# Question #1 from Quiz 24

To copy an integer array 'arr' of size 5 to a vector L, in reverse order

□

```
vector<int> L;
for(int i=0; i<5; i++)
    L.push_back(arr[i]);
```

□

```
vector<int> L(5);
for(int i=4; i>=0; i--)
  L.push_back(arr[i]);
```

☑

```
vector<int> L;
for(int i=4; i>=0; i--)
  L.push_back(arr[i]);
```

right to left scan

arr[i]

keeps growing by appending to end

□

```
vector<int> L(5);
for(int i=0; i<5; i++)
  L[i] = arr[i];
```

☑

```
vector<int> L(5);
for(int i=0; i<5; i++)
  L[i] = arr[4-i];
```

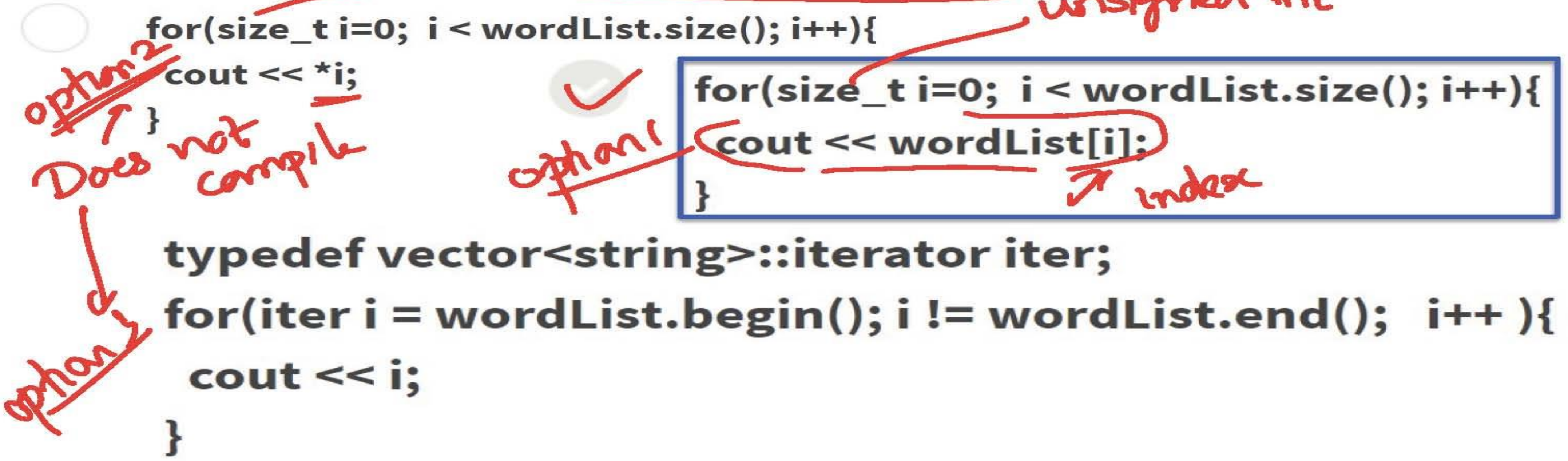
left to right arr scan

L[i] - - arr[4-i]

# Question #2 from Quiz 24

```
typedef vector<string>::iterator iter; // shorthand
for(iter i = wordList.begin(); i != wordList.end();  i++ ){
 cout << *i;   // *  gives value
}
```

wordList is of type vector<string>:
What is equivalent to the above iterative code?



○

```
for(size_t i=0;  i < wordList.size(); i++){
  cout << *i;
}
```

option2

Does not compile

✓

```
for(size_t i=0;  i < wordList.size(); i++){
 cout << wordList[i];
}
```

option1

unsigned int

index

```
typedef vector<string>::iterator iter;
for(iter i = wordList.begin(); i != wordList.end();   i++ ){
  cout << i;
}
```

option3

# Question #3 from Quiz 23

vector<string>  wordList

searchWord="CS101"

```
vector<string>::iterator where = find(wordList.begin(), wordList.end(),searchWord);
bool found = ( where != wordList.end() );
if(found) cout << "Found " ;
```

in #include algorithm

Linear scan

option 1

```
sort(wordList.begin(), wordList.end());
bool found= binary_search(wordList.begin(), wordList.end(),searchWord);
if(found) cout << "Found " ;
```

