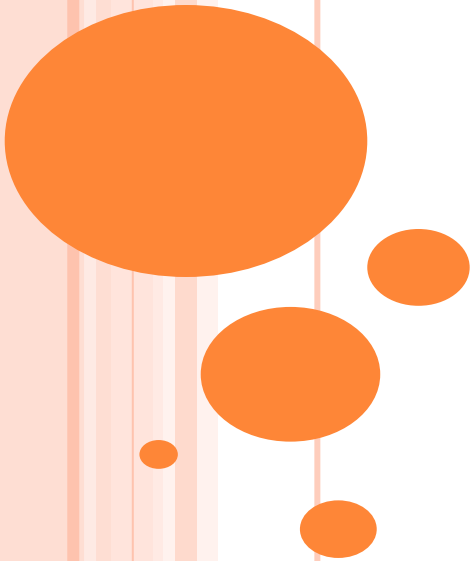


CSE 107: Object Oriented Programming Language

Dr. Tanzima Hashem
Professor
CSE, BUET

Textbook

- Teach Yourself C++, Third Edition.
  - Herbert Schidlt,

- Tentative Plan
  - Week 1-4, 9, 10 … C++
  - Week 5-8, 11-14 … Java

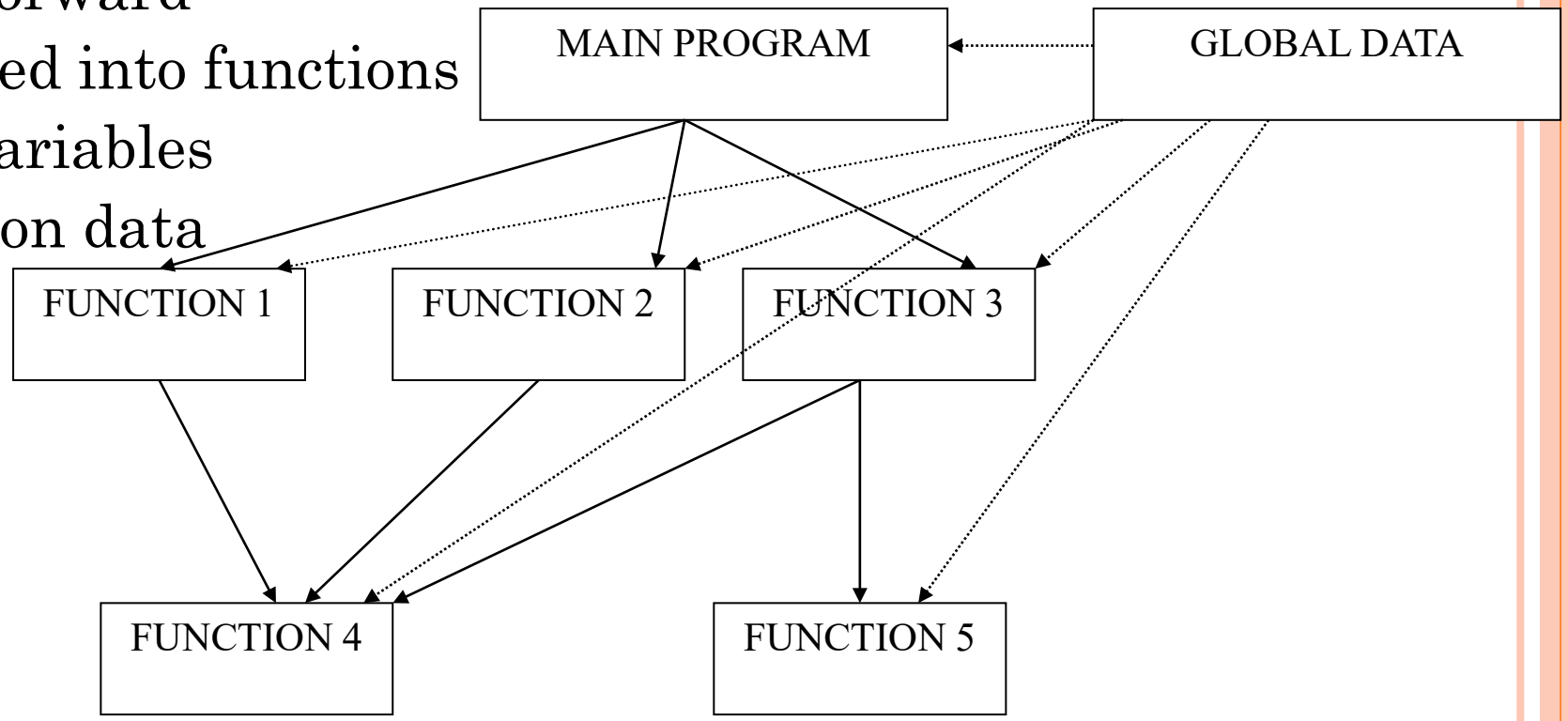
Program

- A program is a set of instructions that tells the computer what to do to come up with a solution for a particular problem

