C++ Programming

Post Graduate Diploma in Advanced Computing (PG-DAC)

ACTS, C-DAC Bangalore

Topics Covered:

- Pointers in C++
  - Pointer Arithmetic
  - Void Pointer
  - Null Pointer
- Functions in C++
  - Categories of Function
  - Call by Value and Call by Reference
- Inline Functions
- Function Overloading

Pointers in C++

What is Pointer

- Used to store the address of another variable.
- Space allocated to a pointer is equal to the size of integers as it stores the address which is integer value.
- Pointers behave as normal variable but stores the address.

Operators Associated With Pointer

- Reference Operator (&)
  - also referred as “address of” operator
  - used to access the address of any variable
  - whenever any variable is declared a memory space is allocated for the variable which has some address
  - this address value is given by address of operator “&”
- Dereference Operator (*)
  - also referred to as value “at operator”
  - used to access the value stored at a particular memory location
  - “*” operator is the value at operator which gives the value at the memory location stored in the pointer variable

Declaration & Initialization of Pointer

Declaration:

```
int *ptr; //points to any integer variable
//Declaration of pointer follows the usual variable creation rules
```

Initialization:

```
ptr = &i; // where i is an integer variable
```

Combination of Declaration & Initialization:

```
int *ptr = &i;
```

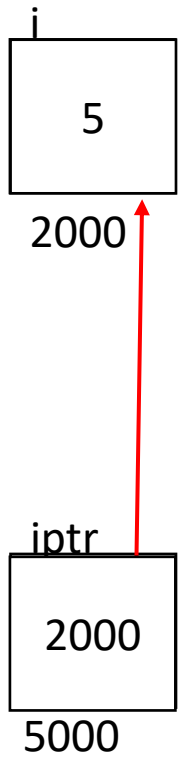
Usage of Pointers

```
#include<stdio.h>

void main()
{
    int i;
    int *iptr;

    i = 5;
    iptr = &i;

    cout<<"\ni "<< i;
    cout<<"\n&i "<< &i;
    cout<<"\niptr "<< iptr;
    cout<<"\n*iptr "<< *iptr;
    cout<<"\n&iptr "<<, &iptr;
```



Operations With Pointers / Pointer Arithmetic

- support only addition and subtraction operation for pointers
- Multiplication, Division, Modulus operation is not supported as the resultant address may or may not be a valid address. The difference in the resultant address and the pointer operands could be huge nd may go out of supported user space.
- Operation on pointer takes the size of data type pointed to in consideration.
- Increment and Decrement operation can also be performed as both addition and subtraction is supported in pointer

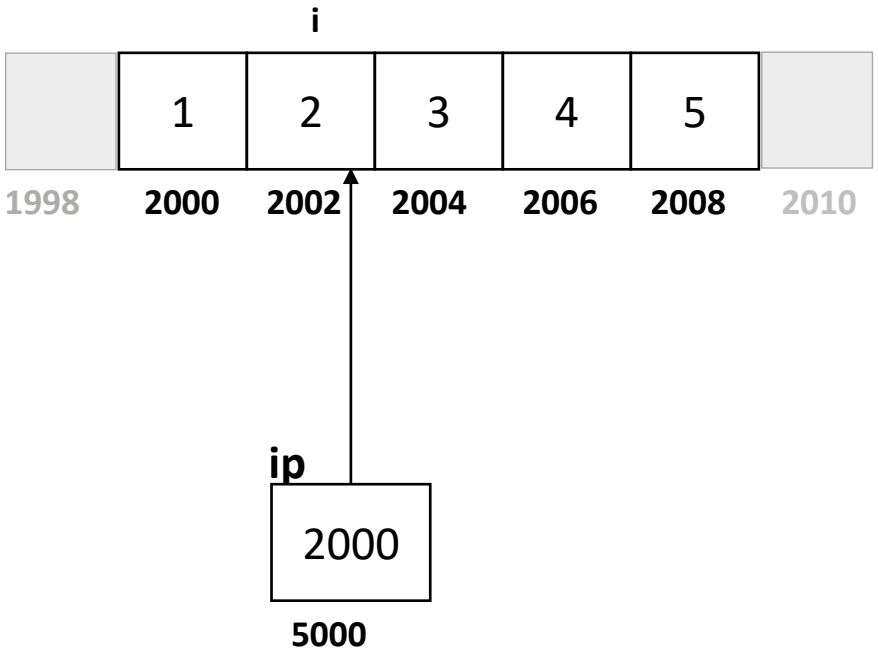
Operations With Pointers / Pointer Arithmetic

```
int i;
int *ip;

i = 2;
ip = &i;

cout<<"\n ip+1 = “<<ip+1;
cout<<"\n *ip+1 "<<*ip+1;
cout<<"\n *ip+1 "<<*(ip+1);

cout<<"\n *ip+1 "<<*ip++;
cout<<"\n ++*ip "<<++*ip;
```



Uses of Pointers

- For call by reference operation in function
- For accessing array elements
- To return multiple values from function
- For dynamic memory allocation
- For implementing certain data structures like linked list.

