Lecture 1
Introduction to OOP

Khola Naseem
khola.naseem@uet.edu.pk

Assessment Plan (Tentative)

- Mid(30%)
- Final(40%)
- Assignment(10%)
- Quiz(20%)

Text and Reference Books

- C++ How to Program, Deitel & Deitel, 5th Edition, Prentice Hall.
- Object Oriented Programming in C++ by Robert Lafore.
- C++ Programming From Problem Analysis To Program Design, 5th Edition, D.S.Malik
- Object Oriented Software Construction, Bertrand Meyer's
- Object-Oriented Analysis and Design with applications, Grady Booch et al, 3Rd Edition, Pearson, 2007

Attendance policy

- 75% required at the end to sit in Final Exam.
- Attendance will be taken at any moment.
- No compensation for attendance.

Credit: Khola Naseem

Homework Policy

- All homework assignments must be done individually or as directed
- Cheating
  - Helping others, getting help, looking up websites for solutions etc.
  - Any deviation from the above rule will be considered cheating and will be subject to the UET academic dishonesty policy

Credit: Khola Naseem

Late Submission Policy

- Late Submission Policy
- Late Submissions : Extensions may be permitted under extraordinary circumstances
- Contact the instructor at least 2 days before the deadline

Credit: Khola Naseem

Outline

Introduction

- Conditional statement:
  - A bank gives loan in two situations
    - If customer's age is more than 30 and salary is more than 30000
    - If customer's age is less than or equal to 30 and income is more than or equal to 20000
    - Program: Ask user to enter age and income. Check whether he\she is eligible to get loan

Introduction

- Repetitive(Looping) control structure
  - Types
- Function
  - What?
  - Why?
  - Function definition and declaration

Introduction

- Function
  - Function definition

```
return_type function_name(parameter_list)
{
  // Code for the function...
}
```

  - Example1:

```
void show()
{
    cout<<"Hello world\n";
}
```

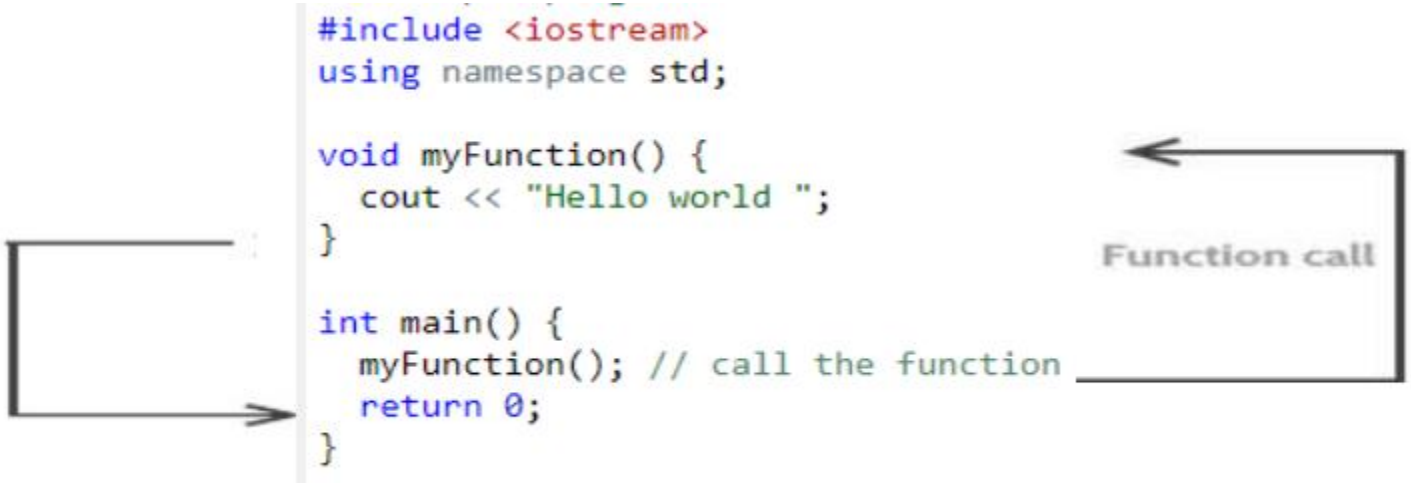
  - Example2:

```
void sum()
{
     int a=5;
     int b =6;
     cout<<a+b;
}
```

Credit: Khola Naseem

Introduction

- Function
  - Calling a function



Credit: Khola Naseem

Introduction

- Function
  - Declaration (Function Prototype):

```
return_type function_name(parameter_list);
```

  - Example

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

void myFunction(); // Function declaration

int main() {
  myFunction();

  return 0;
}

// Function definition
void myFunction() {
  cout << "Hello world "<<endl;
}
```

Credit: Khola Naseem

Introduction

- Function
  - Declaration (Function Prototype):