- Programming Paradigms
  - Procedural/Structured Programming
    - Focus on functions
  - Object Oriented Programming
    - Focus on objects (data)

Procedural Programming

- Simple and Straightforward
  - A program is divided into functions
  - Data is stored in variables
  - Functions operate on data



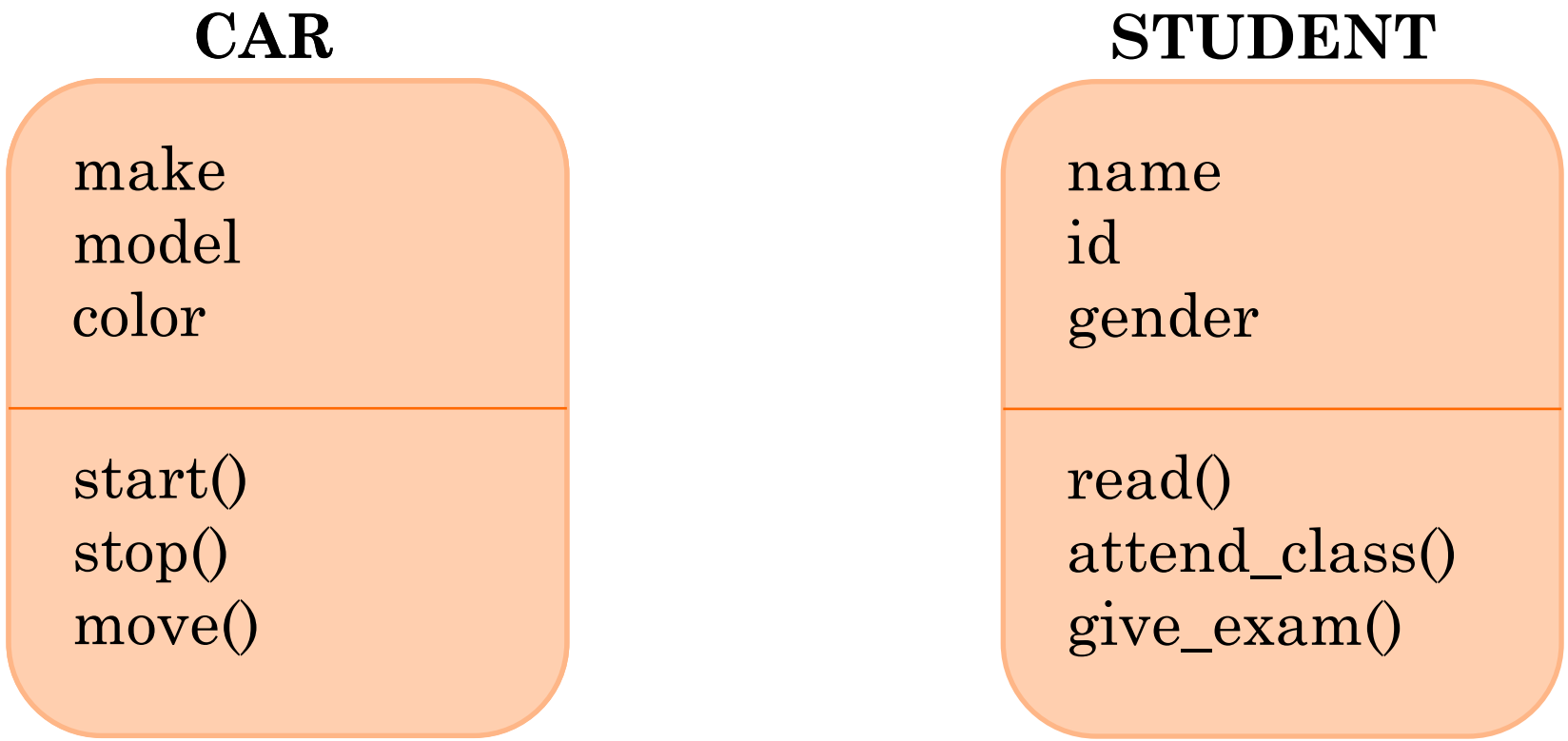
- Problems with procedural programming appear as programs grow larger and more complex