✓ option 3

Always binary search on sorted list (<)

option 2

**AND NOT……..** (Binary search on unsorted list)

```
bool found= binary_search(wordList.begin(), wordList.end(),searchWord);
if(found) cout << "Found " ;
```

# CS 101: Computer Programming and Utilization

Ganesh Ramakrishnan
with
CS101 TAs and Staff
(cs101-help@cse.iitb.ac.in)
(cs101-2019a@googlegroups.com)

Course webpage: http://www.cse.iitb.ac.in/~cs101/

## Lecture 27: Standard Library: Map (concluded)

# The Map Template Class

- A vector or an array give us an element when we supply an index
  - Index must be an integer
- But sometimes we may want to use indices which are not integers, but strings
  - Given the name of a country, we may want to find out its population, or its capital
  - This can be done using a map

index can be any object

map<indextype, valuetype>

# Recap: Template Class with Multiple types

```
template <class T, class M>
class Queue {
  int front, nWaiting;
  T elements[100];
  M customers[100];
public:
  bool insert(T value) {...}
  bool remove(T &val, M cust) {...}
};
```

Managing both Driver details & Customer details through class template

**Similarly:** **map<indextype, valuetype> mapname;**

**Example:** **map<string, float> name2cs101Marks;**

# Map: General Form And Examples

- General form:

  ```
  map<indextype, valuetype> mapname;
  ```

- Examples:

  ```
  map<string,double> population;
  ```

  Running example

  Indices will have type string (country names), and elements will have type double (population)

  ```
  map<string, vector<string>> dictionary;
  ```

  Maps words to their meanings expressed as a vector of other words.

  dictionary["cs101"] = ["  .  ", "  .  ", "  .  "]

# Using A Map

```
map<string,double> population;

population[“India”] = 1.35;
                    // in billions.
population[“China”] = 1.41;
population[“USA”] = 0.32;
```

# Using A Map

```
map<string,double> population;

population[“India”] = 1.35;
                    // in billions.  Map entry created
population[“China”] = 1.41; // Map entry created
population[“USA”] = 0.32; // Map entry created

cout << population[“China”] << endl;
                    // will print 1.41
```

expected

# Using A Map

```
map<string,double> population;

population[“India”] = 1.35;
                    // in billions.  Map entry created
population[“China”] = 1.41; // Map entry created
population[“USA”] = 0.32; // Map entry created

cout << population[“China”] << endl;
                    // will print 1.41

population[“India”] = 1.36;
                    //update allowed
```

Expected behaviour

1.35 overwritten

# Using A Map

```
map<string,double> population;

population[“India”] = 1.35;
                  // in billions.  Map entry created
population[“China”] = 1.41; // Map entry created
population[“USA”] = 0.32; // Map entry created

cout << population[“China”] << endl;
                  // will print 1.41

cout << population["Nepal"] << endl;
                  //what will it print?
```

# Using A Map

```
map<string,double> population;

population[“India”] = 1.35;
                    // in billions.  Map entry created
population[“China”] = 1.41; // Map entry created
population[“USA”] = 0.32; // Map entry created

cout << population[“China”] << endl;
                    // will print 1.41

cout << population["Nepal"] << endl;
                    //what will it print? update indirectly made
```

in this case:
Step1: update to some default 0
Step2: print it

# Checking if An Index is Defined

```
string country;
cout << “Give country name: “;
cin >> country;

if(population.count(country)>0)
            // true if element with index = country
            // was stored earlier
            // count is a known member function
 cout << population[country] << endl;
else cout << “Not known.\n”;
```

count = 0 or 1 if defined

if not yet defined

# Remarks

- A lot goes on behind the scenes to implement a map
- Basic idea is discussed in Chapter 24 of our book
- If you wish to print all entries stored in a map, you will need

  to use iterators, discussed next

Proposal 1 : Vectors for each of Key & value

Proposal 2 : T* (pointer to sequence of instances of type T)
or dynamically allocated arrays on heap

Proposal 3 :- Sorted vector/array of keyvalues for efficient lookup / search
(Verifiable)

