A Course on C++

whoami

- Dr. Vishwa Kiran S
  - PhD
  - M.Tech in Computer Science and Engineering
  - B.E in Electronics and Communication Engineering
  - 19+ Years of Experience
  - Worked for
    - Mistral Solutions Pvt Ltd
    - Waveaxis Technologies Pvt Ltd
  - Handled Corporate training for
    - Nokia, Samsung, Cisco, Texas Instruments, Siemens, L&T infotech, Sasken, Honeywell, Western Digital (Japan)
    - C, C++,Java, Python QT, Linux, Device Drivers, Symbian, Meego, Android, Beagleboard, Rasperry Pi
  - Publications
    - 3 IEEE conference paper
    - 1 Springer Journal , 1 InderScience Publisher
  - Currently associated with
    - BMS Institute of Technology & Management
    - Pushkala Technologies Pvt Ltd
    - Pytriot Solutions LLP

Before starting C++

- Program
- Process
- Code Segment
- Data Segment
- Role of Compiler
- Compiler Driver
- Memory Layout of a C Program

Domains

- Application Programming
- Network programming
- System Programming
- Embedded System Programming
- Device Driver Programming

Application Programming

Web Programming

Network Programming

Kernel Programming

Embedded System

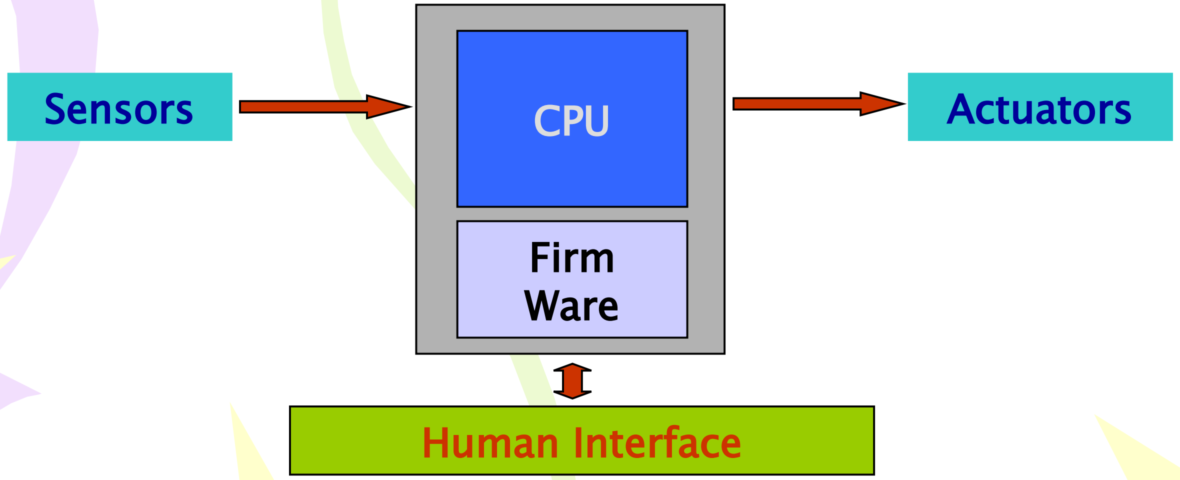
Computer System v/s Embedded System

- Combination of h/w and s/w
- Designed for generic purpose

- Combination of h/w and s/w
- Designed for specific purpose

What is an Embedded System?