Object Oriented Programming

- To manage increasing complexity object-oriented programming was conceived
  - Better organization of the code
  - Smaller code
  - Reuse of code
- It organizes a program around its data (that is, objects) and a set of well-defined interfaces to that data

Objects

- Variables and functions are combined in a unit, called an object
  - Variable → Property
  - Function → Method



Features

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Abstraction

- Humans manage complexity through abstraction
  - People do not think of a car as a set of tens of thousands of individual parts
  - They think of it as a well-defined object with its own unique behavior
  - They can ignore the details of how the engine, transmission, and braking systems work.
  - Instead, they are free to utilize the object as a whole

Abstraction

- Hide some of the functions and properties from outside
- Benefits
  - Simple interface → reduce complexity
  - Reduce the impact of change

Encapsulation

- Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse

- One way to think about encapsulation is as a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper

- Access to the code and data inside the wrapper is tightly controlled through a well-defined interface

Encapsulation

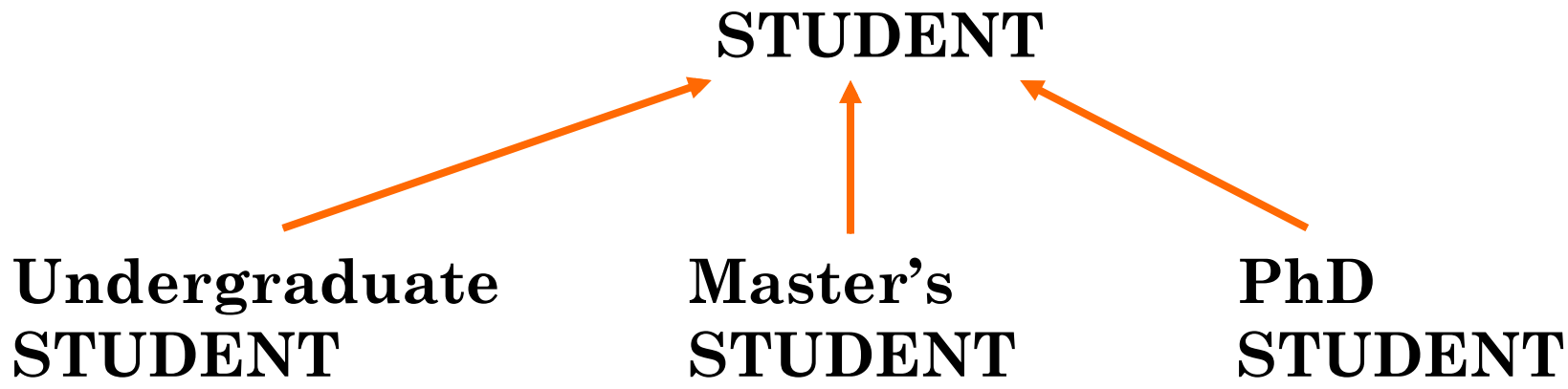
- Generally the basis of encapsulation is the class
- A class defines the structure and behavior (data and code) that will be shared by a set of objects
- Each object of a given class contains the structure and behavior defined by the class
- Objects are sometimes referred to as instances of a class
- A class is a logical construct; an object has physical reality

Encapsulation

- When you create a class, you will specify the code and data that constitute that class
  - These elements are called members of the class
  - The data defined by the class are referred to as member variables or instance variables
  - The code that operates on that data is referred to as member methods or just methods
- The behavior and interface of a class are defined by the methods that operate on its data

