”
- s.insert(pos,t) inserts string t into s starting at pos; e.g., s.insert(3,”123”) changes the value of s to “abc123defgh”

Converting a string Object to a C-string

- A string object is ***not compatible*** with a C-style string, in place where a C-style string is needed, s.c_str() can be used to obtain an equivalent string literal of a string object s; e.g.,

```
ifstream  ifs;
string    s = “score.txt”;
ifs.open(s);            // wrong! open expects a char[ ]
ifs.open(s.c_str());  // correct
```

Input/Output of string Objects

- The insertion operator (<<) can be used to output the value of a string object, e.g.,

  cout << last_name << first_name << " loves HKU\n";
- Reading a string using the extraction operator (>>) will ***stop before*** a white space character.
- The preferred method to read strings from a keyboard is to use getline (which will also remove the end of line or end of file from the input stream):

```
istream      in;
ifstream     inf;
string       s;
getline(in, s);       // input a line from keyboard to s
getline(inf, s);      // input a line from file object inf to s
getline(in, s, '\t'); // input ends on a '\t'
```

Caution on mixing getline and >>

- If getline is preceded by the extractor operator (>>), some white space character or an end of line character may still be in the input stream and that character needs to be ***skipped explicitly*** using the member function ignore:

```
// stops on white space characters
cin >> some_variable;
// skips up to 1000 characters or next '\n'
cin.ignore(1000, '\n');
```

- ***Character input always has to be done with the utmost care!***

Vectors

- Vectors can be thought of as arrays that can grow (and shrink) in length during program execution; to use vectors:

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

- Vector declaration:

```
// an int vector of 100 elements, all elements initialized to 0
vector<int> v(100);   // vector is a template class
// a char vector of 20 elements, all elements initialized to 0
vector<char> c(20);
vector<double> d;    // a double vector without any entries
```

- Individual elements of a vector can be referenced using the square bracket operator ([ ]) just like array elements, e.g., v[2] is the 3rd member of v.
- The member function push_back can be used to add an element to the next available position of a vector, e.g., the following code adds 3 elements to d:

```
d.push_back(0.0); d.push_back(1.0); d.push_back(2.0);
```

Size and Capacity

- The **size** (number of elements) of a vector can be retrieved using the member function size which returns an ***unsigned int***, e.g.,

  for (unsigned i = 0; i < d.size(); i++) . . .
- Storage is allocated to a vector in blocks so that more then *size* elements of storage is actually available in a vector; the actual storage capacity of a vector can be retrieved using the member function **capacity**, e.g., d.capacity()
- The system can be requested to allocate at least a certain number of elements to a vector by using the member function reserve, e.g.,

  d.reserve(32); // sets the capacity of d to at least 32 elements
- The size of a vector can be changed using the member function **resize**, e.g., d.resize(24)
  - if the new size is larger than the current size, the new elements will be initialized to 0
  - if the new size is smaller than the current size, the extra elements will be removed

String as Vector of char

- Suppose we try to implement strings as vectors of character.

```
#include  <vector>
using namespace std;
typedef  vector<char> alt_string;
void alt_strcat(alt_string &t, alt_string s) {
    for (unsigned k = 0; k < s.size(); k++ )
        t.push_back(s[k]);
};
void alt_strcpy(alt_string &t, alt_string s) {
     t.resize(0);
     if (s.size() > 0) alt_strcat(t, s);
};
int alt_strcmp(alt_string t, alt_string s) {
    unsigned m = t.size();
    if (m > s.size()) m = s.size;
    for (unsigned k = 0; k < m; k++)
        if (t[k] > s[k]) return 1;
        else if (t[k] < s[k]) return -1;
    if (t.size == s.size) return 0;
    else if (t.size > s.size) return 1;
    else return -1;
}
```