```
return_type function_name(parameter_list);
```

  - Example

```
#include <iostream>
#include <cmath>
using namespace std;
int sum(int a, int b);

int main(){
   int x, y;
   cout<<"enter first number: ";
   cin>> x;

   cout<<"enter second number: ";
   cin>>y;

   cout<<"Sum of these two :"<<sum(x,y);
   return 0;
}

int sum(int a, int b) {
   int s = a+b;
   return s;
}
```

Credit: Khola Naseem

Introduction

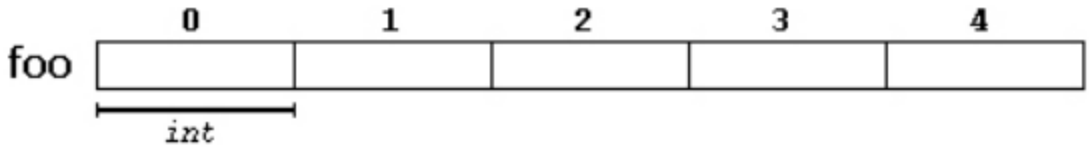
- Function
  - Parameter list

```
void functionName(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

  - functions with arguments, we have two ways to call them
    - Call by Value
    - Call by Reference

Introduction

- **Arrays**
- C++ treats the name of an array as constant pointer which contains base address i.e. address of first location of array.
- For example, a five element integer array foo may be logically represented as



- Declaration
- dataType arrayName [Size];
- Initialization
  - dataType arrayName [4]={e1,e2,e3,e4};

Credit: Khola Naseem

Introduction

- **Types of Arrays**
  - Single dimensional array
  - Two dimensional array
  - Multi dimensional array

Introduction

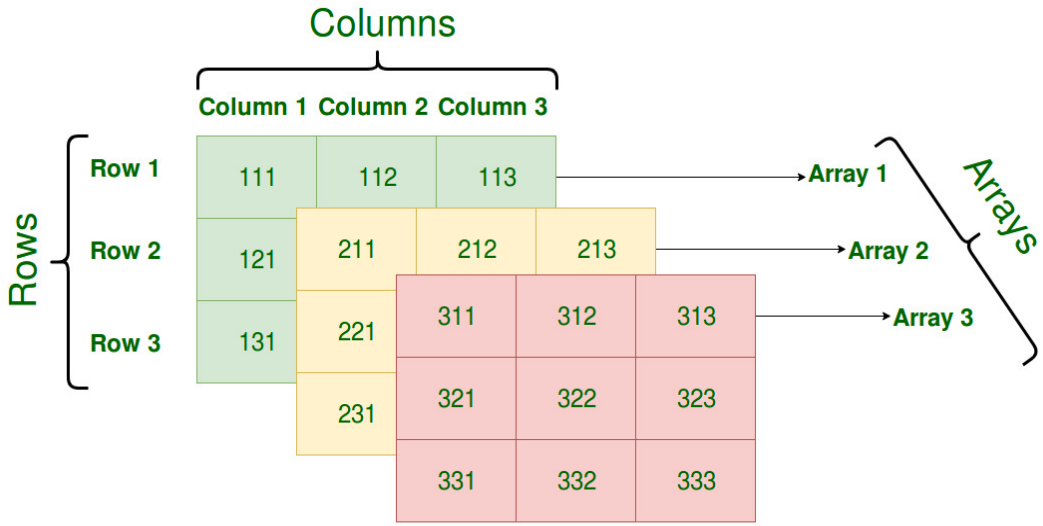
- **Arrays:**
  - Declaration
  - Examples:
    - Onedimensional array
      - int one_d[10];
    - Two dimensional array
      - int two_d[10][20];
    - Three dimensional array:
      - int three_d[10][20][30];
- 2d Array representation:
  - int x[3][3];

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Credit: Khola Naseem

Introduction

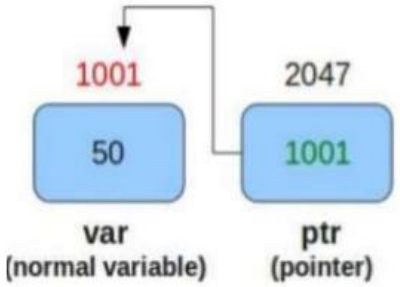
- Multi dimensional arrays:
  - Declare
  - Access the elements



Credit: Khola Naseem

Introduction

- **Pointers:**
  - A pointer is a variable that holds the memory address of another variable of same type.
  - This memory address is the location of another variable where it has been stored in the memory.
  - It is represented by *(asterisk)
  - Address operator is used to access the memory address of a variable and store it in a pointer.
  - DECLARATION:
    - Datatype *variable_name;
  - Initialization:
    - int *ptr = &var;



Credit: Khola Naseem

Introduction

- **Pointers:**
  - Indirection/ Deference Operator *

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

int main()
{
  int num=10;
  int *ptr;
  ptr=&num;
  cout << " num = " << num << endl;
  cout << " &num = " << &num << endl;
  cout << " ptr = " << ptr << endl;
  cout<< " *ptr = " << *ptr << endl;
}
```

  - Output:

```
num = 10
&num = 0x7e1428993b0c
ptr = 0x7e1428993b0c
*ptr = 10
```

Credit: Khola Naseem

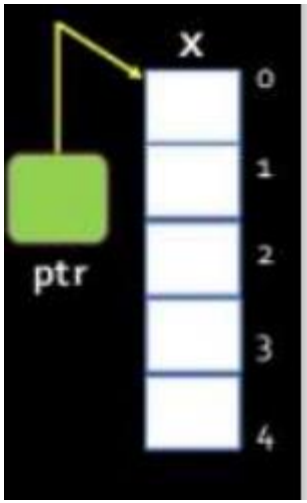
Introduction

- **Pointers and arrays**
- C++ treats the name of an array as constant pointer which contains base address i.e. address of first location of array.
- For eg. int x[10];
- Here x is a constant pointer which contains the base address of the array.
- We can also store the base address of the array in a pointer variable.
- It can be used to access elements of array, because array is a continuous block of same memory locations.

```
int x[5];

int * ptr=x; // ptr will contain the base address of x
```

Introduction

- **Pointers and arrays**
- If we will increment ptr: (ptr++)
- It will change its value to the next integer in the array
- ptr=ptr+2;// ptr will store the address of 3 rd element of the array



Credit: Khola Naseem

Introduction

- **Pointers and arrays**
- Access the elements of the array:

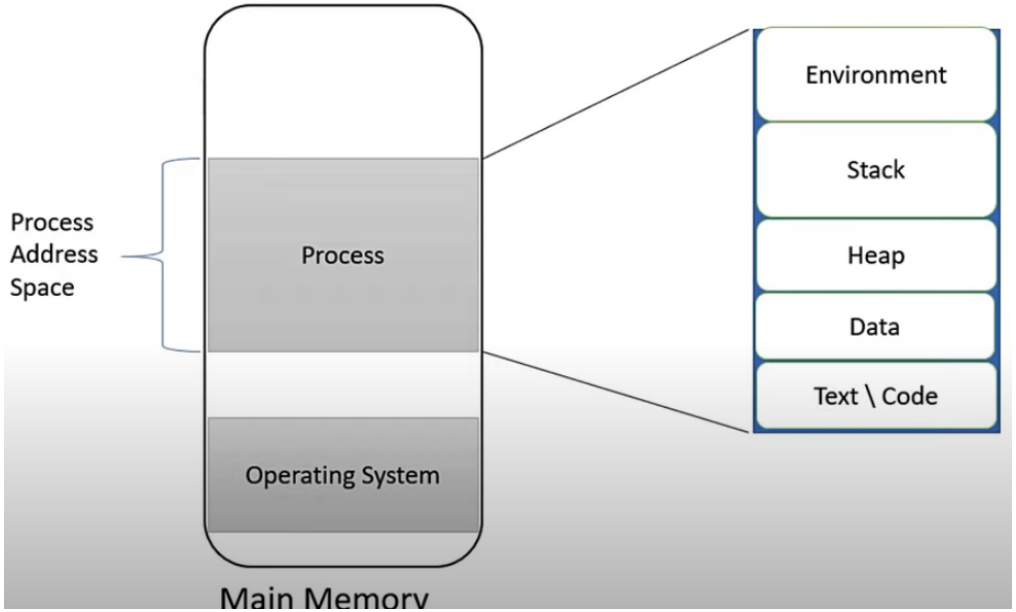
```
#include <iostream>
using namespace std;

int main()
{

    int value[5] = {1,2,3,4,5};
    int *ptr = value;
    for(int i=0;i<5;i++)
    {
        cout<<*ptr++<<endl;
    }


    return 0;


}
```

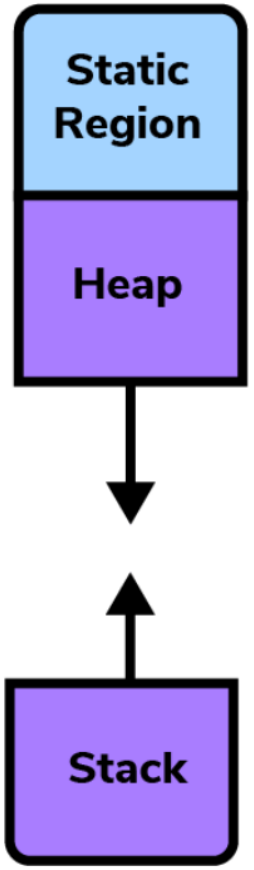
- Output:

```
1
2
3
4
5
```

Credit: Khola Naseem

Memory management with pointers





Credit: Khola Naseem

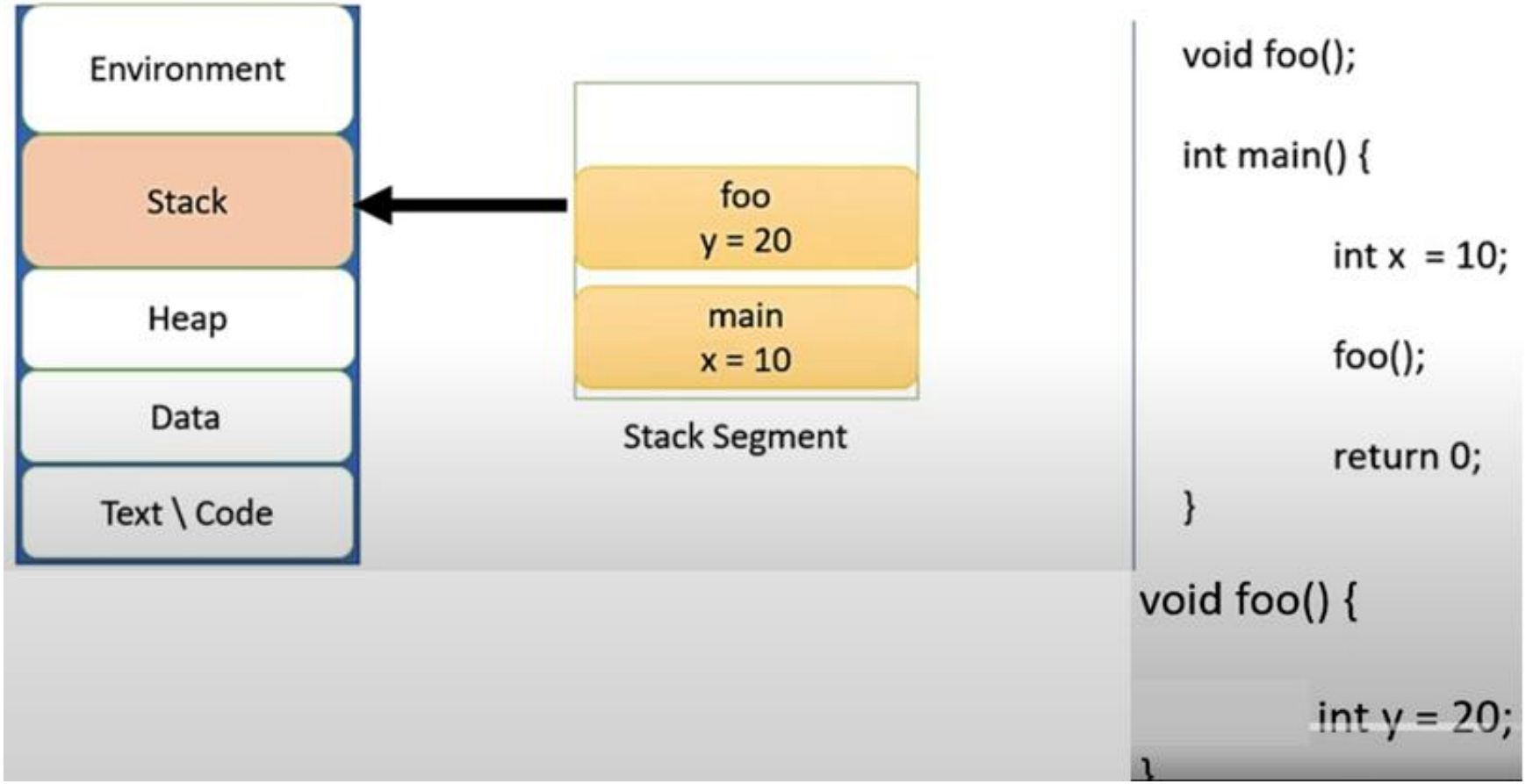
Memory Segment of a C++ program

- The process of allocating and deallocating memory is known as memory management.
- C++ memory is divided into four parts which are listed as follows:
  - Code/text section: It holds the compiled code of the program.
  - Data section: static variabl3 & global variable(They remain in the memory as long as program continues.)
  - Stack: It is used for holding return addresses at function calls, arguments passed to the functions, local variables for functions. It also stores the current state of the CPU.
  - Heap: It is a region of free memory from which chunks of memory are allocated via dynamic memory allocation functions.

Credit: Khola Naseem

Memory Segment of a C++ program

- Stack:



Credit: Khola Naseem

Memory management with pointers

- dynamically allocates memory from the free store/heap/pool, the pool of unallocated heap memory provided to the program.
- There are two unary operators:
  - new and delete: that perform the task of allocating and deallocating memory during runtime.
  - malloc and free

Memory management with pointers

- Example:

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

int main()
{
    int *a=new int;
    *a=100;
    cout<<a<<endl;
    cout<<*a<<endl;
    delete a;
    cout<<*a;


}
```