INHERITANCE

- Inheritance is the process by which one object acquires the properties of another object



- Without hierarchical classification, each object would need to define all of its characteristics explicitly
- Inheritance helps to eliminate redundant code

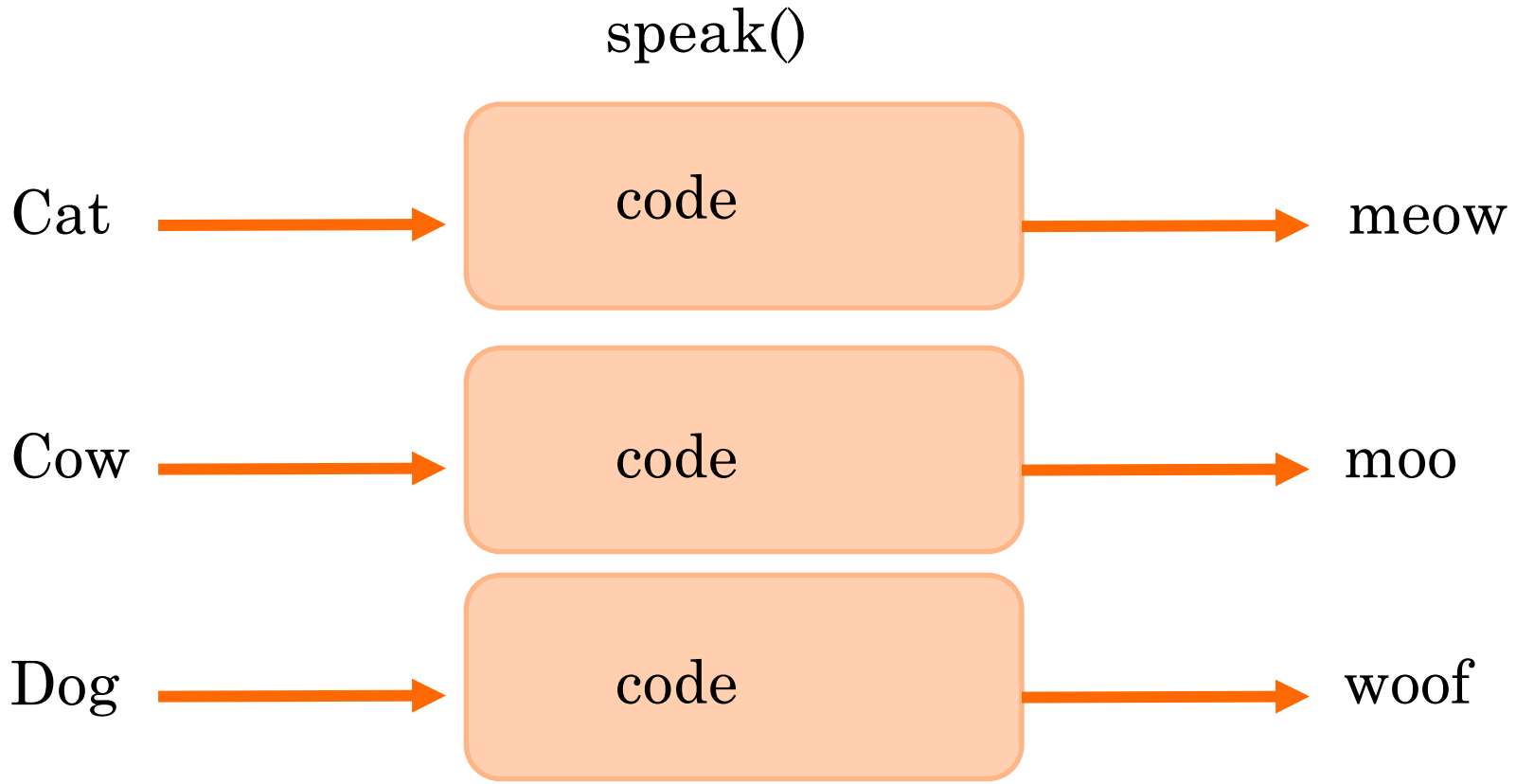
POLYMORPHISM

- Poly – Many
- Morph – Form

- Polymorphism is the characteristic that enables an entity to co exist in more than one form

