Preparing for Final Exam

Chapters

Chapter 2 Computers, People, and Programming

- Hello World program

```
#include<iostream>

using std::cout;

int main()
{  // where a C++ programs start
   cout << "Hello, world\n";

   return 0;   // return success
}
```

Chapter 2 Computers, People, and Programming

- Hello World program
- Compilation
- Linking
- Programming environments
- Integrated Development Environment (IDE)

```
#include<iostream>

using std::cout;

int main()
{  // where a C++ programs start
     cout << "Hello, world\n";

     return 0;   // return success
}
```

Chapter 2 Computers, People, and Programming

- Hello World program
- Compilation
- Linking
- Programming environments
- Integrated Development Environment (IDE)

Questions to note:

(a) Name the four parts of a function

Chapter 2 Computers, People, and Programming

- Hello World program
- Compilation
- Linking
- Programming environments
- Integrated Development Environment (IDE)

Questions to note:

(a) Name the four parts of a function

- A return type
- A name
- A parameter list
- A function body

Chapter 2 Computers, People, and Programming

- Hello World program
- Compilation
- Linking
- Programming environments
- Integrated Development Environment (IDE)

Questions to note:

(b) Name a function that must appear in every C++ program

Chapter 2 Computers, People, and Programming

- Hello World program
- Compilation
- Linking
- Programming environments
- Integrated Development Environment (IDE)

Questions to note:

(b) Name a function that must appear in every C++ program

function **main**

Chapter 2 Computers, People, and Programming

- Hello World program
- Compilation
- Linking
- Programming environments
- Integrated Development Environment (IDE)

To-do:

Look through other questions of the Review part of the Chapter and be ready to answer similar questions.

Chapter 3: Input and Type

- Builtin types:
  - int, double, bool, char
- Library types: string, vector
- Input and output
- Operators—“overloading”
- Variable names in C++
- Simple computations
- Literals
- Declaration & initialization
- Type safety

```
// inch to cm and cm to inch conversion:
int main() {
   const double cm_per_inch = 2.54;
   int val;
   char unit;
   while (cin >> val >> unit) {
    // keep reading
    if (unit == 'i')   // 'i' for inch
      cout << val << "in == “
          << val*cm_per_inch << "cm\n";
    else if (unit == 'c')   // 'c' for cm
       cout << val << "cm == "
          << val/cm_per_inch << "in\n";
    else
       return 0; // terminate on a “bad
         // unit”, e.g. 'q'
   }
}
```

Chapter 3: Input and Type

- Builtin types:
  - int, double, bool, char
- Library types: string, vector
- Input and output
- Operators—“overloading”
- Variable names in C++
- Simple computations
- Literals
- Declaration & initialization
- Type safety

Questions to note:

(a) What is a literal?

Chapter 3: Input and Type

- Builtin types:
  - int, double, bool, char
- Library types: string, vector
- Input and output
- Operators—“overloading”
- Variable names in C++
- Simple computations
- Literals
- Declaration & initialization
- Type safety

Questions to note:

(a) What is a literal?

Literals are constant values

```
int a = 6;
double b = 5.6;
string prompt=”Enter your name: “;
```

Chapter 3: Input and Type

- Builtin types:
  - int, double, bool, char
- Library types: string, vector
- Input and output
- Operators—“overloading”
- Variable names in C++
- Simple computations
- Literals
- Declaration & initialization
- Type safety

Questions to note:

(b) Write a program that converts spelled-out one-digit numbers such as “zero” and “two” into digits. When the user enters a number-name, the program should print out the corresponding digit.

Chapter 3: Input and Type

- Builtin types:
  - int, double, bool, char
- Library types: string, vector
- Input and output
- Operators—"overloading"
- Variable names in C++
- Simple computations
- Literals
- Declaration & initialization
- Type safety

Questions to note:

(b) Write a program that converts spelled-out one-digit numbers such as "zero" and "two" into digits. When the user enters a number-name, the program should print out the corresponding digit.

No solution is given.

Switch statement use is suggested.