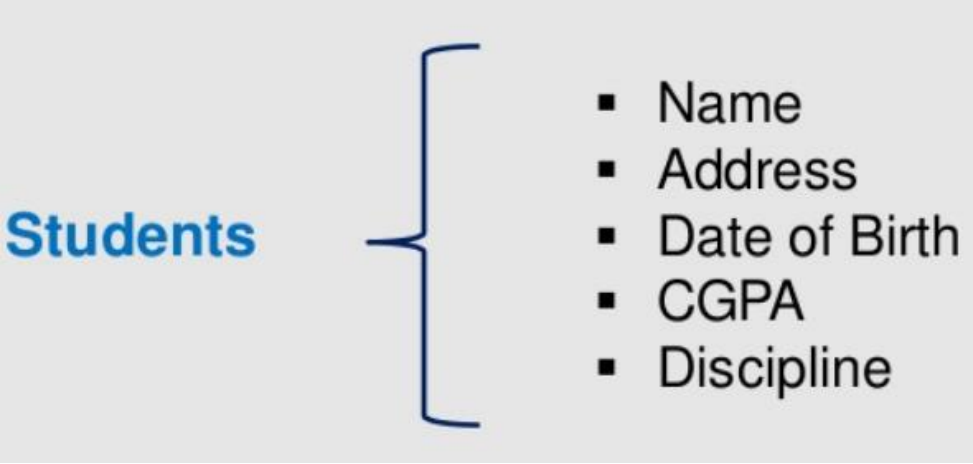
- Output:

```
0x34f4980
100
0
```

Credit: Khola Naseem

Structures

- It is often desirable to group data of different types and work with that grouped data as one entity.
- We now have the power to accomplish this grouping with a new data type called a structure.
- A user defined data type is called a structure, class etc.
- Example:



Credit: Khola Naseem

Structures

- Declaration:
  - We declare a structure using the keyword struct followed by an identifier

```
struct struct_name
{
     // struct members
}
```

  - The structure members (variables) are defined with their type and variable names inside the open and close braces { and }
  - Example:

```
struct Student
{
    char stuName[30];
    int stuRollNo;
    int stuAge;
};
```

Structures

- When a structure is created, memory is not allocated. Memory is only allocated after a variable is added to the struct.
- Creating Struct Instances
- In the above example, we have created a struct named Student. We can create a struct variable as follows:
  - Student s1;
- The s1 is a struct variable of type Student. We can use this variable to access the members of the struct.

Initializing Structures

- Like normal variable structures can be initialized.
- 1st way:
- Example :
  - Person p1 = {“ABC", 20, 1000};
- 2nd way:

```
Person p1;
p1.name="ABC";
p1.age =25;
p1.salary=1000;
```

Initializing Structures

- Array of structure:
- student s[10];
  - s[0].name;
  - s[1].name;
  - ..
  - s[9].name

Difference between structured and unstructured programming

Unstructured programming:

- Unstructured Programming is a type of programming that generally executes in sequential order i.e., these programs just not jumped from any line of code and each line gets executed sequentially.
- the program should be written as a single continuous block without any breakage
- Such programs cannot be used for medium and complex projects. Instead, they can be used for small and easier projects
- Example

Difference between structured and unstructured programming

Structured Programming:

- Structured Programming is a type of programming that generally converts large or complex programs into more manageable and small pieces of code.
- These small pieces of codes are usually known as functions or modules or sub-programs of large complex programs.
- It is known as modular programming .
- Such programs can be used for small and medium-scale projects and also for complex projects.
- Example

Difference between structured and object oriented programming

Object Oriented Programming:

- The object oriented programming allows constructing a program using a set of objects and their interactions.
- An object is any entity that has states and behaviors.
- States represent the attributes or data of an object, whereas the methods represent the behaviors of objects.
- These objects interact with other objects by passing messages.
- This model is based on real life entities that focuses on by whom task is to be done rather than focusing on what to do.