- Optimally designed Hardware
- Dedicated Software / Firmware

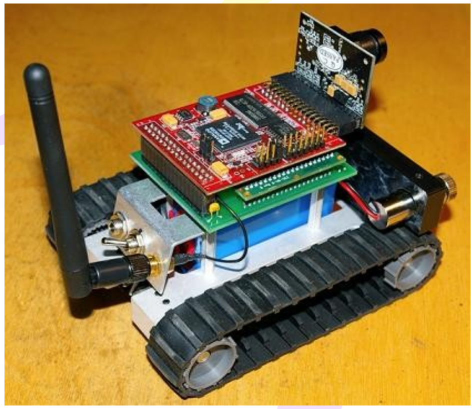


ES Domains

Aerospace

Automotive

Robotics



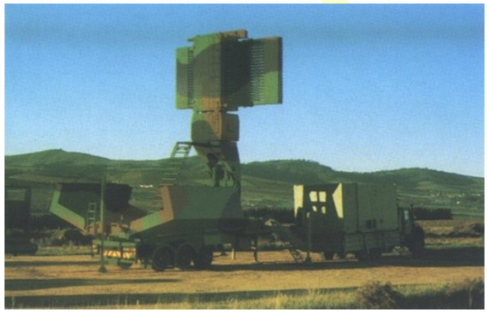
Industrial Control

Medical

Communication

Livestock Management

Military



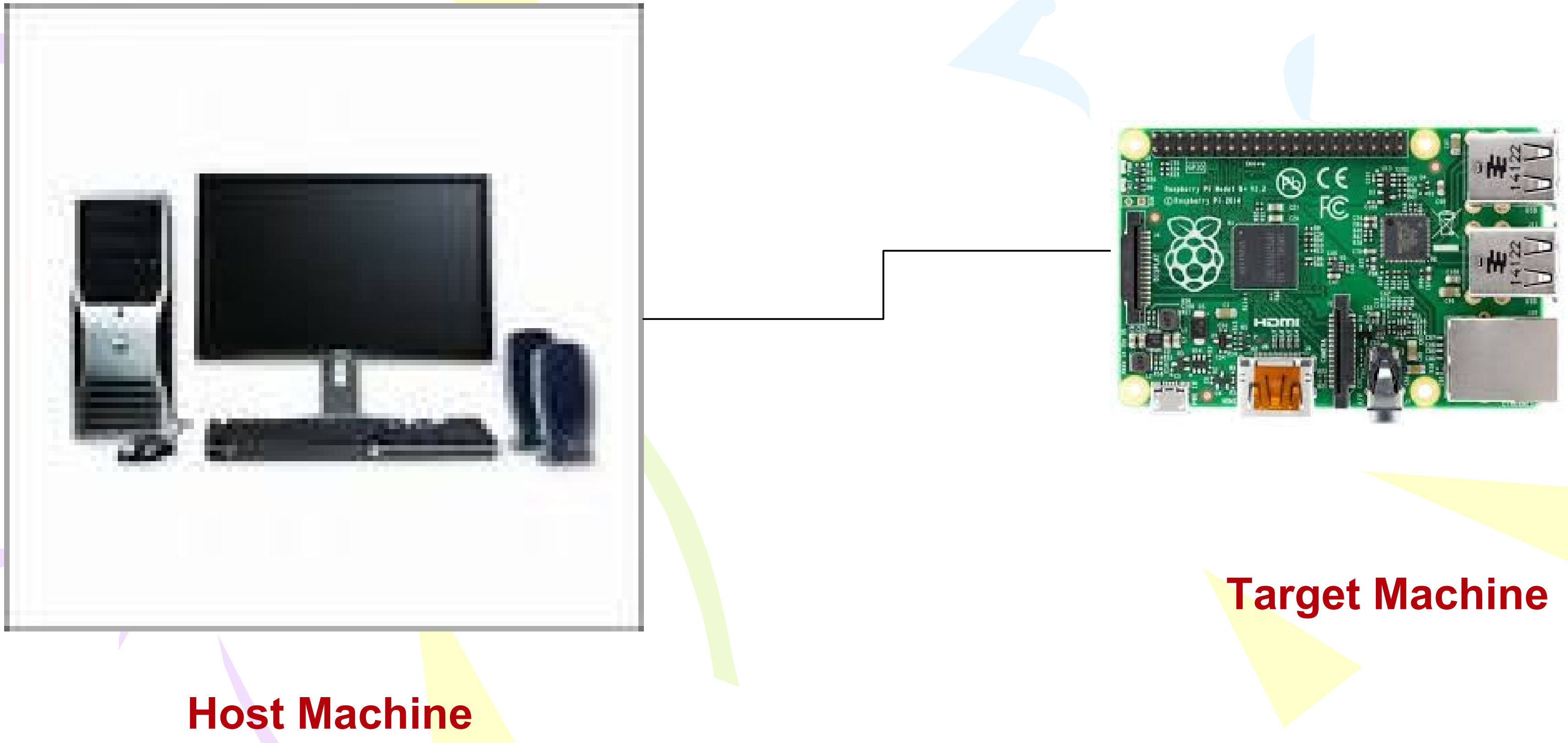
ES Domains

Types of Development

- Native Development
- Cross Development

Native Development

Cross Development



Skill Set for Embedded System Developer

- C
- C++
- Java
- Python
- QT
- GTK
- Linux

Why do we need an Operating System?

?

Government v/s Operating System

- Manages Resources
- Land
- Water
- Minerals
- Finance

- Manages System Resources
- CPU
- Memory
- Files
- Devices

What is Operating System?

- Resource Manager
- Interface between user and Resources
- Examples
  - Unix, Gnu/Linux, Windows x, Mac
  - VxWorks, psos, Qnx, RTLinux,Nucleus,
  - Symbian, IOS, Android,Meego, Bada, TIZEN

What is Operating System?

- Resource Manager
- Interface between user and Resource

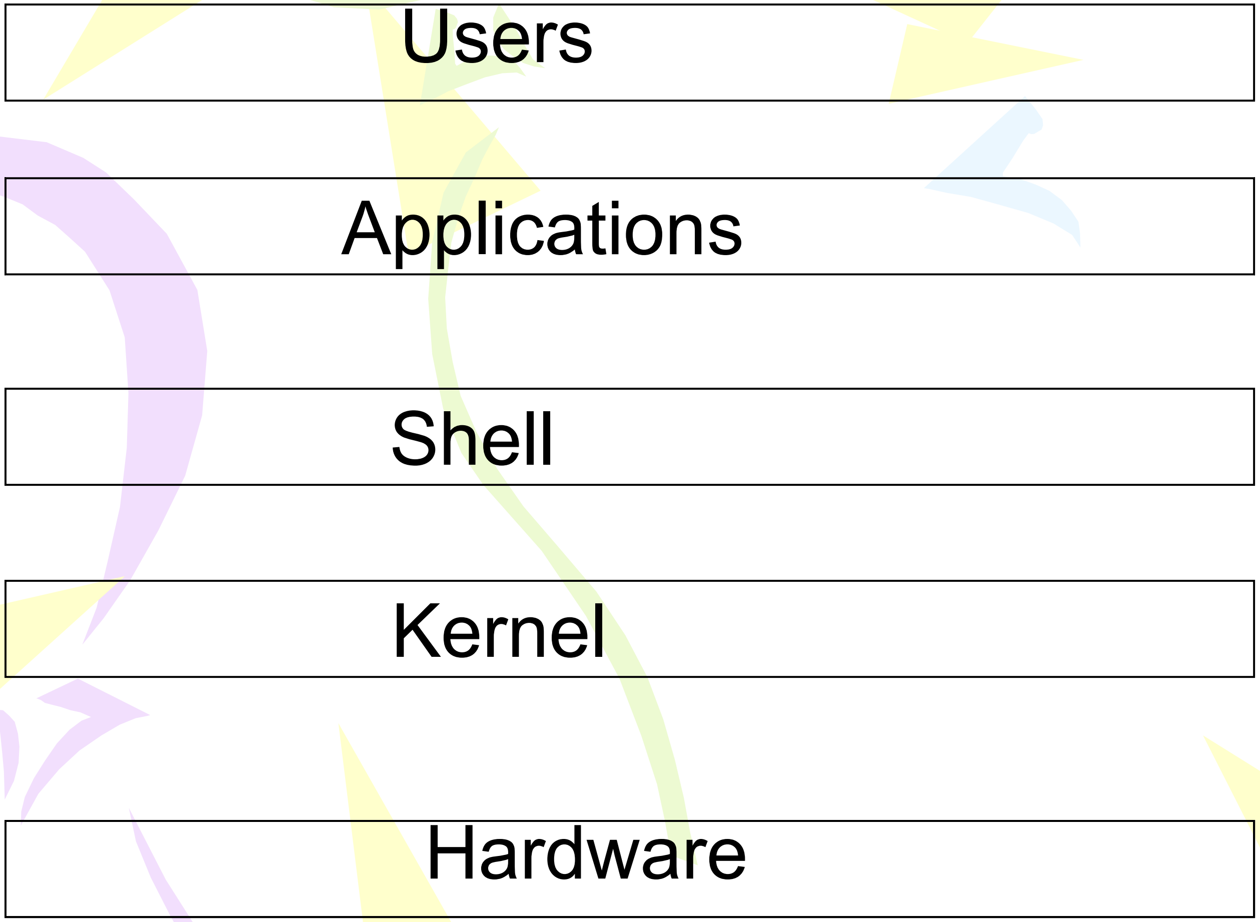
Levels of understanding Operating System?