Chapter 3: Input and Type

- Builtin types:
  - int, double, bool, char
- Library types: string, vector
- Input and output
- Operators—“overloading”
- Variable names in C++
- Simple computations
- Literals
- Declaration & initialization
- Type safety

To-do:

(a) be ready to answer questions from **Review** at the end of the chapter

(b) be ready to work on **programming Exercises** that were given as HW assignment (graded and not graded)

(c) review the quiz questions

Chapter 4: Computation

- Expressing computations
  - Correctly, simply, efficiently
  - Divide and conquer
  - Use abstractions
  - Organizing data, `vector`

- Algorithms
  - sort()

- Language features
  - Expressions
    - Boolean operators (e.g. ||)
    - Short cut operators (e.g. +=)
    - Constant expressions (const and constexpr)
  - Statements
    - if-statements
    - switch statements
    - assignment statements, …
  - Control flow
  - Functions
    - declaration
    - definition
    - why do we need functions

Chapter 4: Computation

```
// Eliminate the duplicate words; copying unique words
    vector<string> words;
   string s;
   while (cin >>s && s!= "quit")
      words.push_back(s);
   sort(words.begin(), words.end());
   vector<string>w2;
   if (0 < words.size()) {
      w2.push_back(words[0]);
      for (int i=1; i < words.size(); ++i)
        if (words[i-1]!=words[i])
           w2.push_back(words[i]);
    }
   cout<< "found " << words.size()-w2.size() << " duplicates\n";
   for (int i=0; i<w2.size(); ++i)
    cout << w2[i] << "\n";
```

Chapter 5: Errors

- Errors (“bugs”) are unavoidable in programming
  - Sources of errors?
    - Poor specification
    - Incomplete programs
    - Unexpected arguments, etc.
  - Kinds of errors?
    - Compile-time errors
    - Link-time errors
    - Run-time errors
    - Logic errors

- Minimize errors
  - Organize code and data
  - Debugging
  - Testing
- Do error checking and produce reasonable messages
  - Input data validation
  - Function arguments
  - Pre/post conditions
- Exceptions
  - throw

Chapter 5: Errors

```
int f2(int a, int b)
{
   if (a < 0 or b < 0)
      throw invalid_argument(“
negative arguments in function
call”)

   else
   {
    // …
   }
}
```

```
int main()
{
   try
   {
   // …
   }
   catch (out_of_range&)
  {  cerr << "oops – some vector
"                       " index out of
range\n";
   }
   catch (…) {
   cerr << "oops – some exception\
n";
   }
   return 0;
}
```

Chapter 8: Functions

- Declarations and definitions
- Headers and the preprocessor
- Scope
  - Global, class, local, statement
- Function calls
  - by value,
  - by reference (via pointer), and
  - by const reference
- Namespaces
  - Qualification with :: and using

```
namespace Jack {
//  in Jack's header file
class Glob{ /*…*/ };
class Widget{ /*…*/ };
}

// in our code
#include "jack.h";
#include "jill.h";

void my_func(Jack::Widget p)
{
// OK, Jack's Widget class will not
// clash with a different Widget
// …
}
```

Chapter 8: Functions

Questions to note:

(a) What is the difference between *function definition* and *function declaration*?

(b) What is the difference between *pass-by-reference* and *pass-by-value*?

(c) What is a call stack?

Chapter 8: Functions

Questions to note:

(d) Define a function `prod()` that accepts two vectors passed by const reference, v1 and v2, and a vector passed by reference, v3.

The function should modify the vector v3, by adding/appending the products of corresponding pairs of values from the first two vectors v1 and v2. It is possible for the vectors v1 and v2 to have different sizes. If their sizes are different, then only add the products only as long as it is possible, and then stop.

Chapter 8: Functions

Questions to note:

(d) Define a function `prod()` that accepts two integer vectors passed by const reference, v1 and v2, and an integer vector passed by reference, v3.

The function should modify the vector v3, by adding/appending the products of corresponding pairs of values from the first two vectors v1 and v2. It is possible for the vectors v1 and v2 to have different sizes.

If their sizes are different, then only add the products only as long as it is possible, and then stop.

```
void prod(const vector<int> v1, const vector<int> v2, vector<int> v3);
```

Chapter 9: Classes

- User defined types
  - class and struct
  - private and public members
    - Interface
  - const members
  - constructors/destructor
  - operator overloading
  - Helper functions
  - Enumerations enum
- Date type

Questions to note:

- What is a constructor and what types of constructors you know?