# Iterators (again)

- A map can be thought of as holding a sequence of pairs, of the form (index, value)
- For example, the population map can be considered to be the sequence of pairs

  [("China",1.41), ("India",1.35), ("USA", 0.32)]
- You may wish to access all elements in the map, one after another, and do something with them
- For this, you can obtain an iterator, which points to (in an abstract sense) elements of the sequence

(key, value)

first second

# Iterators for Maps

- An iterator for a map<index,value> is an object with type map<index,value>::iterator
- An iterator points to elements in the map; each element is a struct with members first and second
- We can get to the members by using dereferencing

(*mi).first

(*mi).second

- Note that this simply means that the dereferencing operators are defined for iterators
- If many elements are stored in an iterator, they are arranged in (lexicographically) increasing order of the key

Recall Proposal 3 for implementing map

# Example

```
map<string,double> population;
population[“India”] = 1.35;

map<string,double>::iterator mi;
mi = population.begin();
            // population.begin() : constant iterator
            // points to the first element of population
            // mi points to (India,1.35)
cout << mi->first << endl; // or (*mi).first << endl;
            // will print out India
cout << mi->second << endl;
            // will print out 1.35
```

# Example

```
map<string,double> population;
population["India"] = 1.35;
population["China"] = 1.41;
population["USA"] = 0.32;
for(map<string,double>::iterator
    mi = population.begin();
    mi != population.end();
                    // population.end() : constant iterator
                    // marking the end of population
     mi++)
                     // ++ just sets mi to point to the
                     // next element of the map
     // loop body
```

loops around the three entries

# Example (Contd.)

```
map<string,double> population;
population["India"] = 1.35;
population["China"] = 1.41;
population["USA"] = 0.32;
for(map<string,double>::iterator
     mi = population.begin();
     mi != population.end();
     mi++)
{
   cout << mi→first << " , " << mi→second;

         China, 1.41        India, 1.35   USA, 0.32
}
// will print out countries and population in alphabetical order
```

expect alphabetically sorted order of iteration

# Example (Contd.)

```
map<string,double> population;
population[“India”] = 1.35;
population[“China”] = 1.41;
population[“USA”] = 0.32;
for(map<string,double>::iterator
    mi = population.begin();
    mi != population.end();
    mi++)
{
 cout << (*mi).first << “: “ << (*mi).second << endl;
 // or cout << mi->first << “: “ << mi->second << endl;
}
// will print out countries and population in alphabetical order
```

# Remarks

- Iterators can work with vectors and arrays too
- Iterators can be used to find and delete elements from maps and vectors.

```
map<string,double>::iterator
        mi = population.find("India");
population.erase(mi);
```

Entry for key= India will vanish from map hereafter.

# Map with user-defined class as index

- Any class used as indextype on a map must implement the < operator

V3 V3

0, 0, 1 < 0, 0, 2

0, 1, 1 < 0, 2, 3

# Map with user-defined class as index

- Any class used as indextype on a map must implement the "<" operator.
- Example, the following code will not work because "<" is not defined on V3.
  - class V3 {public: double x,y,z};
  - map<V3, string> vec2string;
- A correct implementation of V3 may be something like:

```
class V3 {
    public:
    double x,y,z;
    bool operator<(const V3& a) const {
        if (x < a.x) return true;
        if (x == a.x && y < a.y) return true;
        if (x==a.x && y == a.y && z < a.z) return true;
        return false;
    }
};
```

# CS 101: Computer Programming and Utilization

Ganesh Ramakrishnan
with
CS101 TAs and Staff
(cs101-help@cse.iitb.ac.in)
(cs101-2019a@googlegroups.com)

Course webpage: http://www.cse.iitb.ac.in/~cs101/

## Lecture 27: Program Organization and Functions (Not part of endsem)

# About These Slides

- Based on Chapter 11 of the book

*An Introduction to Programming Through C++*

by Abhiram Ranade (Tata McGraw Hill, 2014)

functions & classes as use cases

- Original slides by Abhiram Ranade

– First update by Sunita Sarawagi

– Second update by Ganesh Ramakrishnan

# A different role for functions

- We said that a function should be created if you find yourself writing code to perform the same action at different places in the program.
- However, functions have a different role too: A function is an “organizational/logical unit” of a program.