- Level 0
  - Command level
  - Understanding commands to use the resources and shell programming
- Level 1
  - System programming
  - Understand system calls to use the resources
- Level 2
  - Understanding Operating System Internals
  - How OS manages Resources?
- Level 3
  - Kernel Programming
  - Piece of program written by you becoming a part of kernel

Basic terminologies

- Kernel
- Shell
- Program
- Process

System Structure



Program v/s Process

• Set of instructions	Set of instructions
• Resides in harddisk	Resides in Memory
• Passive entity	Active entity
• Does not consume system resources (except hard disk space)	Consumes various resources during its life time

A Simple C Program

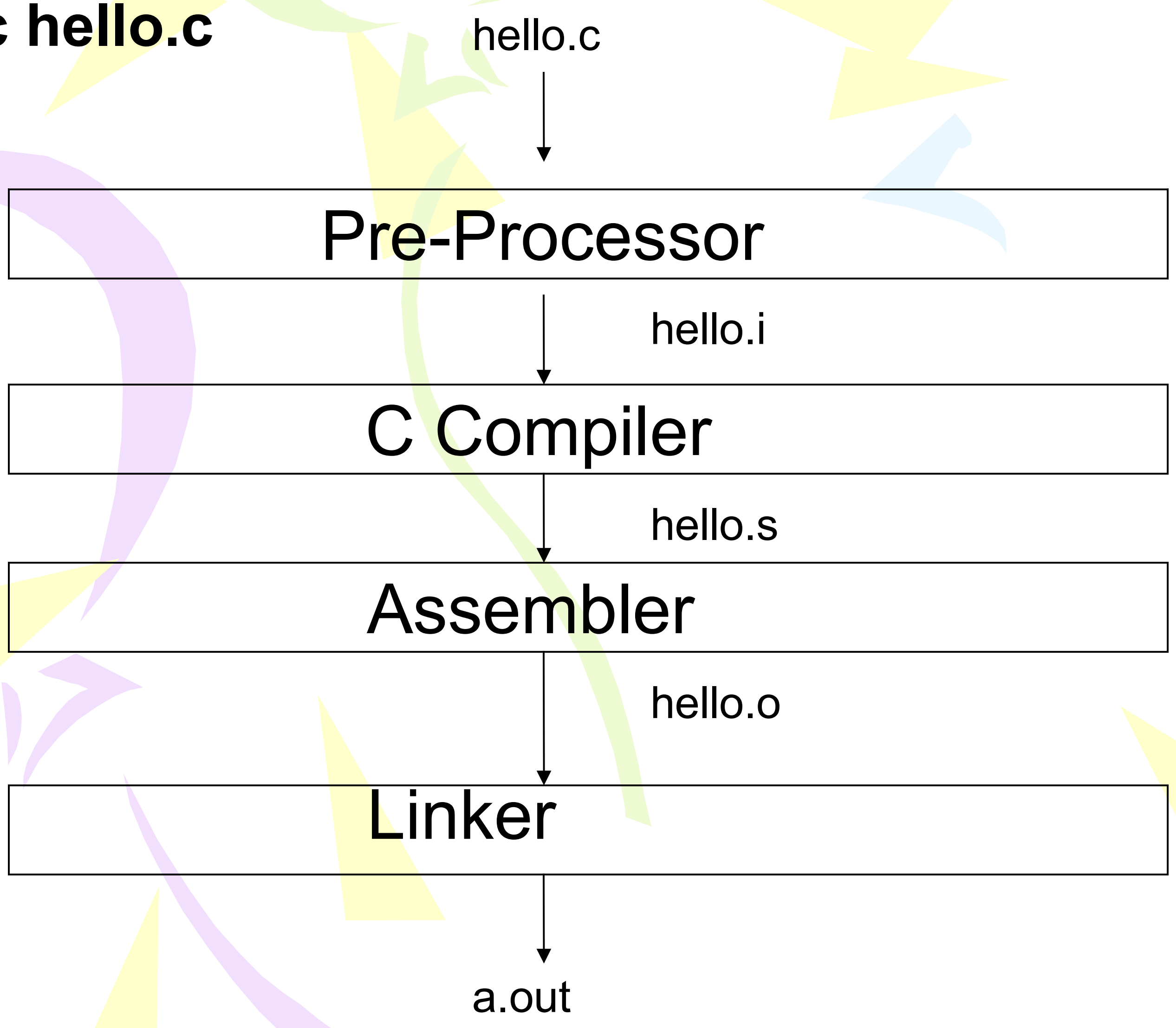
```
/* Filename:       hello.c
   Author:             Brian Kernighan & Dennis Ritchie
   Date written:  ?/?/1978
   Description:   This program prints the greeting
                  “Hello, World!”
*/

#include  <stdio.h>

int main ( void )
{
    printf ( “Hello, World!\n” ) ;
    return 0 ;
}
```