Difference between structured and object oriented programming

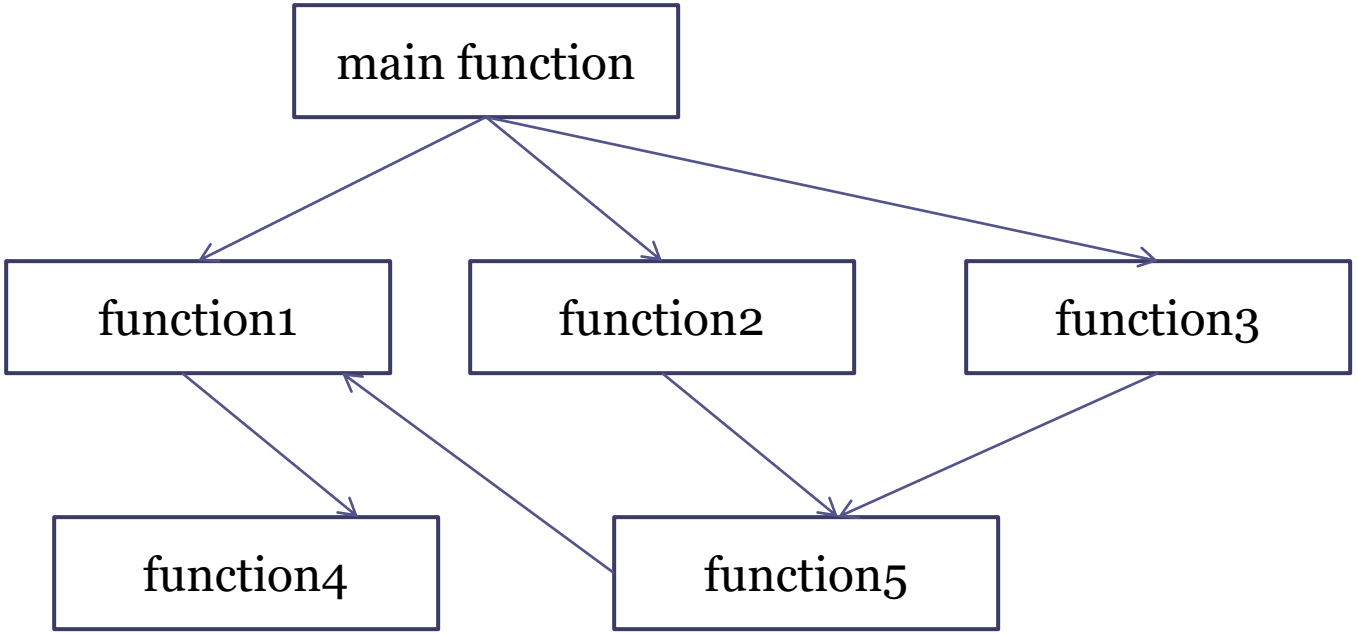
- **Object Oriented Programming:**
  - Data is more secure
  - Provide access specifiers
  - Objects communicate via message passing
  - Easier to modify
  - Can solve complex problems easily.
  - reusability