- Polymorphism is a feature that allows one interface to be used for a general class of actions

POLYMORPHISM



Benefits of OOP

- Abstraction
  - Reduce complexity+ isolate impact of changes
- Encapsulation
  - Reduce complexity+ increase reusability
- Inheritance
  - Eliminate redundant code
- Polymorphism
  - Reduce complexity

Summary

- Procedural Programming
  - List of instructions for a computer to follow
  - Groups of similar tasks are organized as functions
  - Functions share global data

- Object Oriented Programming
  - Emphasis on data rather than procedures/functions
  - Programs are divided into objects
  - Data structures are designed such that they characterize the objects
  - Functions that operate on the data of an object are tied together in the same data structure

Introduction to C++

- C++ is the C programmer's answer to Object-Oriented Programming (OOP).
- C++ is an enhanced version of the C language.
- C++ adds support for OOP without sacrificing any of C's power, elegance, or flexibility.
- C++ was invented in 1979 by Bjarne Stroustrup at Bell Laboratories in Murray Hill, New Jersey, USA.

C++ New Header

- The new-style headers do not have .h extension and do not specify filenames
  - **#include <iostream>**
  - **#include <vector>**
  - **#include <string>**
  - **#include <cstring>**
  - **#include <cmath>**
- Specify standard identifiers that are mapped to files by the compiler
- C++ still support C-style header files, e.g.,
  - **#include <stdio.h>**
  - **#include <string.h>**

C++ Namespace

- A namespace is a declaration region
- It localizes the names of identifiers to avoid name collisions
- The contents of new-style headers are placed in the std namespace

- In order to use the library functions with new- style headers we need to bring std namespace into visibility
  - **using namespace std;**

- In C, the names of the library functions and other such items are not localized by any namespace, instead, they are implicitly placed into global namespace

C++ Console I/O

- I/O is performed using I/O operator instead of I/O functions

- The output operator (insertion operator) is <<
- The input operator (extraction operator) is >>

- In order to use either << or >> operator programs must begin with the followings

```
#include <iostream>
…
using namespace std;
```

C++ Console I/O (Output)

- Output operator << is associated with cout, a predefined stream linked to the console output (monitor)
- In general, cout << *expression*

```
printf("Hello World!");
cout << "Hello World!";

printf("%d", iCount); /* int iCount */
cout << iCount;

printf("%f", 100.99);
cout << 100.99;

printf("\n")
cout << "\n", or cout << '\n', or cout << endl
```

C++ Console I/O (Input)

- Input operator >> is associated with cin, a predefined stream linked to the console input
- In general, cin >> variable;

```
scanf("%s", strName); /* char strName[16] */
cin >> strName;

scanf("%d", &iCount); /* int iCount */
cin >> iCount;

scanf("%f", &fValue); /* float fValue */
cin >> fValue;
```

C++ Console I/O (I/O Chaining)

- cout << “Hello” << ‘ ‘ << “World” << ‘!’;
- cout << “Value of iCount is: ” << iCount;
- cout << “Enter day, month, year: ”;
- cin >> day >> month >> year;
  - cin >> day;
  - cin >> month;
  - cin >> year

C++ Console I/O (An Example)

```
#include <iostream>
using namespace std;
int main()
{
        char str[16];
        cout << “Enter a string: ”;
        cin >> str;
        cout << “You entered: ” << str;
}
```

Classes

- C++ Classes are the logical abstraction or model of C++ Objects
- A Class declaration defines a new type
- It determines what an object of that type will look like
- It determines the nature of the data and functions of that type
- Classes must be defined before creating the objects, i.e., objects cannot be created without the classes
- Definition of a class does not create any physical objects rather a logical abstraction

CLASSES

- General syntax -

```
class class-name
{
        // private functions and variables
public:

        // public functions and variables
}object-list (optional);
```

Classes

- Member access specifiers

- public:
  - Can be accessed outside the class directly
    - The public stuff is *the interface*

- private:
  - Accessible only to member functions of class
  - Private members and methods are for internal use only