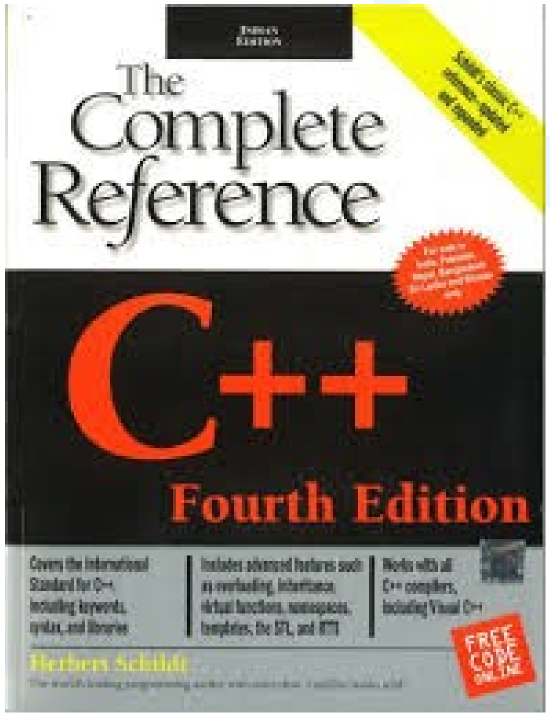
Compilation and Build Process

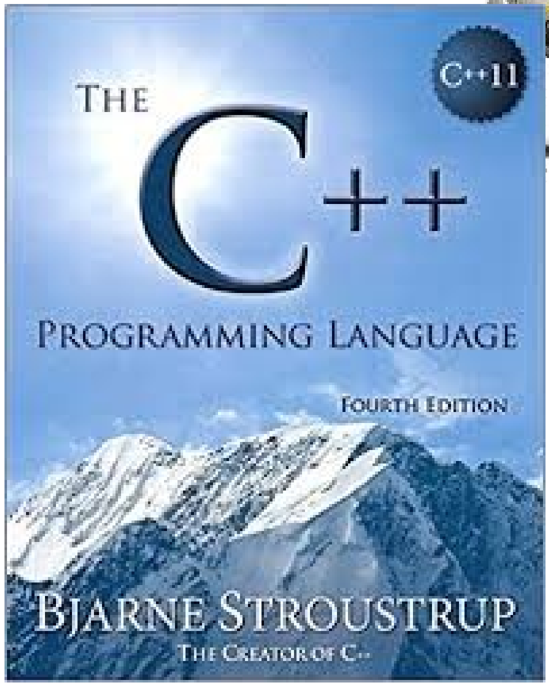
$gcc hello.c

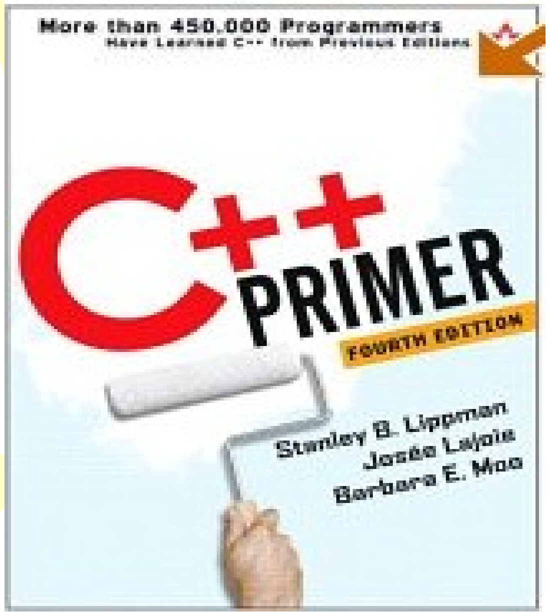


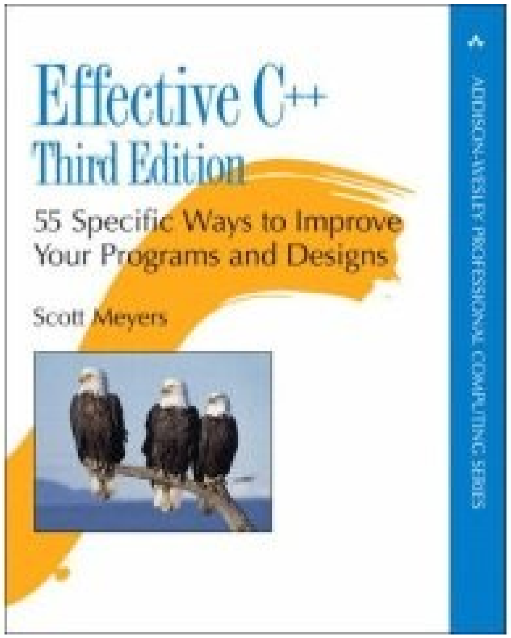
Prerequisites

- Basics of Operating System Concepts
- Working Knowledge of GNU/Linux
  - vi, gcc, cp, mv, mkdir ......
- C Programming Knowledge
  - Data Types, Storage Class, Statements, Loops, Functions, Arrays, Structures and Pointers


The Complete Reference
C++
Fourth Edition
Covers the International Standard for C++, including keywords, syntax, and libraries
Includes advanced features such as overloading, inheritance, virtual functions, namespaces, templates, the STL, and RTTI
Works with all C++ compilers, including Visual C++
Herbert Schildt
FREE CODE ONLINE