Difference between structured and object oriented programming

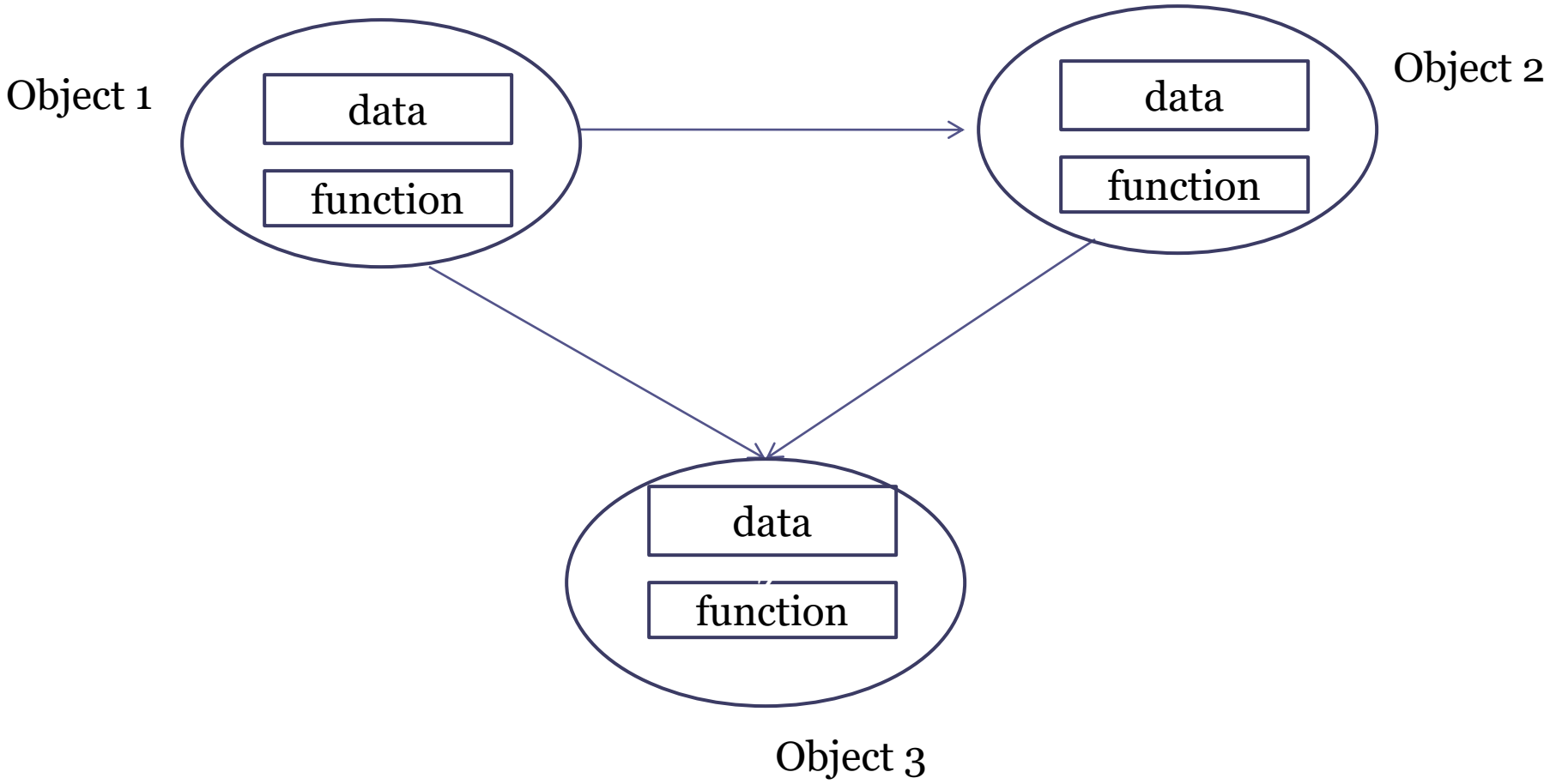
Important Features of OOP

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Difference between structured and object oriented programming



Difference between structured and object oriented programming



Objects send and receive messages to invoke actions

Credit: Khola Naseem

Important Features of OOP

- Abstraction
  - Showing only the essential features and hiding the unnecessary features
  - Hide implementation but provide functionality

Important Features of OOP

- Encapsulation
  - The process of bundling the data with the methods that operate on that data and inside a single class
  - Classes provides us with this feature – Encapsulation
  - Can be implemented using access specifiers

Important Features of OOP

- Inheritance
  - Feature that enables the characteristics or properties of a parent to reach its child
  - C++ supports inheritance
  - A class can inherit one or more classes
  - Inherited class is called as parent class or super class or base class
  - Class that inherits a parent class is called as child class or sub class or derived class

Important Features of OOP

- Polymorphism
  - Poly – Many
  - Morph – Form
  - Polymorphism is the characteristic that enables an entity to co exist in more than one form
  - C++ supports function overloading and operator overloading to implement polymorphism

Object oriented programming

- Classes
  - Objects
    - States and behavior
- Example:
  - Assemble a bicycle
  - In preparing a dinner, you might consult a recipe that specifies a list of ingredients and procedures for combining them correctly
- From a programming perspective, a class can be considered a construction plan for objects and how they can be used. This plan lists the required data items and supplies instructions for using the data

Credit: Khola Naseem

Object oriented programming

- in real life, a **car** is an **object**. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.
- C++ is an object-oriented programming language.
- A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.

Object oriented programming

- After one or more objects from this plan, or class, have been constructed, they can then be operated on only in ways defined by the class
- connection between the recipe and a C++ class

Recipe Name: Gary's Sardine Spread

Ingredients:

Measure	***Contents***
1 can	Boneless and skinless sardines
2 stalks	Celery
1/4 medium	Red onion
1 tablespoon	Mayonnaise
1/4 cup	Parsley
dash	Olive oil
splash	Red wine vinegar
dash	Salt
dash	Pepper

Method of Preparation:

Finely shred the sardines using two forks
Finely dice the celery and onion and mix well with sardines
Add olive oil and mix well
Add mayonnaise and mix well
Add red wine vinegar and mix well
Finely dice the parsley and mix well
Salt and pepper to taste

```
// Class declaration section
Class Name: AverageofTwoNumbers
    // A list of data items to use (the parts list)
   Type      Name
   double    firstNumber
   double    secondNumber
    // A list of necessary methods (prototypes)
    double assignValues(double, double);
    double calculateAndDisplay(double, double);
// Class implementation section (the instructions)
    Code for the two methods listed previously
```

Credit: Khola Naseem

Object oriented programming

- A class defines both the Data members and Member function(the types of operations that can be performed on the data)

$$C++\ class = data + functions$$

- Declaration:

```
// class declaration section
class className
{
  data declarations
  function prototypes
};

// class implementation section
function definitions
```

- 2nd methods

```
// class declaration section
class className
{
  data declarations
    function definitions
};
```

Credit: Khola Naseem

Object oriented programming

- A class defines both the Data members and Member function(the types of operations that can be performed on the data)

C++ class = data + functions

- Declaration: Example:

```
class Date
{
  private:
     int month;
     int day;
     int year;
  public:
     Date(int = 7, int = 4, int = 2012);
     void setDate(int, int, int);
     void showDate();
};
```

OR

```
class Date
{
  private:
     int month;
     int day;
     int year;
  public:
     Date(int = 7, int = 4, int = 2012) { }
     void setDate(int, int, int)
                                  { }
};
```

Object oriented programming

- Classes

```
class Box {
    public:
        float length;
        float breadth;
        float height;

        double calculateArea(){
            return length * breadth;
        }

        double calculateVolume(){
            return length * breadth * height;
        }

};
```

- **C++ Objects:**
- When a class is defined, only the specification for the object is defined; no memory or storage is allocated.
- To use the data and access functions defined in the class, we need to create objects.