Void Pointer

- A void pointer is a pointer that has no associated data type with it.
- can hold address of any type
- has to be type-casted to the type of variable its going to point to.

```
#include<stdio.h>
int main()
{
int a = 10;
void *ptr = &a;
cout<<*(int *)ptr;
return 0;
}
```

Void Pointer : Uses & Limitations

- Uses:
  - in case of dynamic memory allocation
  - for implementation of generic functions like a function of sort that can perform sorting either in ascending or descending order

- Limitations:
  - Pointer Arithmetic is not possible with void pointers due to size constraint
  - It can’t be dereferenced (*ptr is not possible with void *ptr)

NULL Pointer

- NULL is a constant value equivalent to ‘0’ or ‘\0’

- NULL Pointer refers to a fixed location that has the value of NULL
- NULL Pointers are used for specific purposes.
- Any pointer created should ideally be pointing to either a valid address or pointing to NULL otherwise it can generate the dangling pointer (pointer that does not refer to valid location)

NULL Pointer : Uses

- To initialize a pointer variable when that pointer variable isn't assigned any valid memory address yet.

```
int * pInt = NULL;
```

- To check for a null pointer before accessing any pointer variable. By doing so, we can perform error handling in pointer related code e.g. dereference pointer variable only if it's not NULL.

```
if(pInt != NULL) //We could use if(pInt) as well
        { /*Some code*/}
else
        { /*Some code*/}
```

- To pass a null pointer to a function argument when we don't want to pass any valid memory address.

```
int fun(int *ptr)
{
        //statements in function fun
        return 10;
}

fun(NULL);
```

Functions in C++

What are Functions

- set of statements that are used to perform a specific task
- independent block of statements
- Eg: function to compute grade of a student, matrix addition
- main() is specific type of function that is used by the compiler to start the execution of the program with

Advantages of Function

- Modularity
  - the code can be divided into fragments based on what operation it is required to perform
  - makes handling large complex programs easier
- Code Reuse
  - the same set of statements required again and again can be created as function
  - each time it is required, the function can be called
  - no need to write the code each time
- Easy to Debug
  - since code is divided into independent fragments, if any error occurs, it becomes easier to detect which part of code is not working

Types of Functions

- Built-in Function
  - functions available in header files
  - system functionality is already defined in header files
  - each time it is required, only function call needs to be done
  - needs to include the corresponding header file with code
- User Defined Functions
  - functions not available in system but created based on user requirement
  - the function definition needs to be added in the same program or in a separate file which in turn should be included with the program

Function Components

- Function Declaration:
  - provides the signature for the function
  - Syntax:

```
return_type function_name( parameter list );
```

- Function Definition:
  - provides the actual process for the function
  - Syntax:

```
return_type function_name( parameter list ) {
 body of the function
}
```

- Function Calling
  - refers to calling the user defined/ built-in functions
  - Syntax:

```
function_name( parameter_value_list ) ;
OR
data_type variable_name = function_name( parameter_value_list );
```

Example Program For Functions

PROGRAM WITHOUT USER DEFINED FUNCTION

```
#include<stdio.h>

void main()
{
    int num1, num2, max;
    cout<<"\nEnter fist number :";
    cin>>num1;
    cout<<"\nEnter second number : ";
    cin>>num2;
    if (num1 > num2)
        max = num1;
    else
         max = num2;
    cout<<"\nMaximum of two numbers
    “<<num1<<“ & “<<num2<<“ is “<<max;
}
```