Chapter 9: Classes

- User defined types
  - class and struct
  - private and public members
    - Interface
  - const members
  - constructors/destructor
  - operator overloading
  - Helper functions
  - Enumerations enum
- Date type

Questions to note:

- What is a constructor and what types of constructors you know?
  - default constructor
  - constructor for one or more parameters
  - copy constructor
  - move constructor

Chapter 9: Classes

- User defined types
  - class and struct
  - private and public members
    - Interface
  - const members
  - constructors/destructor
  - operator overloading
  - Helper functions
  - Enumerations enum
- Date type

Questions to note:

- Design a data type that will represent a complex number in its rectangular form, a+bi

Chapter 9: Classes

- User defined types
  - class and struct
  - private and public members
    - Interface
  - const members
  - constructors/destructor
  - operator overloading
  - Helper functions
  - Enumerations enum
- Date type

Questions to note:

- Design a data type that will represent a complex number in its rectangular form, a+bi
  - …
  - Will you consider overloading the output operator<< to display the objects of type Complex?

Chapter 10: Streams

- The I/O stream model,
  - istream
  - ostream
- File types
  - Opening for input/output
  - Error handling
    - check the stream state
- User defined output operator<< and input operator>>
- only Sections 10.1-10.6

Questions to note:

- Write a program that produces the sum of all the numbers in a file of whitespace-separated integers

Chapter 17: Vector and Free Store

- Built vector type
- Pointer type
- The **new** operator to allocate objects on the free store (heap)
- Run-time memory organization
  - Code, static data, free store/heap, stack (review!)
- Memory leaks
- **void***
- **`this`** pointer
- Pointers vs references

Questions to note:

- What is a null pointer? When do we need to use one?

Chapter 17: Vector and Free Store

- Built vector type
- Pointer type
- The **new** operator to allocate objects on the free store (heap)
- Run-time memory organization
  - Code, static data, free store/heap, stack (review!)
- Memory leaks
- **void***
- **`this`** pointer
- Pointers vs references

Questions to note:

- What is a null pointer? When do we need to use one?
  - When declaring a pointer, set it to **nullptr** if not ready to initialize
  - When the pointer is not pointing to an object at the moment – set it to **nullptr**
  - Recall moving – set the pointer to **nullptr**

Chapter 17: Vector and Free Store

- Built vector type
- Pointer type
- The **new** operator to allocate objects on the free store (heap)
- Run-time memory organization
  - Code, static data, free store/heap, stack (review!)
- Memory leaks
- **void***
- **`this`** pointer
- Pointers vs references

Questions to note:

- Draw the pictorial memory representation that reflects the execution of the following code fragment:

```
char* p = new char(6);
p[0] = 'a';
p[1] = 'b';
p[2] = 'c';

char* p2;
p2 = p;
*p2 = 'd';
p2 += 2;
*p2 = 'h';
```

Chapter 18: Vectors and Arrays

- Vector copy constructor
- Vector copy assignment
- Shallow and deep copy
- Arrays—avoid if possible
- Moving

Questions to note:

- What is an **explicit** constructor? Where would you prefer one over the (default) alternative?

Chapter 18: Vectors and Arrays

- Vector copy constructor
- Vector copy assignment
- Shallow and deep copy
- Arrays—avoid if possible
- Moving

Questions to note:

- What is an **explicit** constructor? Where would you prefer one over the (default) alternative?
- Recall this issue we had:

```
vector v1 = 7;
// v1 has 7 elements, each with the value 0
v1 = 20; // v1 is now a new vector with 20 elements
```

(Initialization: implicit conversions and explicit constructors)

Chapter 18: Vectors and Arrays

- Vector copy constructor
- Vector copy assignment
- Shallow and deep copy
- Arrays—avoid if possible
- Moving

Questions to note:

- Define a copy constructor for vector class

  ```
  vector(const vector& other);
  ```

Chapter 19: Vectors and Arrays

- Overloading [ ] (const and non-const)
- Overloading at()
- Changing vector size
- Added
  - resize(int n),
  - push_back(double d)
- Optimized copy assignment (self-study)
- Templates
- Range checking
- Exception handling
- `unique_ptr`

