C++ Language Primer & OO Concepts

Chapter 1

Objectives

- This module covers two important topics.
- Section I covers some of the basic C++ constructs used in the course:
  - C++ variables, expressions, control loops, arrays, and I/O
- Section II describes the basic concepts of an object and a class
  - Sample code located in Chap01
  - Labs located in Labs/Lab1

main

- **Programs begin with main()**
  - **Can return exit status**
  - **Can accept command line arguments**
  - **“falling off the bottom” of main means return success**

```
int main(int argc, char **argv)
{
  ...
  return 0;
}
```

```
int main()
{
  ...
}
```

← equivalent

variables

- **Built-in variable types**

integers →

floats →

```
int main(){
  int       i, j, k;
  short     s;
  long      l;
  unsigned  u;

  double    d = 1.2;
  float     f;

  char      c = 'A';

  bool      b = true;
  ...
}
```

comments

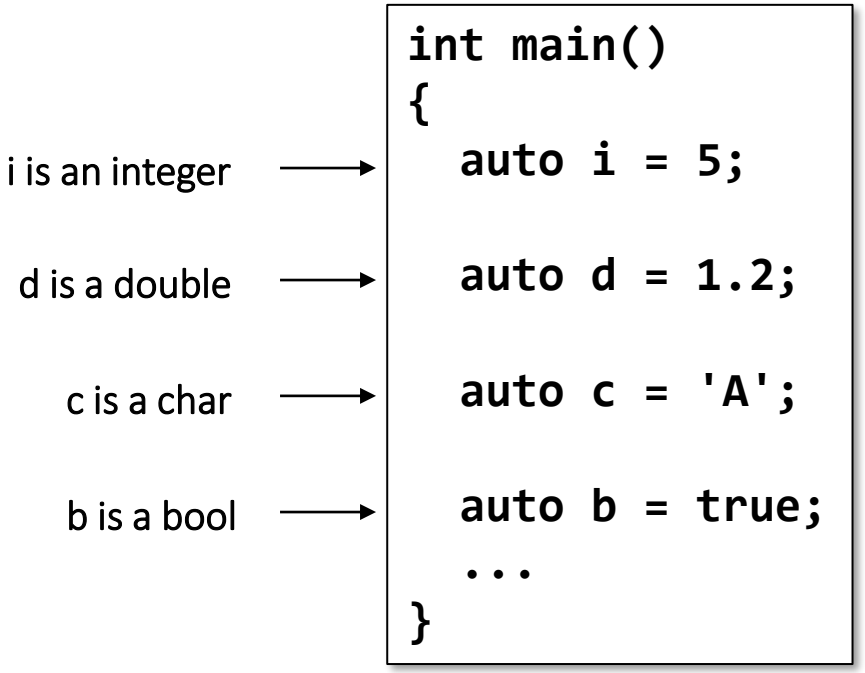
- **Two styles of comments:**
  - **potentially multiline**
  - **single line**

```
/*
  program to compute the log
  base 2 of a number.
*/
int main(){
  int x = 100;  // number
  int n = 0;    // counter

}
```

auto

- **auto deduces the variable type from the initializer**



expressions

- **Common arithmetic operators available**

```
+ addition
- subtraction
* multiplication
/ division
% modulus (remainder)
```

```
int main()
{
  ...
  x = (5 + y) * 2;
}
```

Shorthand expressions

- **Concise versions of operators**
  - **combination assignment operators**
  - **increment/decrement**

add 5 →

post-increment →

pre-decrement →

```
int main(){
  ...

  x += 5; // same as: x = x + 5

  x++;
  --x;
}
```

if statement

- **Execute statements if condition is true**
  - **may have optional else part**

```
== equal
!= not equal
<  less
<= less or equal
>  greater
>= greater or equal
```

optional →

```
int main(){
  ...
  if (income < 10000)
  {
    aid = true;
    rate = 0.10;
  }
  else
  {
    aid = false;
    rate = 0.25;
  }
}
```

Short-circuit evaluation

- **Logical expression are evaluated left-to-right**
- **Evaluation guaranteed to stop as soon as final outcome is determined**