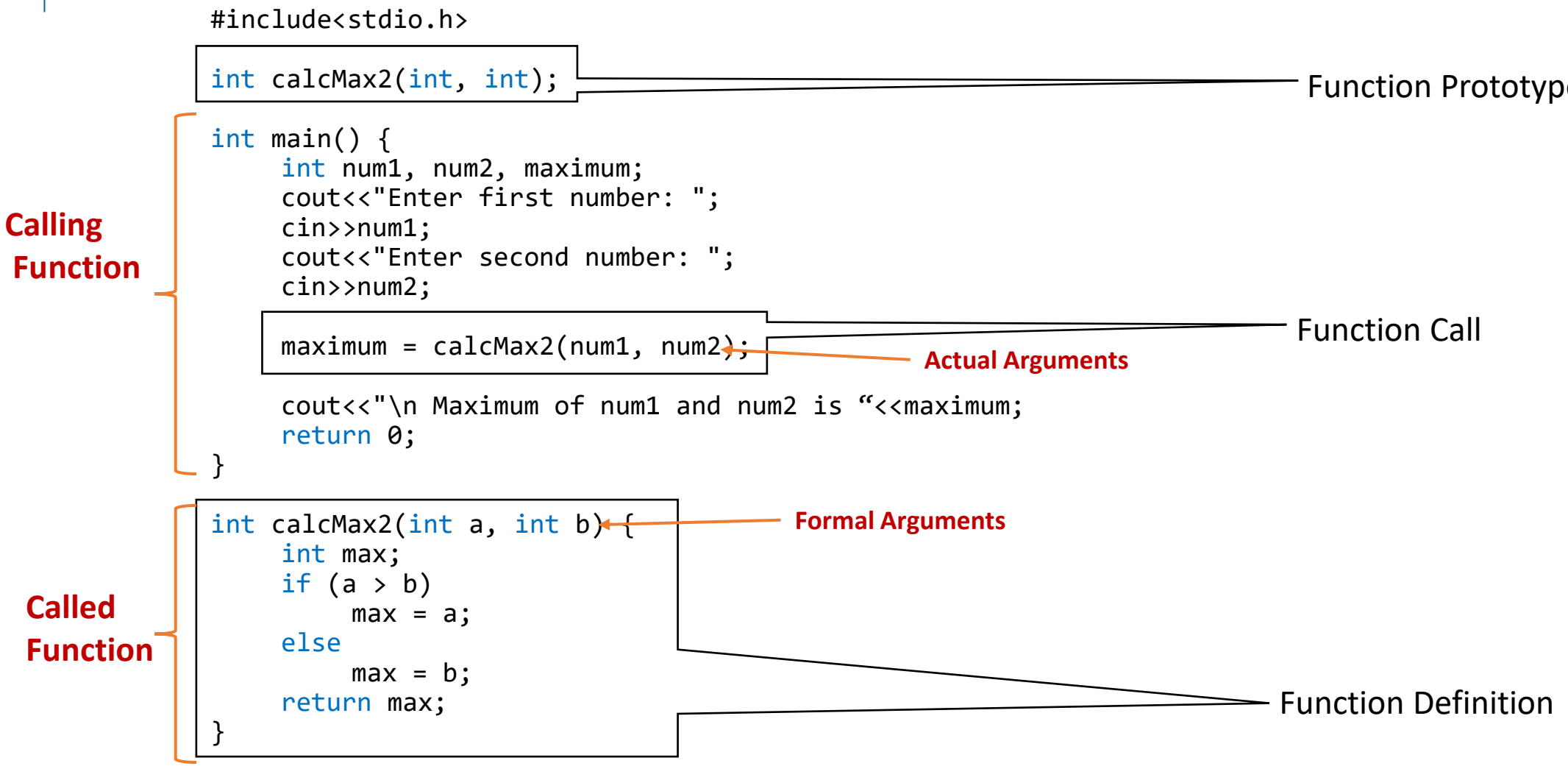
PROGRAM WITH USER DEFINED FUNCTION

```
#include<stdio.h>
int calcMax2(int num1, int num2);//Declaration
void main()
{
    int num1, num2, max;
    cout<<"\nEnter fist number :";
    cin>>num1;
    cout<<"\nEnter second number : ";
    cin>>num2;
    max = calcMax2(num1, num2); //Call
    cout<<"\nMaximum of two numbers “<<num1<<“
    & “<<num2<<“ is “<<max;
}
int calcMax2(int num1, int num2)//Definition
{
    if (num1 > num2)
        return num1;
    else
        return num2;
}
```

Terminologies With Function

- Actual Parameters & Formal Parameters
  - Actual parameters are the actual values that are passed wen a function call is made
  - Formal parameters act like a placeholders or containers defined in function definition for the values being passed during function call
- Calling Function & Called Function
  - Called function is the function that is being called to perform certain task
  - Calling Function is the function where the function call is made

Illustration of Terminologies Associated With Functions



Categories of Functions

Functions can be created in 4 different formats based on return type of the function & number of arguments:

- Function with no parameter and no return value
- Function with no parameter and return value
- Function with parameter(s) and no return value
- Function with parameter(s) and return value

Illustration of Return Types & Parameter Passing

```
#include<stdio.h>

int max(int, int);

int main() {
     int num1, num2, maximum;
     cout<<"Enter first number: ";
     cin>>num1;
     cout<<"Enter second number: ";
     cin>>num2;

     maximum = max(num1, num2);

     cout<<"\n Maximum of "<<num1<<" and "<<num2 is "<<maximum;
     return 0;
}

int max(int a, int b) {
     int max2;
     if (a > b)
          max2 = a;
     else
          max2 = b;
     return max;
}
```