Questions to note:

- Give an example of `unique_ptr` use

Chapter 19: Vectors and Arrays

- Overloading [ ] (const and non-const)
- Overloading at()
- Changing vector size
- Added
  - resize(int n),
  - push_back(double d)
- Optimized copy assignment (self-study)
- Templates
- Range checking
- Exception handling
- unique_ptr

Questions to note:

- Give an example of unique_ptr use

```
unique_ptr<int> a{ new int };
// only a owns access
int* b = a; // error

int* b = a.release();
delete b;
```

Chapter 19: Vectors and Arrays

- Overloading [ ] (const and non-const)
- Overloading at()
- Changing vector size
- Added
  - resize(int n),
  - push_back(double d)
- Optimized copy assignment (self-study)
- Templates
- Range checking
- Exception handling
- `unique_ptr`

Questions to note:

- Give an example of `unique_ptr` use

```
unique_ptr<int> a = new int ;
// error
int* b = a; // error
```

```
int* b = a.release();
delete b;
```

Classes: inheritance, polymorphism, hierarchies, etc.

- Mostly from Chapter 14
  - Section 14.3 in particular
- Encapsulation
- Polymorphism
- Inheritance
  - Hierarchies
  - Has-a vs is-a relationship
  - `private, protected, public`

Questions to note:

- Why use inheritance?
  - it reduces the duplication of existing code
  - it can save time during program development by taking advantage of proven, high-quality, already defined classes

Recursion with C++

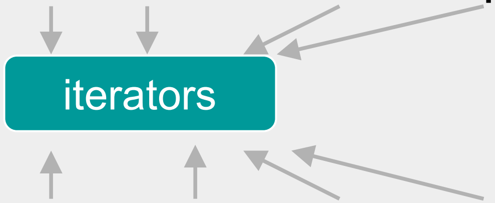
- Recursion concepts
  - Base case(s)
  - Recursive calls
- Fibonacci numbers
- Structural recursion
- Palindromes
- How to convert an iterative function to a recursive one

To-do:

- Review the lecture slides:
  - Definition of recursion
  - Call stack
  - Examples
- Do the practice

Chapter 20: The STL (containers and iterators)

- Generic programming
  - “lifting an algorithm”
- Standard Template Library
- 60 Algorithms
  - sort, find, search, copy, …

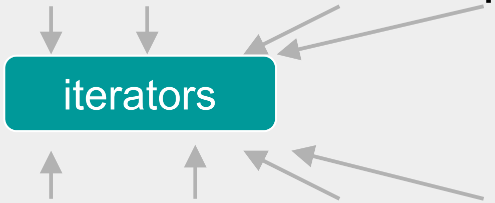


iterators

  - vector, list, map, unordered_map,…
- 10 Containers
- iterators define a sequence
- Function objects

Chapter 20: The STL (containers and iterators)

- Generic programming
  - “lifting an algorithm”
- Standard Template Library
- 60 Algorithms
  - sort, find, search, copy, …



  - vector, list, map, unordered_map,…
- 10 Containers
- iterators define a sequence
- Function objects

```
// Concrete STL-style code  for a more
// general version of summing values

// Iter should be an Input_iterator
// T should be something we can + and =
template<class Iter, class T>
T sum(Iter first, Iter last, T s)
{ // T is the  “accumulator type”
        while ( first != last ) {
              s = s + *first;
              ++first;
        }
        return s;
}
```

Chapters 20 - 21

- Sequences and iterators
- Parameterized find method
- Parameterized find_if method
  - predicates
- Predicate as function
- Predicate as function object
- Lambda expressions

To-do:

- Review the lecture slides:
  - terminology
  - Examples
  - In-class work
- Do the practice

Chapter 21: Algorithms and Maps

- Associative containers:
  - map
  - set
  - unordered_map
- Standard algorithms
  - copy, sort,

To-do:

- Review the lecture slides:
  - Examples of container use
  - In-class work

Final Exam structure

- Part 1
  - 10 multiple choice, true/false questions
  - 3 points each question
- Part 2
  - 6 short answer questions
  - 5 points each
- Part 3
  - 4 coding questions
  - 10 points each