Classes

```
class Rectangle{
        int height, width;
public:
        void set_values (int,int);
        int area() {return (height*width);}
};

void Rectangle::set_values (int a, int b) {
        height = a;
        width = b;
}
```

CLASSES

```
class Circle{
        double radius;
public:
        void setRadius(double r) {radius = r;}
        double getDiameter() { return radius *2;}
        double getArea();
        double getCircumference();
};
double Circle::getArea()
{
    return radius * radius * (22.0/7);
}
double Circle:: getCircumference()
{
    return 2 * radius  * (22.0/7);
}
```

C++ Objects

- C++ Classes are used as the type specifier to create C++ Objects

  Rectangle recta, rectb;

- An object declaration creates a physical entity of its class type, i.e., occupies memory space class type

- Each object has its own copy of data

C++ Objects

```
int main () {

        Rectangle recta, rectb;
        recta.set_values (3,4);
        rectb.set_values (5,6);

        cout << "recta area: " << recta.area() << endl;
        cout << "rectb area: " << rectb.area() << endl;

        recta.height=5;

        // Not possible, height is a private member

        return 0;
}
```

Function Overloading

- Two or more functions can share the same name as long as either
  - The type of their arguments differs, or
  - The number of their arguments differs, or
  - Both of the above
- The compiler will automatically select the correct version
- The return type alone is not a sufficient difference to allow function overloading

Function Overloading

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

class printData {
   public:
      void print(int i) {
        cout << "Printing int: " << i << endl;
      }
      void print(double  f) {
        cout << "Printing float: " << f << endl;
      }
      void print(char* c) {
        cout << "Printing character: " << c << endl;
      }
};
```

Function Overloading

```
int main(void) {
   printData pd;

   // Call print to print integer
   pd.print(5);

   // Call print to print float
   pd.print(500.263);

   // Call print to print character
   pd.print("Hello C++");

   return 0;
}
```

```
Printing int: 5
Printing float: 500.263
Printing character: Hello C++
```

Function Overloading

```
int sum (int x, int y)
{
 cout << x+y;
}

double sum(double x, double y)
{
 cout << x+y;
}
```

Function Overloading

```
int sum (int x, int y)
{
 cout << x+y;
}

int sum(int x, int y, int z)
{
 cout << x+y+z;
}
```

Function Overloading

```
// This is incorrect and will not compile.
int f1 (int a);
double f1 (int a);
.
.
.
f1(10)          //which function  does the computer call???
```

C++ Comments

- Multi-line comments
  - /* one or more lines of comments */
- Single line comments
  - // …

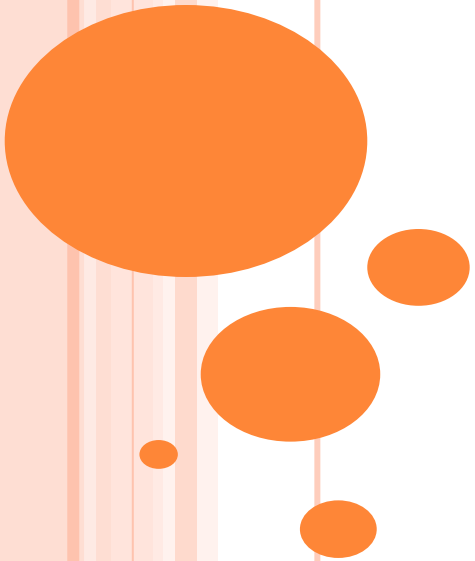
C++ Keywords (partial list)

- bool
- catch
- delete
- false
- friend
- inline
- namespace
- new
- operator
- private
- protected
- public
- template
- this
- throw
- true
- try
- using
- virtual
- wchar_t

Some Differences Between C and C++

- No need to use “void” to denote empty parameter list.
- All functions must be prototyped.
- If a function is declared as returning a value, it ***must*** return a value.
- Return type of all functions must be declared explicitly.
- Local variables can be declared anywhere.
- C++ defines the **bool** datatype, and keywords **true** (any nonzero value) and **false** (zero).

Acknowledgement
http://faizulbari.buet.ac.bd/Courses.html
http://mhkabir.buet.ac.bd/cse201/index.html



The End

Topic Covered: Chapter 1