COMP2004 Programming Practice 2002 Summer School

Kevin Pulo
School of Information Technologies
University of Sydney

Unsigned integers

- Unsigned integers are always >= 0
- Prefix with unsigned
- Useful for things like string lengths
- Allows a larger (positive) range
- Eg. assuming 2 byte (16 bit) ints:
  - int: -32768 to 32767
  - unsigned int: 0 to 65535

sizeof

- Number of bytes used by int, long, etc isn't guaranteed
- Use the sizeof() operator to find out their actual sizes
  - sizeof(int)
  - long l;
  - sizeof(l)
- Can omit parenthesis on variables
  - sizeof l

Standard sizes

- The C/C++ standard specifies only minimum sizes
  - sizeof(char) >= 1
  - sizeof(short) >= 2
  - sizeof(int) >= 2
  - sizeof(long) >= 4
  - sizeof(float) >= 2
  - sizeof(double) >= 4
  - sizeof(long double) >= 4

Static casting

- Used to convert between types
  - Usually numeric types
- In Java:
  - double d = (int)(2.0/3.0) + 0.2;
- In C++, use the builtin static_cast<>() function:

```
double d;
d = static_cast<int>(2.0/3.0) + 0.2;
```

Mathematical functions

- #include <math.h> to use
- Sometimes need -lm when compiling
  g++ -Wall -g -o hello hello.cc -lm
- Functions:
  - sqrt(), abs(), sin(), cos(), tan(), log(), log10(), exp(), pow(), etc
- Majority work with double variables
- For more info, see the math.h section of the "C++ Library Reference"
  - "Useful Resources" on PP website

std::string

- Text is used in a lot of programs
  - Input and output is often text
  - Data is often text
  - Manipulating text is common
- std::string objects are used here
- #include <string> to use them
- String literals are strange
  - Not the same as std::string
  - Can often be used, but not always

Constructing Strings

- std::string s; - the empty string
- std::string s = "abc"; - obvious
- std::string s("def"); - also obvious
- std::string s(6, '-'); - the string "------"
- std::string s = s1; - copy of s1
- std::string s(s1, 2, 3); - substring of s1

Accessing Elements

- Subscripts
  - char c = s[5]; // get 6th char
  - s[5] = 'a'; // set 6th char
  - Not bounds checked
- at() member function
  - char c = s.at(5); // get 6th char
  - s.at(5) = 'a'; // set 6th char
  - Bounds checked
- s.size() and s.length()
  - Length of string

Converting to C-style strings

- #include <stdlib.h> has some useful functions which use char * variables
- To convert a std::string to char *, use the c_str() method:

```
#include <stdlib.h>
std::string s = "52241";
long l = atol(s.c_str());
```

Comparisons

- s1.compare(s2) returns an int
  - < 0 if s1 < s2
  - > 0 if s1 > s2
  - = 0 if s1 equal to s2
- Can also use <, <=, ==, >=, >, !=
  - eg. s1 == s2
  - Don't do "abc" == "abc"
    - Will compile, but doesn't make sense
    - std::string("abc") == std::string("abc")

Substrings

- Defined by start and length
- s1.substr(0, 5)
  - first five character substring

```
std::string s = "hello, how are you?";
std::cout << s.substr(7, 3) << '\n';
```

- Returns a newly created std::string

Append and Concatenate

- Append string s2 to s1
  - s1 += s2;
  - s1.append(s2);
- Append character 'a' to s
  - s += 'a';
  - s.append('a');
- Concatenate
  - s3 = s1 + s2;
  - s3 = 'a' + s1;

Insertion

- Insert before given index

```
std::string s1 = "abcghi";
std::string s2 = "def";
s1.insert(3, s2);
s1.insert(5, 2, '*');
std::cout << s1; // output abcde**fghi
```

Searching

- Returns index of start of match
- s1.find(s2);
  - first occurance of s2 in s1
- s1.rfind(s2);
  - last occurance of s2 in s1

```
std::string s1 = "abcdef---defghi";
std::string s2 = "def";
s1.find(s2); // returns 3
s1.rfind(s2); // returns 9
```

Searching

- s1.find_first_of(s2);
  - first element of s1 also in s2
- s1.find_last_of(s2);
  - last element of s1 also in s2

```
std::string s1 = "abcdef---defghi";
std::string s2 = "gec";
s1.find_first_of(s2); // returns 2
s1.find_last_of(s2); // returns 12
```

Searching

- s1.find_first_not_of(s2);
  - first element of s1 not in s2
- s1.find_last_not_of(s2);
  - last element of s1 not in s2

```
std::string s1 = "abcdef---defghi";
std::string s2 = "ghiabc";
s1.find_first_of(s2); // returns 3
s1.find_last_of(s2); // returns 11
```

Replace

- Specify what to replace and what to replace with
- Replacement does not have to be the same length
- s1.replace(0, 5, s2);
  - replace first five characters with s2
- s1.replace(3, 1, 10, '*');
  - replace fourth character with 10 *s

std::vector

- #include <vector> to use
- We have to specify the type to store
- Elements added with push_back()
- Elements accessed like std::string

```
std::vector<int> vi;
std::vector<char> vc;
vi.push_back(1);  vi.push_back(10);
vc.push_back('a');  vc.push_back('z');
std::cout << vi.at(1) << vc[0] << '\n';
```