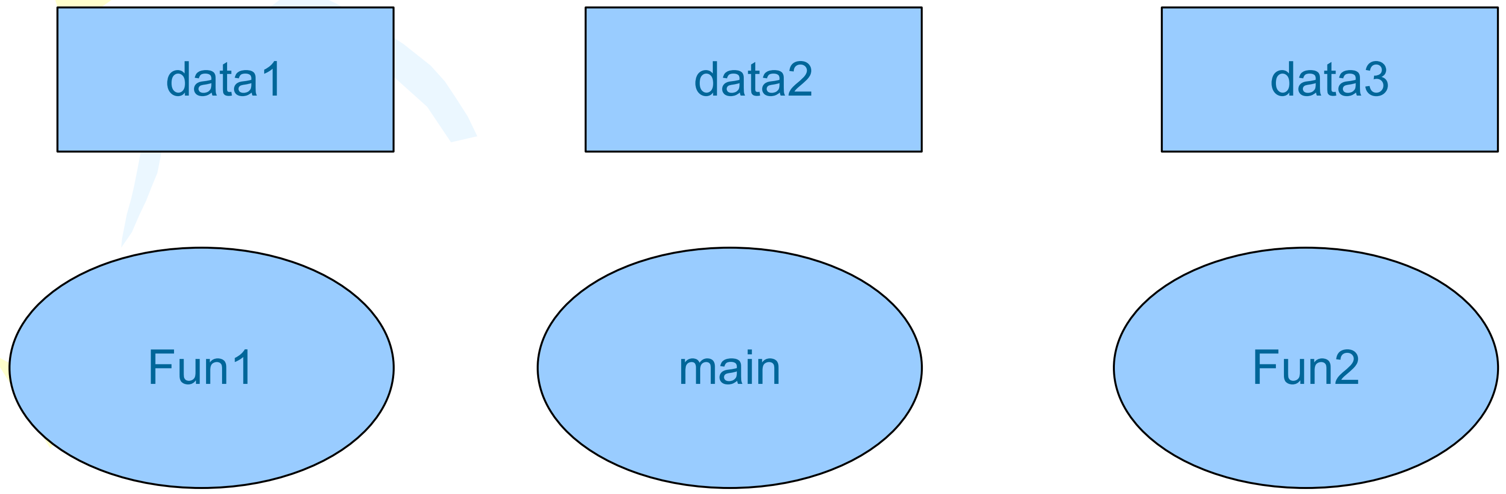

C++11
THE C++ PROGRAMMING LANGUAGE
FOURTH EDITION
BJARNE STROUSTRUP
THE CREATOR OF C++


More than 450,000 Programmers Have Learned C++ from Previous Editions
C++ PRIMER
FOURTH EDITION
Stanley B. Lippman
Josée Lajoie
Barbara E. Moo


Effective C++
Third Edition
55 Specific Ways to Improve Your Programs and Designs
Scott Meyers
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Procedure-Oriented Programming

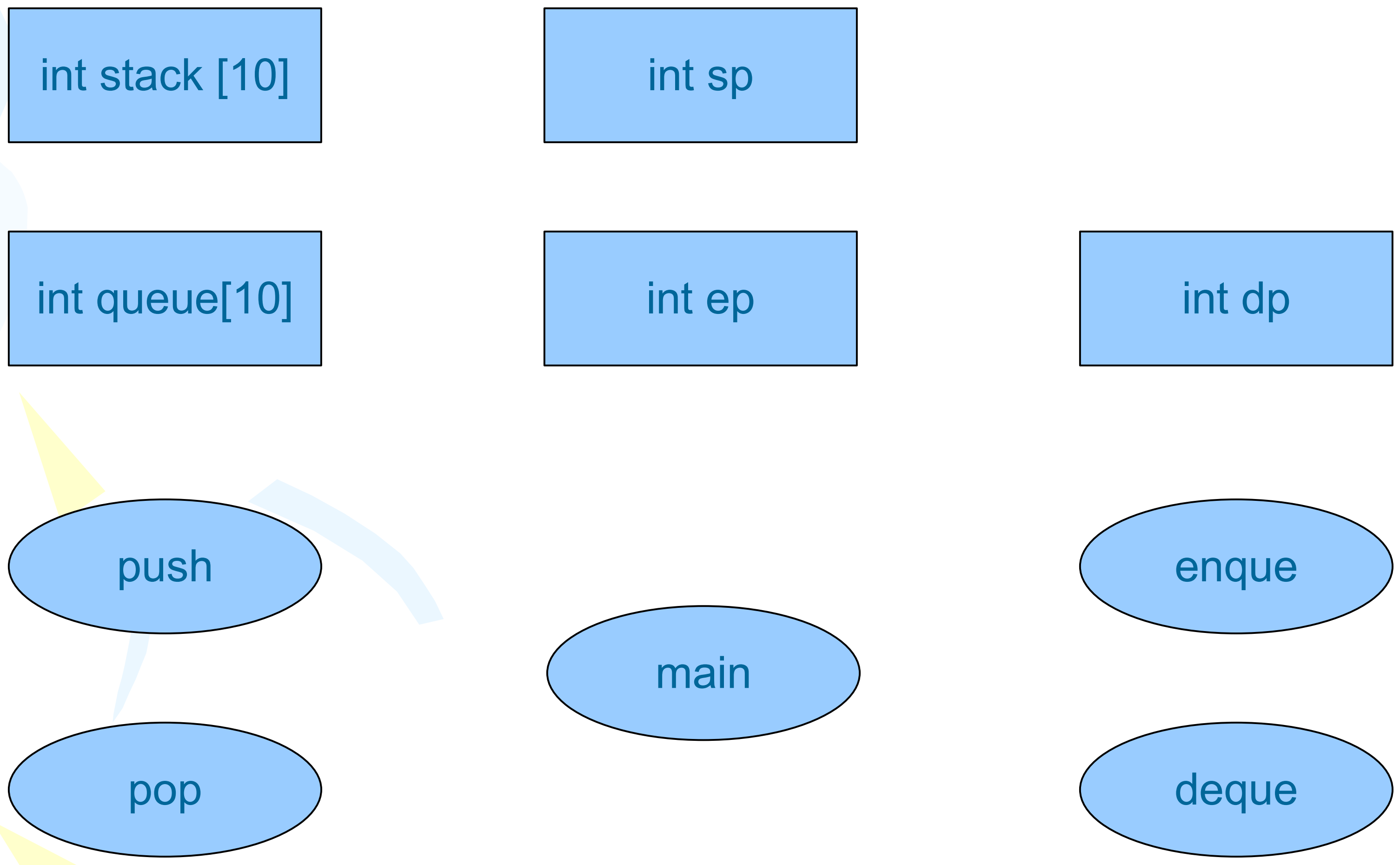
- Problem divided into functions/procedures
- Procedures are dissociated from data
- The code design is centred around procedures