P	Q	P || Q	P && Q	!P	!Q
T	T	T	T	F	F
T	F	T	F	F	T
F	T	T	F	T	F
F	F	F	F	T	T

only divide if **d** not zero →

```
int main()
{
   ...

   if (d != 0 && n/d > 2)
      ...
}
```

Conditional operator ?:

- **Makes an expression out of a test**
  - **a shorthand if statement**
  - **C and C++'s sole "ternary" operator (takes 3 operands)**

compute maximum →

```
int main()
{
   ...

   n = x > y ? x : y;
}
```

while and do loop

- ***while*** **repeats until condition is false**
- **Expression is evaluated before body**

repeats while true →

```
int main()
{
  ...

  while (x > 0)
  {
    ...
  }
}
```

- ***do*** **repeats until condition is false**
- **Expression is evaluated after body**

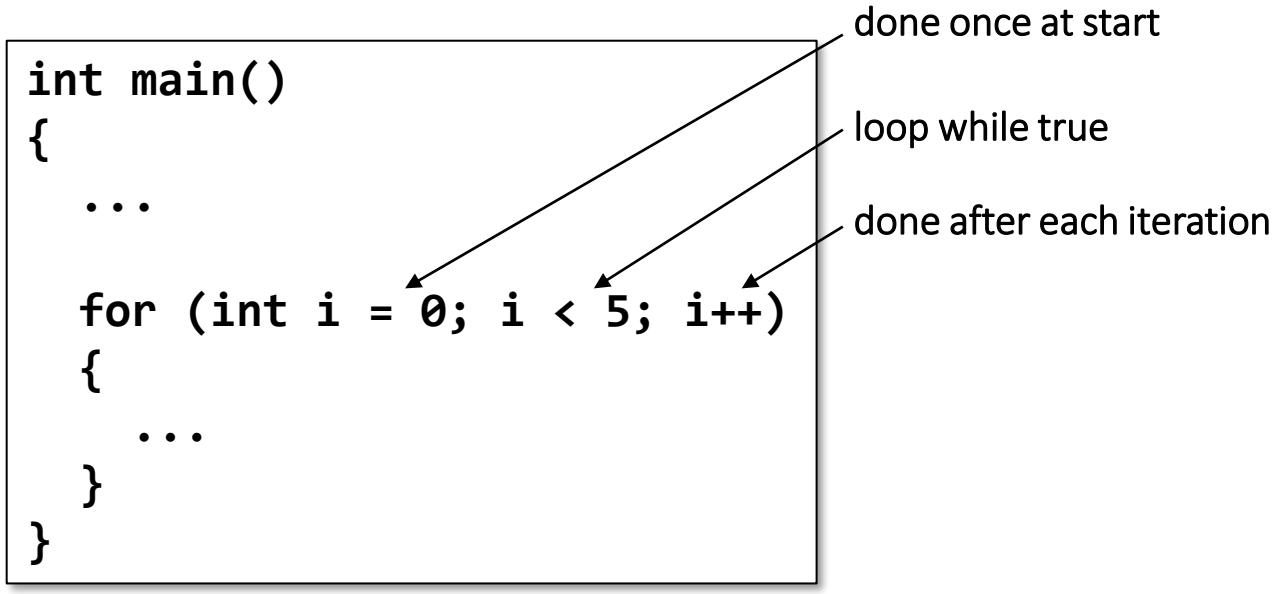
repeats while true →

```
int main()
{
  ...

  do
  {
    ...
  } while (x > 0)
}
```

for loop

- **Localizes control information**
  - **initialization, test condition, prepare-for-next-iteration expression**
  - **parts separated by semicolons**



Arrays

- **Arrays give an indexed sequence of elements**
  - **size must be a compile-time constant expression**
  - **indices are 0 through size-1**
  - **no bounds-checking**
  - **values must be explicitly set or initialized (next slide)**

```
int main()
{
  int a[6];

  a[0] = 1;
  a[1] = 2;
  a[2] = 4;
  ...

  a[5] = 32;
}
```

Array initialization

- **Arrays can be initialized**
  - **explicit size may be omitted (if initializer list is supplied)**

```
int main()
{
  int a[] { 1, 2, 4, 8, 16, 32 };

  ...
}
```