# Physical units of code: files

- If several people write different functions of the same program, it is more convenient if each uses a different file. gcd.cpp, lcm.cpp
- We need ways by which functions (as also classes/structs) in one file can call functions in other files

Declaration

# Outline

Functions and program organization

- The main program is a function
- How to split a program into many files
  - Function declarations
  - Separate compilation
  - Header files
- Namespaces
- Using C++ without simplecpp

But link files for execution

# Splitting a program into many files

- A program may contain several functions. All need not be placed in the same file.

lcm.cpp gcd.cpp

- If code in file F calls a function f, then function gcd f must be declared inside F, textually before any call to it.

lcm.cpp

- A function definition is a declaration, but there can be other ways to declare.
- Every function must be defined in just one of the files that are used for a program.

# Function declaration

- A function declaration is the definition without the body.
  - The return type, name, parameter types and optionally parameter names.
- Example: declaration of gcd function:

```
int gcd(int m, int n);
int gcd(int, int);    // also acceptable.
```

- The declaration tells the compiler that if gcd appears later, it will be a function and take 2 ints as arguments.
  - This helps the compiler to translate your program into machine language, without needing to look up the definition of gcd.
- If a file calls a function but contains only a declaration of it; it cannot be completely compiled to enable execution.
  - Whatever is in it, is compiled, and the result is called an object module.
  - To get an executable programs, all the object modules containing all called functions must be linked together.

# Separate compilation

- File gcd.cpp

```
int gcd(int m, int n){ … }
```

- File lcm.cpp

```
int gcd(int, int);
int lcm(int m, int n){
     return m*n/gcd(m,n);}
```

- File main.cpp

```
int lcm(int, int);
int main(){
cout << lcm(36,24) << endl;
}
```

- function definitions
- function declarations
- As you can see, each file contains a declaration of the function that is called in it.
- You may compile and link all files together by giving

```
g++ main.cpp lcm.cpp gcd.cpp
```

But requires names of all 3 files

- You may compile each file separately, e.g. by giving

```
g++ -c main.cpp
```

- -c will ask compiler to produce main.o (object module).
- Object modules can be linked together to get an executable by typing

```
g++ main.o lcm.o gcd.o
```

combined effect

a.exe created on windows

On linux typically a.out

# Header files

- Tedious to remember what declaration to include in each file.
- Instead, put all declarations in a header file, and "include" the header file into every file.
- Header files have suffix .h or .hpp., or no suffix.
- The directive "#include filename" is used to include files. It is simply replaced by the content of the named file.
- OK to declare functions that do not get used.
- OK to have both a declaration and then the definition of a function in the same file.
- If header file is mentioned in " ", it is picked up from the current directory.
- If it is mentioned in < >, it is picked up from some standard place, e.g. simplecpp

- File gcdlcm.h

```
int gcd(int, int);
int lcm(int,int);
```

- File gcd.cpp

```
#include "gcdlcm.h"
int gcd(int m, int n){ … }
```

- File lcm.cpp

```
#include "gcdlcm.h"
int lcm(int m, int n){ … }
```

- File main.cpp

```
#include <simplecpp>
#include "gcdlcm.h"
int main(){
cout << lcm(36,24) << endl;
}
```

All declarations

The #include <file.h> gets replaced by declarations in file.h

# Header files for classes

- Typically, separate file for each large class with the same name.
- Header file declares the entire class but skips definition of large functions that are declared in a .cpp file
- Includes similar to other header files.

- **File queue.hpp**

```
class Queue {
   private:
     // declare private data members.
   public:
      // declare, not define large functions
      bool insert(int driver);
      ...
 }
```

declarations in .hpp

- **File queue.cpp**

```
   #include "queue.hpp"
   bool Queue::insert(int driver) {...}
   ...
```

definitions in .cpp

- **File main.cpp**

```
#include "queue.hpp"
int main() {
  Queue q; .... }
```

# Concluding Remarks

- Functions are building blocks of programs.
- Functions can be put into many files, provided each file contains a declaration before the use.
- Declarations go into header files.
- Details discussed in the book.