Limitations of Procedure Oriented Programming

- Data is not secure
- It can be manipulated by any procedure
- One function might corrupt the data of another function
- Compilers do not prevent unauthorized functions from accessing/manipulating data
- Operation performed on builtin type cannot be performed on user defined type
- No object code level reusability

Data structure application



C vs. C++

- Procedure Oriented
- Not Suitable for OOD
- Importance given for only functions

Procedure Oriented

Suitable for OOD

Importance given for both functions and data

C++ Features

- Encapsulation
  - Binding data and functions
- Re-usability
  - Inheritance
  - Containership
  - Templates
- Polymorphism
  - Static or Compile Time
  - Dynamic or Run Time

Topics Covered during the Course

- Classes and Objects
- Functions
- Constructors and Destructors
- Dynamic Memory Allocation
- Operator Overloading
- Function Overloading

Inheritance

Virtual Functions

Exception Handling

Templates

STL

Multithreading

Structure in C and C++

- only data
- access specifier cannot be used
- Size of empty structure is zero

Both data and functions

Access specifier can be used

Size of empty structure is one

Classes

- User defined data type
- Similar to structure
  - Can have both data and functions
- Class is a template
- In C++ all data types are implemented as classes
  - Both built in and user defined data type

Classes contd.

- Access specifiers can be used inside the class definition

```
class emp
{
    private:
        int emp_id;
    public:
        void set(int id)
        {
            emp_id = id;
        }
};
```

Class Name

Access Specifier

Data Members

Methods or Member Functions

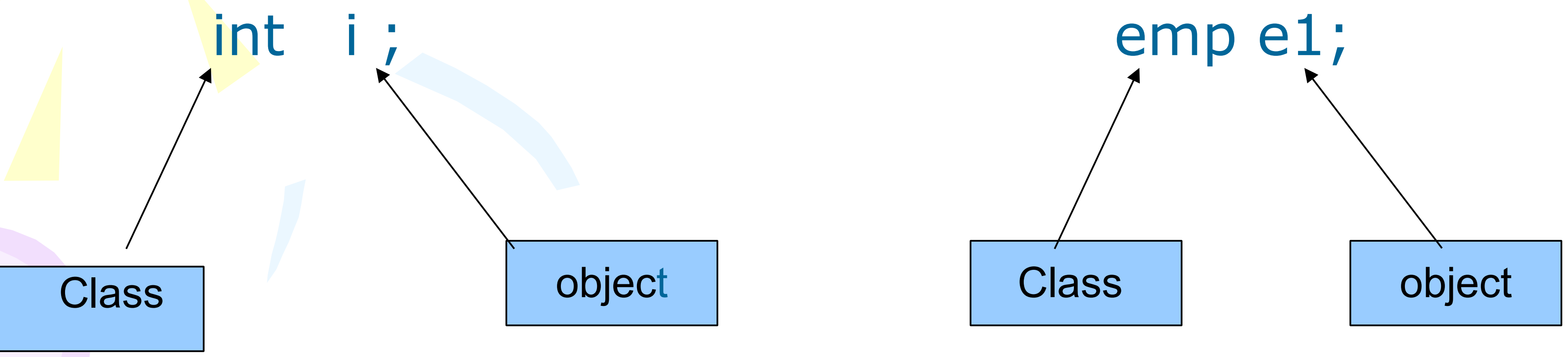
Structure vs. classes

- By default members of structures are public

By default members of classes are private

Objects

- Instance of a class
- Compiler needs to allocate memory for objects
  - ie. it occupies space