Return Statement:
maximum = max

Parameter Passing:
int a = num1;
int b = num2;

```
int max(int a, int b) {
     int max2;
     if (a > b)
          max2 = a;
     else
          max2 = b;
     return max2;
}
```

Return Value & Parameter Values

- Return Types:
  - A function not returning value has a return type of void.
  - A function returning a value will be of any type other than void
  - All function that return a value should have a return statement

- Parameter Values
  - A Function call may pass some values for the function to utilize that are parameters
  - If no values are to be passed then the parameter set is empty.
  - The assignment of values follows the left to right association.

Categories of Functions

With No Parameter and No Return Value

```
#include<stdio.h>
void displayMessage(){
  cout<<"Function with no parameter no return value";
}
void main(){
  displayMessage();
}
```

With No Parameter but Return Value

```
#include<stdio.h>
int getUserInt(){
  int x;
  cout<<"Enter an integer value";
  cin>>x;
  return x;
}
void main(){
  int x = getUserInt();
  cout<<“\nx=”<<x;
}
```

Categories of Functions

With Parameter(s) but No Return Value

```
#include<stdio.h>
void displayValue(int x){
  cout<<"Value for x is %d", x);
}
void main(){
  displayValue(7);
}
```

With Parameter(s) and Return Value

```
#include<stdio.h>
int squareInt(int x){
  int squar;
  squar = x * x;
  return squar;
}
void main(){
  int sq;
  sq = squareInt(5);
  cout<<“\nSquare of 5 is %d”, sq);
}
```

Function Calls / Parameter Passing

- There are two methods to pass the arguments to the function:
  - Parameter passing by value (Call By Value)
  - Parameter passing by reference (Call By Reference)
- For each method, the type of parameter that is passed varies.

Function Call – Call By Value / Parameter Passing By Value

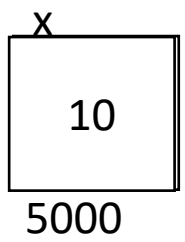
- Call By Value is the method where the function creates a copy of the variables passed as argument and performs all the operations on the copy of variables.
- The actual arguments remain unchanged while the operations are performed by function.
- The changes can be reflected to the original by returning the values and then modifying the original values.

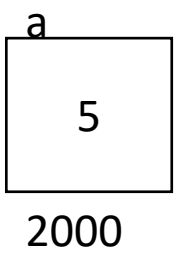
Function Call – Call By Value / Parameter Passing By Value : Example

```
#include <stdio.h>

void add5(int x){
      x = x + 5;
      cout<<“\n x = “<<x;
}

int main(){
      int a;
      cout<<"Enter the values for a : ";
      cin>>a;//Assuming entered value is 5
      add5(a);
      cout<<“\n a = “<<a;
      return 0;
}
```





Function Calls – Call By Reference / Parameter Passing By Reference

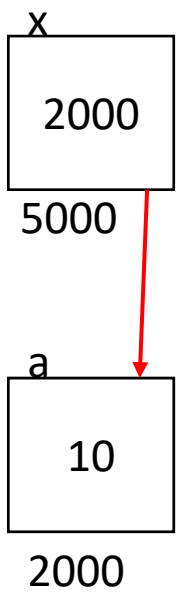
- In call by reference, the address of the variables are passed from the calling function to the called function.
- The called function stores the address using pointer variables.
- So the changes that are made in called function are made at the addresses of the same variables.
- Both the actual argument and the formal argument refer to the same memory location.

Function Call – Call By Reference / Parameter Passing By Reference : Example

```
#include <stdio.h>

void add5(int *x){
      *x = *x + 5;
      cout<<“\n x = ”<<x;
}

int main(){
      int a;
      cout<<"Enter the values for a : ";
      cin>>a;//Assuming entered value is 5
      add5(&a);
      cout<<“\n a = “<<a;
      return 0;
}
```



Function Calls – Call By Value Vs Call By Reference

CALL BY VALUE	CALL BY ADDRESS
When a function is called the values of variables are passed	When a function is called the addresses of variables are passed
The type of formal parameters should be same as type of actual parameters	The type of formal parameters should be pointer of type of actual parameters
Formal parameters contains the values of actual parameters	Formal parameters contain the addresses of actual parameters
Change of actual parameters in the function call will not affect the actual parameters in the calling function.	The actual parameters are changed since the formal parameters indirectly manipulate the actual parameters
Execution is slower since all the values have to be copied into formal parameters	Execution is faster since only addresses are copied.

EXAMPLE PROGRAM

- C Program to swap two numbers using call by value and call by reference