Constructing Vectors

- std::vector<double> v;
  - empty vector
- std::vector<std::string> v(10);
  - vector with 10 default strings
- std::vector<char> v(10, '-');
  - vector with 10 '-' elements

Accessing Elements

- As for strings
  - v.at(4) - bounds checking
  - v[4] - no bounds checking
- v.front() - returns first element
- v.back() - returns last element
- v.push_back(val) - add val to end
- v.pop_back() - remove last element
  - undefined on empty vector
- v.size() - returns size of vector

Special types

- Strings and vectors provide types
- std::string::size_type
  - Integral type to use for indexes
  - eg. pass to at(), returned from find()
- std::string::difference_type
  - Integral type for element distances
- Similarly
  - std::vector<T>::size_type
  - std::vector<T>::difference_type

Special types usage

```
#include <string>
#include <iostream>
int main() {
   std::string s;
   std::cin >> s;
   std::string::size_type a = s.find('a');
   std::string::size_type e = s.find('e');
   std::string::difference_type d = e - a;
   std::cout << d << '\n';
}
```

Typedefs

- User defined type names
- Provides an "alias" for existing types
- typedef new_type existing_type;

```
typedef std::string::size_type str_sz;
typedef std::string::difference_type str_df;
str_sz a = s.find('a');
str_sz e = s.find('e');
str_df d = e - a;
```

No Match?

- What happens if the string had no 'a'?
- s.find('a') would return std::string::npos
- npos is a std::string::size_type value
- It is larger than the largest possible string
- It is used as a not found value
- And to indicate "rest of string" for substr(), replace(), etc

Other data structures

- The STL provides several other useful data structures
  - stack, list, map, set, queue, deque, priority_queue
- They all work in much the same way
- Use an STL reference to get full details

std::stack

```
#include <stack>
int main() {
   std::stack<int> s;
   for (int i = 0; i < 10; i++)
      s.push(i);
   while ( ! s.empty()) {
      std::cout << s.top() << stl::endl;
      s.pop();
   }
}
```

- Also has s.size()

Pass by Value

- C++ passes arguments by value

```
int sum(std::vector<int> v) {
   int result = 0;
   for (std::vector<int>::size_type i = 0;
                  i != v.size(); ++i)
      result += v[i];
   return result;
}
```

- Arguments can be safely modified
- Inefficient due to copying

Pass by Value

```
void pass_by_value(int i) {
   i = 4;
}

int main() {
   int j = 1;
   pass_by_value(j);
   std::cout << j << std::endl;
   // outputs 1, not 4
}
```

Pass by Reference

- Reduces copying

```
int sum(std::vector<int> &v) {
   int result = 0;
   for (std::vector<int>::size_type i = 0;
                  i != v.size(); ++i)
      result += v[i];
   return result;
}
```

- Modifying argument also modifies the original

Pass by Reference

```
void pass_by_ref(int &i) {
   i = 4;
}

int main() {
   int j = 1;
   pass_by_ref(j);
   std::cout << j << std::endl;
   // outputs 4, not 1
}
```

Pass by Const Reference

- Disallows modification of argument
- The usual way of passing large objects

```
int sum(const std::vector<int> &v) {
   int result = 0;
   for (std::vector<int>::size_type i = 0;
                i != v.size(); ++i)
      result += v[i];
   return result;
}
```

Pass by Const Reference

```
void pass_by_const_ref(const int &i) {
   // compilation error on following line
   i = 4;
}

int main() {
   int j = 1;
   pass_by_const_ref(j);
   std::cout << j << std::endl;
}
```

A Simple Example

- A program that takes marks
- And outputs a histogram

```
#include <string>
#include <vector>
#include <iostream>
#include <iomanip>

// width in characters of histogram bars
const int bar_width = 20;

// the count of marks in each decile
std::vector<int> marks(10, 0);
```

```
// find maximum in vector
int find_max(const std::vector<int> &v) {
   int max = 0;
   for (int i = 0 ; i < v.size() ; ++i)
         if (max < v[i])
               max = v[i];
   return max;
}
```

```
// read in the marks
int mark;
while (std::cin >> mark) {
      int decile = mark / 10;
      if (decile > 9)
            decile = 9;
      marks[decile]++;
}

// get maximum decile count
max_len = find_max(marks);
```

```
// output the histogram
for (int i = 0; i < 10 ; ++i) {
      std::cout << std::setw(2) << i*10;
      std::cout << '-' << std::setw(2);
      if (i != 9)
            std::cout << i*10 + 9;
      else
            std::cout << "00";
      std::cout << ' ' << std::string(
         marks[i] * bar_width / max_len,
         '#') << std::endl;
}
```

Sample Output

```
 0- 9 ##########
10-19 #####
20-29 #####
30-39 ####
40-49 ########
50-59 ##########
60-69 ###############
70-79 ####################
80-89 ####################
90-00 ###############
```