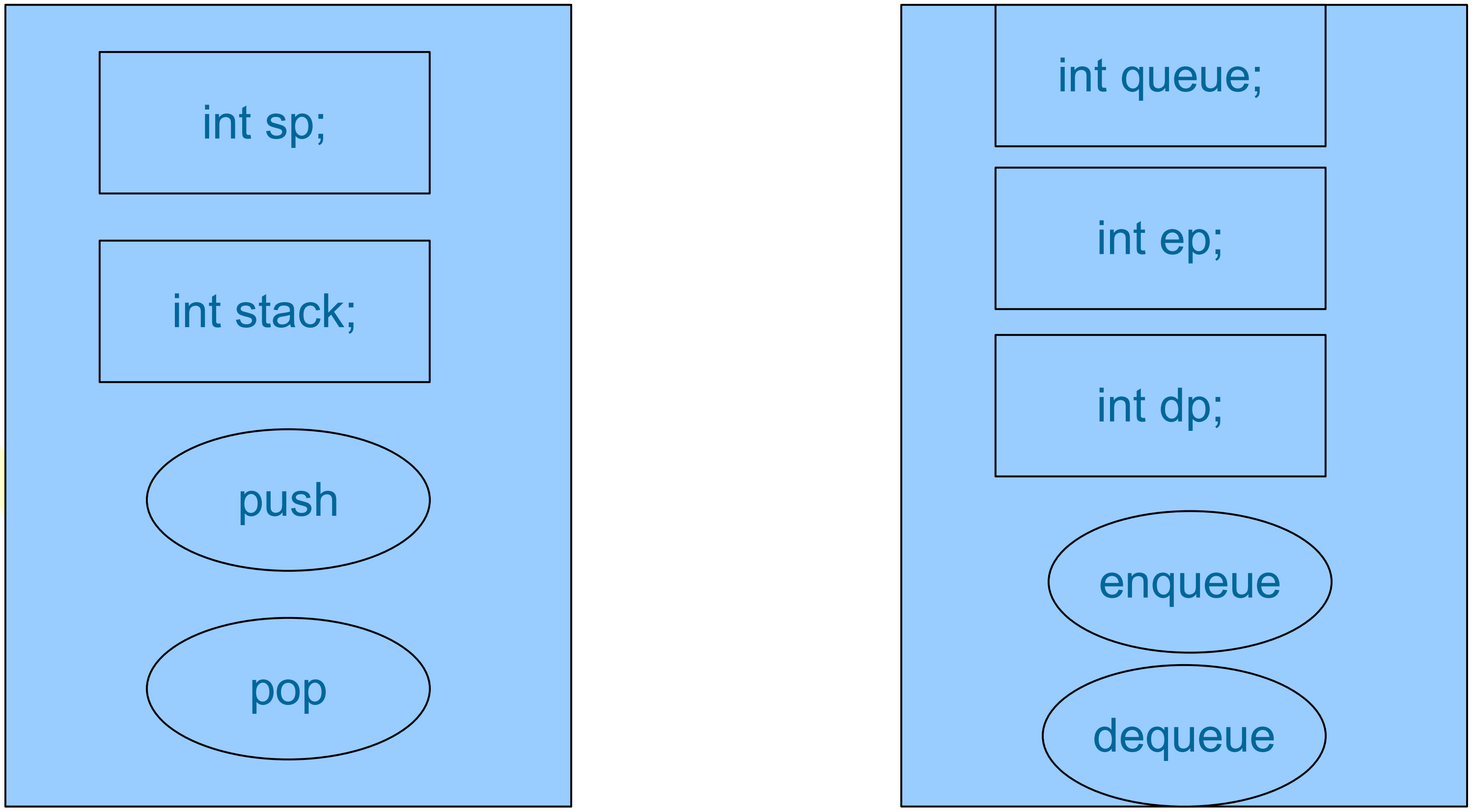


```
int   i ;                                   emp e1;
```

Objects contd.

- Global objects
  - Memory allocated by compiler
- Local objects
  - Instruction generated by compiler
- Size of an object
  - Generally summation of size of all data members
- Object size of an empty class is one

Encapsulation



stack

queue

Key Terminologies

- Program
- Process
- Class
- Object
- methods
- Encapsulation
- Inheritance
- polymorphism
- templates

Excercise

- Write a program to demonstrate the limitation of C program
- Write a program to demonstrate encapsulation feature in c++
- Rewrite generic stack application in c++ by providing data security

Functions

Functions

- Non Member Functions
- Member Functions
- Static Member Functions
- Friend Functions
- Function Overloading
- Strict Prototyping

Non Member Functions

- Functions not associated with any class
- Used to delegate responsibility
- May or may not accept arguments
- May or may not return values

Member Functions

- Defined inside a class
- Declared inside a class and definition can be outside the class
- Member function can also be
  - static
  - friend
  - const
  - inline

Static Member Functions

- Function which can access only static data members
- Static data member is common for all the objects
- Non static member function can access
  - Static data member
  - Non static data member

Friend Function

- A Class can consider a non member function as its friend
- Friend function can access private data member of that class