Credit: Khola Naseem

Object oriented programming

- Classes
- **C++ Objects:**
- When a class is defined, only the specification for the object is defined; no memory or storage is allocated.
- To use the data and access functions defined in the class, we need to create objects.
- A objects of a class can be defined in any function of the program. We can also create objects of a class within the class itself, or in other classes.
- **Example:**
  - **Classname ObjectVariableName;**
    - **Box box1;**

Object oriented programming

- Classes
- **C++ Objects:**

```
class Box {
    public:
        float length;
        float breadth;
        float height;

        double calculateArea(){
            return length * breadth;
        }

        double calculateVolume(){
            return length * breadth * height;
        }

};

int main(){
    // create objects
    Box box1;  // one object

}
```

Credit: Khola Naseem

Object oriented programming

- Access data member and member functions:
  - We can access the data members and member functions of a class by using a . (dot) operator. For example,
    - box1.length=3.44;
    - box1.calculateVolume();

```
class Box {
    public:
        float length;
        float breadth;
        float height;

        double calculateArea(){
            return length * breadth;
        }

        double calculateVolume(){
            return length * breadth * height;
        }

};

int main(){
    // create objects
    Box box1;  // one object
    box1.length=3.55;
    box1.breadth =4.55;
    box1.height =5.6;
    cout<<"Area of the box is "<<box1.calculateArea()<<endl;
    cout<<"Volume of the box is "<<box1.calculateVolume();

}
```

Credit: Khola Naseem

Access specifier or Access modifier:

- The access modifiers of C++ are
  - Public
  - Private
  - Protected
- Data hiding is one of the key features of object-oriented programming languages such as C++.
- Data hiding refers to restricting access to data members of a class. This makes it impossible for other functions and classes to manipulate class data.
- However, it is also important to make some member functions and member data accessible so that the hidden data can be manipulated indirectly.
- The access modifiers of C++ allows us to determine which class members are accessible to other classes and functions, and which are not.

Access specifier or Access modifier:

- **Public access modifier:**
  - The public keyword is used to create public members (data and functions).
  - The public members are accessible from any part of the program.
- Example:

```
class Box {
    public:
        float length;
        float breadth;
        float height;

        double calculateArea(){
            return length * breadth;
        }

        double calculateVolume(){
            return length * breadth * height;
        }

};

int main(){
    // create objects
    Box box1;  // one object
    box1.length=3.55;
    box1.breadth =4.55;
    box1.height =5.6;
    cout<<"Area of the box is "<<box1.calculateArea()<<endl;
    cout<<"Volume of the box is "<<box1.calculateVolume();

}
```

- Notice that the public elements are accessible from main(). This is because public elements are accessible from all parts of the program.

Access specifier or Access modifier:

- **private Access Modifier:**
  - The private keyword is used to create private members (data and functions).
  - The private members can only be accessed from within the class.
  - However, friend classes and friend functions can access private members.

```
class Box {
    private:
        float length;
        float breadth;
        float height;
    public:
        double calculateArea(){
            return length * breadth;
        }

        double calculateVolume(){
            return length * breadth * height;
        }

};

int main(){
    // create objects
    Box box1;  // one object
    box1.length=3.55;
    box1.breadth =4.55;
    box1.height =5.6;
    cout<<"Area of the box is "<<box1.calculateArea()<<endl;
    cout<<"Volume of the box is "<<box1.calculateVolume();

}
```

Error

7	15	E:\UET\Spring 23\OOP\Class\class_1.cpp	[Error] 'float Box::length' is private
24	10	E:\UET\Spring 23\OOP\Class\class_1.cpp	[Error] within this context
8	15	E:\UET\Spring 23\OOP\Class\class_1.cpp	[Error] 'float Box::breadth' is private
25	10	E:\UET\Spring 23\OOP\Class\class_1.cpp	[Error] within this context
9	15	E:\UET\Spring 23\OOP\Class\class_1.cpp	[Error] 'float Box::height' is private
26	10	E:\UET\Spring 23\OOP\Class\class_1.cpp	[Error] within this context

Credit: Khola Naseem

Access specifier or Access modifier:

- **private Access Modifier:**

```
class Box {
    private:
        float length;
        float breadth;
        float height;
    public:
        double calculateArea(){
            return length * breadth;
        }

        double calculateVolume(){
            return length * breadth * height;
        }

};

int main(){
    // create objects
    Box box1;  // one object
    cout<<"Area of the box is "<<box1.calculateArea()<<endl;
    cout<<"Volume of the box is "<<box1.calculateVolume();

}
```

- Output

```
E:\UET\Spring 23\OOP\Class\Classes.exe
Area of the box is 0
Volume of the box is 0
--------------------------------
Process exited after 0.1794 seconds with return value 0
Press any key to continue . . .
```

Credit: Khola Naseem

Access specifier or Access modifier:

- **Protected Access Modifier:**
  - protected access specifier is used in inheritance in C++.
  - The protected keyword is used to create protected members (data and function).
  - The protected members can be accessed within the class and from the derived class.