Multi-dimensional arrays

- **Arrays may be more than one dimension.**
- **If initialized multidimensional arrays must have bounds for all dimensions except the first (left-most):**

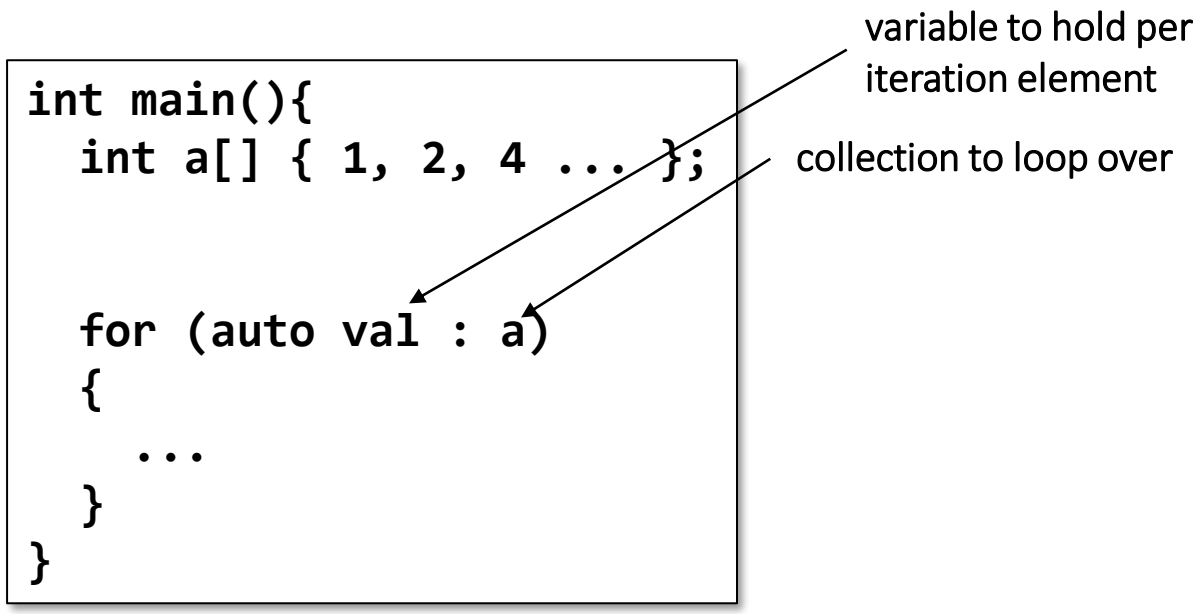
```
int a[][2] = {{1,2}, {3,4}, {5,6}};
```

```
int main(){
  int a[3][4];

  for (i = 0; i < 3; i++)
  {
    for (j = 0; j < 4; j++)
    {
      a[i][j] = i * j;
    }
  }
}
```