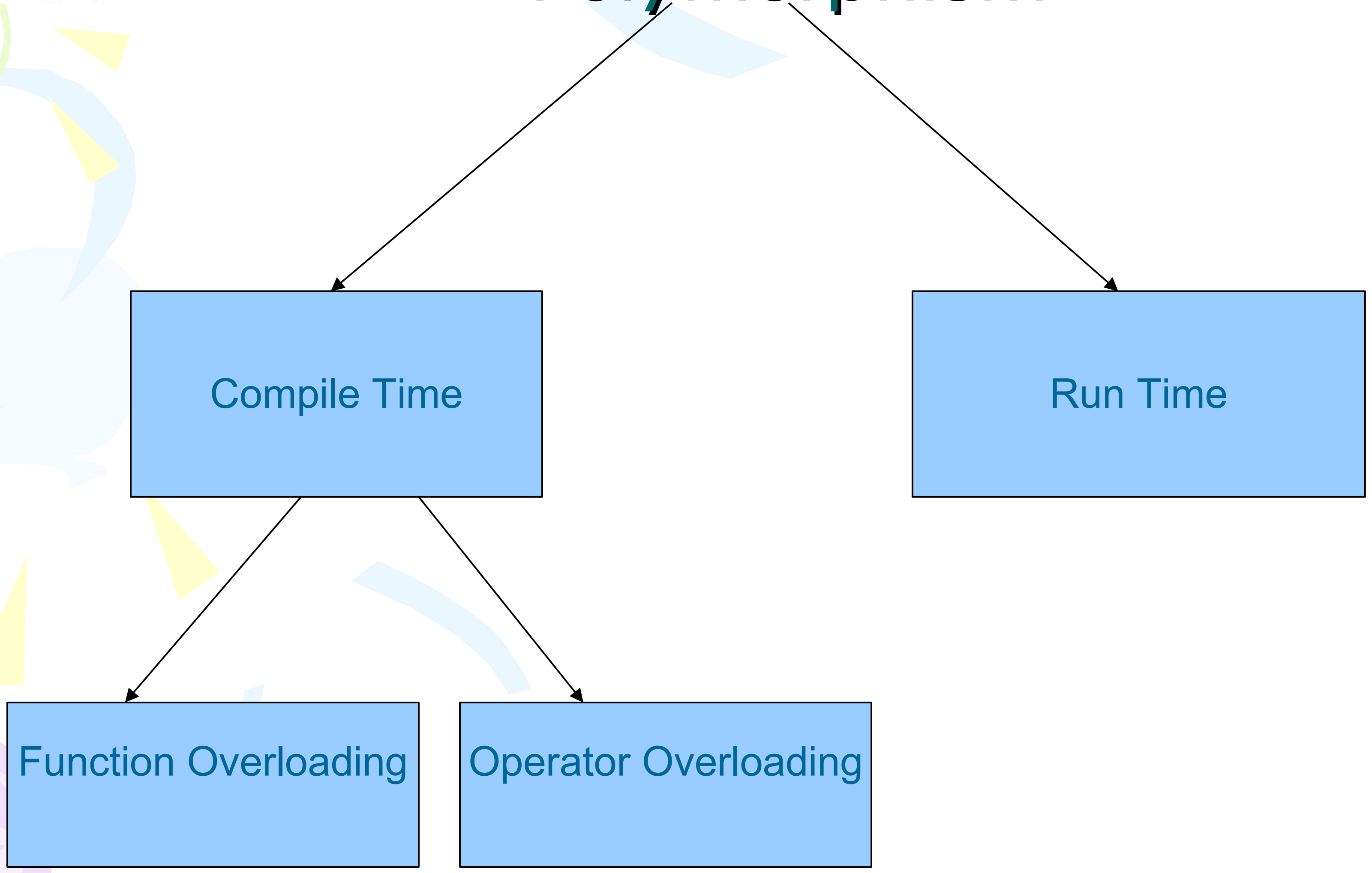
Const member function

- Read only method
- Methods are specified as constants by suffixing the prototype and the function definition with const keyword

```
class book
{
        int isbn;
        public:
            void get(void) const;
};
void book::get() const
{
}
```

Polymorphism

Polymorphism



Function overloading

- Polymorphism feature
- Multiple definitions can have same name
- Both member and non member functions can be overloaded
- These functions should differ in
  - Number of arguments to be passed
  - Order of arguments to be passed
  - Type of arguments

Default Values for Formal Arguments of Functions

- One can specify default value for some or all the formal arguments of a function
- If no argument is passed for an argument, default value specified for it is passed
- If parameters are passed in the normal fashion
  - The default vale is ignored
- An example

```
int add(int n1, int n2, int n3 = 40);
```

Constructors and Destructor

Constructors

- The constructor gets called automatically for each object that has just created
- It appears as member function
- It has the same name as that of the class
- It may or may not take parameters
- It does not return any value
- The prototype of constructor looks like this
  <class name> (<parameter list>);

Constructor contd..

- The compiler embeds a call to the constructor for each object when it is created

  plot p1; // memory allocated for the object

  p1.plot( ); // compiler calls constructor implicitly

Types of Constructor

- The Default constructor or Zero argument constructor
- The parameterized constructors or N argument constructor
- The copy constructor

Destructor

- The destructor gets called for each object that is about to go out of scope
- It appears as a member function of each class whether we define it or not
- It has the same name as that of the class but prefixed with a tilde sign
- It does not take parameters
- It does not return anything
- The prototype is
  ~<class name> ( );

Destructor contd..

- void fun ( )
  ```
  {
        plot p;

             p.~plot(); // implicitly called by
  compiler
  }
  ```

Operator Overloading

Operator Overloading

- C++ lets us redefine the meaning of the operators when applied to objects of class type
- Overloading an operator means programming an operator to work on operands of types it has not yet been desinged to operate
- The '+' operator can work on operands of type char, int, float, and double
- by default '+' operator cannot add two objects of userdefined data type ex:

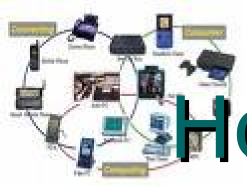
```
    plot p1,p2,p3;
    p3 = p1 + p2;
```

Overloading Operators – The Syntax

- class <class_name>

```
{
    <return_type> operator <op>
(<arg_list>);

}
```



How does the compiler interpret the operator-overloading functions

- Consider a statement
  p3 = p1 + p2; // p1, p2, p3 are all objects of plot
- The above statement is interpreted as
  p3 = p1.**operator+** (p2) ;
  - If the operator overloaded function has been defined as member function
- If the operator overloaded functions is defined as non member function then,
  p3 = **operator+** (p1 , p2);

Rules for Operator Overloading

- New operator cannot be created
  - Such as **, the following piece of code will generate compiler error

```
class test
{
    public:
        void operator **();
};
```

Rules for Operator Overloading

- Meaning of existing operators cannot be changed
  - Any operator overloading function should take at least one operand of the class of which it is a member or friend

```
Class test
{
 public:
    friend int operator + ( int ,int); //Compile
Time Error
};
```

Rules for Operator Overloading

- Some of the existing operator cannot be overloaded
  - :: (scope resolution operator)
  - . (member selection)
  - .* (member selection through pointer to member)
  - ?: (conditional operator)
  - sizeof (finding the size of values and types)
  - typeid (finding the type of object pointed at)

Rules for Operator Overloading

- Number of arguments that an existing operator takes cannot be changed
  - Operator overloading functions should take the same number of parameters that the operator being overloaded ordinarily takes. For ex, the division operator takes two arguments. Hence the following class definition causes a compile-time error

```
class test
{
 public:
     void operator /();
}
```

Rules for Operator Overloading

- Overloaded operators cannot take default arguments

```
class test
{
    public:
        void operator / (int = 0);
};
```