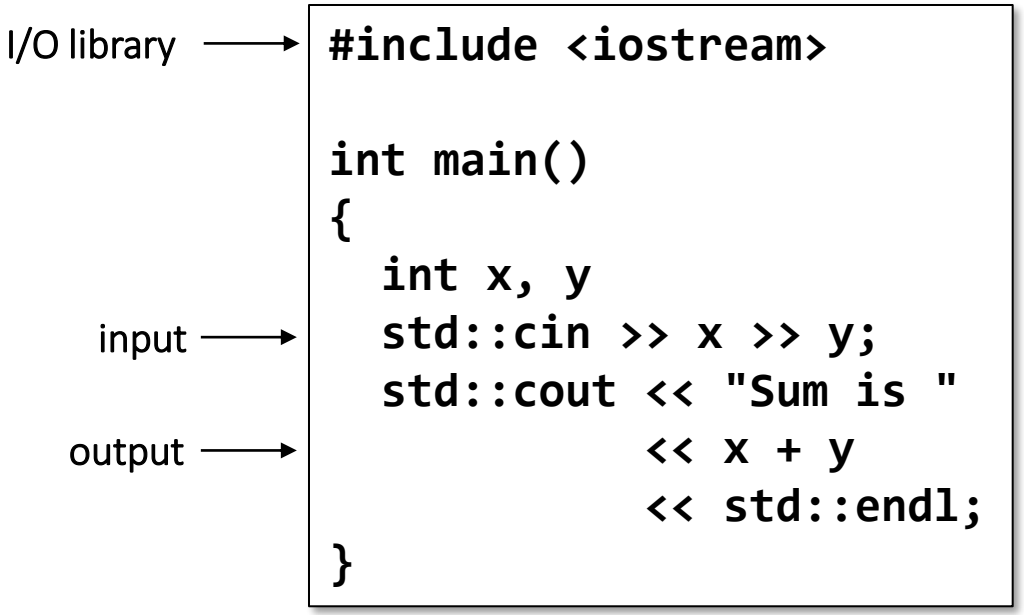
range for loop

- **Iterates from begin to end of collection**



Input/Output

- **Console input/output provided by <iostream>**
  - **cin, cout, cerr, …**
- **Additional character and line-based operations provided by <string>**
  - **to_string, stoi, stof,…**
- **Several functions to manipulate C strings and arrays provided by <cstring>**
  - **strcpy, strcat, strlen, …**



Section I Summary

- **C++ has many important language constructs you will use throughout this course:**
  - **Programs begin at main()**
  - **Many built-in variable types**
  - **Standard and shorthand expressions**
  - **Control and loop constructs**
  - **Arrays**
  - **I/O using the iostream library**
- **Now is a good time to review the sample code and work on the lab exercise.**

Section II Objectives

- **Describe the basic concepts of an object and a class.**
- **Explain how the object model provides the framework for abstraction, encapsulation and instantiation.**
- **Define the concept of an abstract data type.**
- **Define the terms method and message.**
- **Explain the use of class inheritance for enhancing reusability of code.**
- **Define the term polymorphism and explain how it can be used to make object-oriented programs more flexible and easier to maintain.**

What is an object?

An object is a software entity which has attributes and behavior.

The behavior of an object is represented by the set of operations, while the attributes represent the data.

The state of an object is the value of these attributes at any point in time.

Behaviors may alter or query state.

Abstraction

An abstraction captures the essential features of an entity, suppressing unnecessary details.

All instances of an abstraction share these common features.

Abstraction helps us deal with complexity.

Encapsulation

The implementation of an abstraction should be hidden from the rest of the system or encapsulated.

Data itself is *private*, walled off from the rest of the program.

Data can only be accessed through functions with a *public* interface.

Class and Instantiation

A class groups all objects with common behavior and common structure.

A class describes the data and behaviors and allows production of new objects of the same type.

An object is an instance of a class.

Data Types

Data types describe a set of objects with the same representation.

There can be several operations associated with each data type.

The data itself is directly accessible to the rest of the program.

Abstract Data Types

An *Abstract Data Type* (ADT) is a data type that "hides" the implementation of its operations.

- **There is a *public* set of operations.**
- **The data itself is *private*.**

ADT's can be implemented via a class in C++.

ADT's in C++ may have “sugar coating”, such as overloaded operators, that may make their usage appear identical to that of built-in types.

Abstract Data Type Example

C++ does not have a built-in type for complex numbers. We can however create a new type called Complex with operations such as addition, subtraction, multiplication, division.

```
Complex c(1, 3);
Complex d(2, 4)
Complex e = c + d;
```

Methods

In object-oriented software, a function defined for an object is called a method.

A method is only accessible via its object, while a function is a free-standing entity.

Objects communicate between each other through method invocation.

Class Inheritance

Inheritance is defining a new class by specifying only its difference from another class.

Derived classes (subclasses) inherit behavior from the base class (superclass).

Polymorphism

Polymorphism is the ability of two or more classes of objects to respond to the same message, each in its own way.

The receiving object's method is determined at run time by a process known as dynamic binding.

An object does not need to know to whom it is sending a message

Section II Summary

- **An object has both state and behavior.**
- **The object model provides the framework for abstraction, encapsulation and instantiation.**
- **An abstraction captures the essential features of an entity, suppressing unnecessary details.**
- **Encapsulation protects internal data from corruption and isolates users of an object from changes in data representation.**
- **A class is a basis for objects with similar behavior and structure, supporting instantiation.**
- **An abstract data type has private data and a public set of operations.**
- **Class inheritance supports reuse by providing the capability to define new objects by specifying differences and extensions to an existing object.**