C++ : An Overview

ME5107: Numerical Methods in Thermal Engineering

Jan-May 2025

History of C++

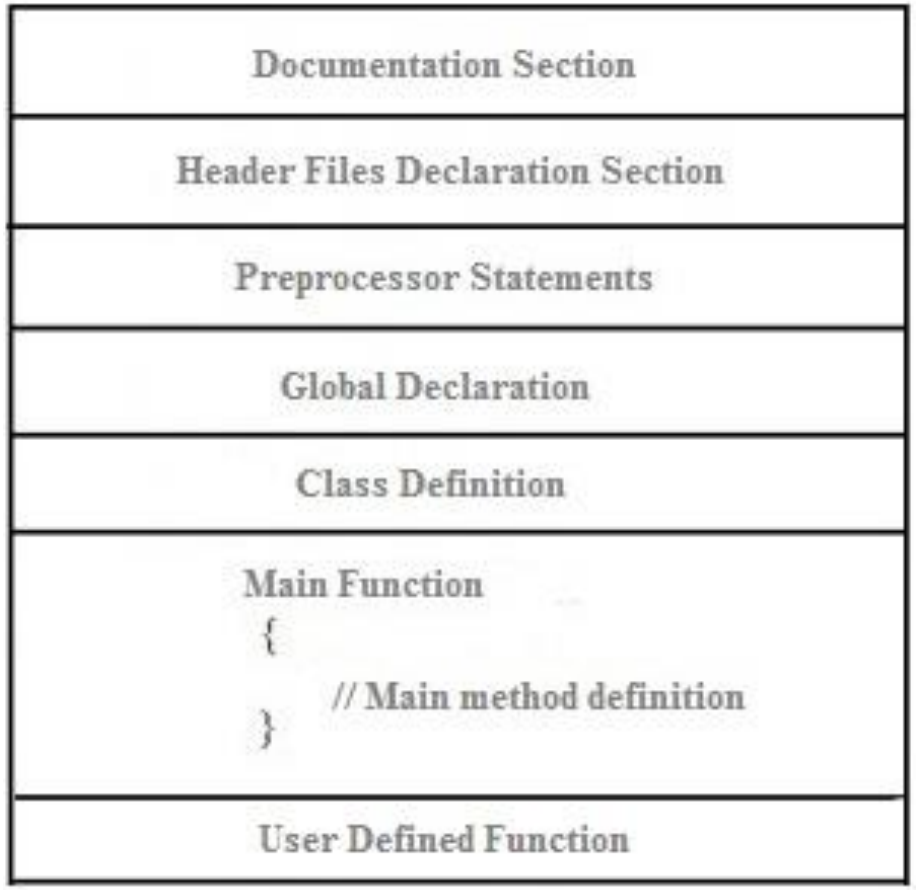
- C++ is a middle level OOP (object oriented programming) language created by Bjarne Stroustrup in 1979 at Bell Labs.
- It is an extension of C and hence, all programmes written in C can run on C++.
- Over the years, it has been a handy tool to build other software packages, especially driver software and software for server applications.

Features of C++

C++ standard library can be categorised into two parts:

1. Standard function library: consisting of general purpose, stand alone functions (inherited from C)
2. Object oriented class library: consisting of classes and associated functions. The Standard Template Library (STL) is a part of this library.

Structure of a C++ Program



How to Run C++ Program

1) Put following command in the terminal to open gedit editor to write your C++ code.

   gedit filename.cpp

2) Write your C++ code in the editor and save the file.

3) Put following command to run the code.

   g++ filename.cpp –o output

4) Step 3 will create output file named “output”. Put following command to open this file.

   ./output

A simple C++ code

```
// A program to print “Hello World”

#include <iostream>      /*Preprocessor directive also includes
                           calling header files and defining
                           constants*/
using namespace std;     //To select the definitions of std library

int main()  //Main function, where the program execution begins
{
      cout << “\n Hello World\n”;   //Output statement
      return 0;
}
```

Variables and Data Types in C++

- Variable is a container for storing some data values. It has a **lvalue** and a **rvalue**.
- Different data types used in C++ are:
  - Primary data type: short, int, float, char, double, long, bool
  - Derived data type: array, function, pointer, reference, string
  - User defined data type: class, structure, union, etc.
- **typedef** command can be used to define a new data type in terms of existing data types.

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
signed short int	2bytes	-32768 to 32767
long int	8bytes	-2,147,483,648 to 2,147,483,647
signed long int	8bytes	same as long int
unsigned long int	8bytes	0 to 4,294,967,295
long long int	8bytes	-(2^63) to (2^63)-1
unsigned long long int	8bytes	0 to 18,446,744,073,709,551,615
float	4bytes	
double	8bytes	
long double	12bytes	

Courtesy: tutorialspoint.com

Variable Scope

- Variables can either have global scope or local scope.

```
// A program to explain variable scope

#include <iostream>

using namespace std;

int a = 15; //global definition of variable 'a'

int main()  // Main function
{
 int a = 10;      //local definition of variable 'a'

 cout << a;
 return 0;
}
```

Self study:

- Variable naming convention
- Keywords and reserved words
- Constants
- Literals & escape sequences
- Type qualifiers
- Storage classes

Basic I/O operations with iostream

- Input operations are done using *cin* (standard input stream) and output/printing operations are done using *cout* (standard output stream).

Syntax:

```
cin >> Var1 >> Var2 >>... >> VarN;
cin.getline("character string", max_string_length);
cin.get(character_input);

cout << "Some printing" << Var1 << "Another printing" << Var2 << endl;
```

<< - insertion operator
endl – causes the cursor to move to a new line (same as '\n')

What is the difference between '\n' and endl ?

Operators in C++

- **Arithmetic operators**
- Relational operators
- Logical operators
- Assignment operators
- Bitwise operators

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Courtesy: w3schools.com

Try Problem 2: Take 2 integers as input and perform all arithmetic operations on them.

Operators in C++

- Arithmetic operators
- **Relational operators**
- Logical operators
- Assignment operators
- Bitwise operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Courtesy: w3schools.com

Operators in C++

- Arithmetic operators
- Relational operators
- **Logical operators**
- Assignment operators
- Bitwise operators

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Logic Gates : AND, OR, NAND, NOR, XOR.

Courtesy: w3schools.com

Operators in C++

- Arithmetic operators
- Relational operators
- Logical operators
- **Assignment operators**
- Bitwise operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Courtesy: w3schools.com

Operators in C++

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- **Bitwise operators**

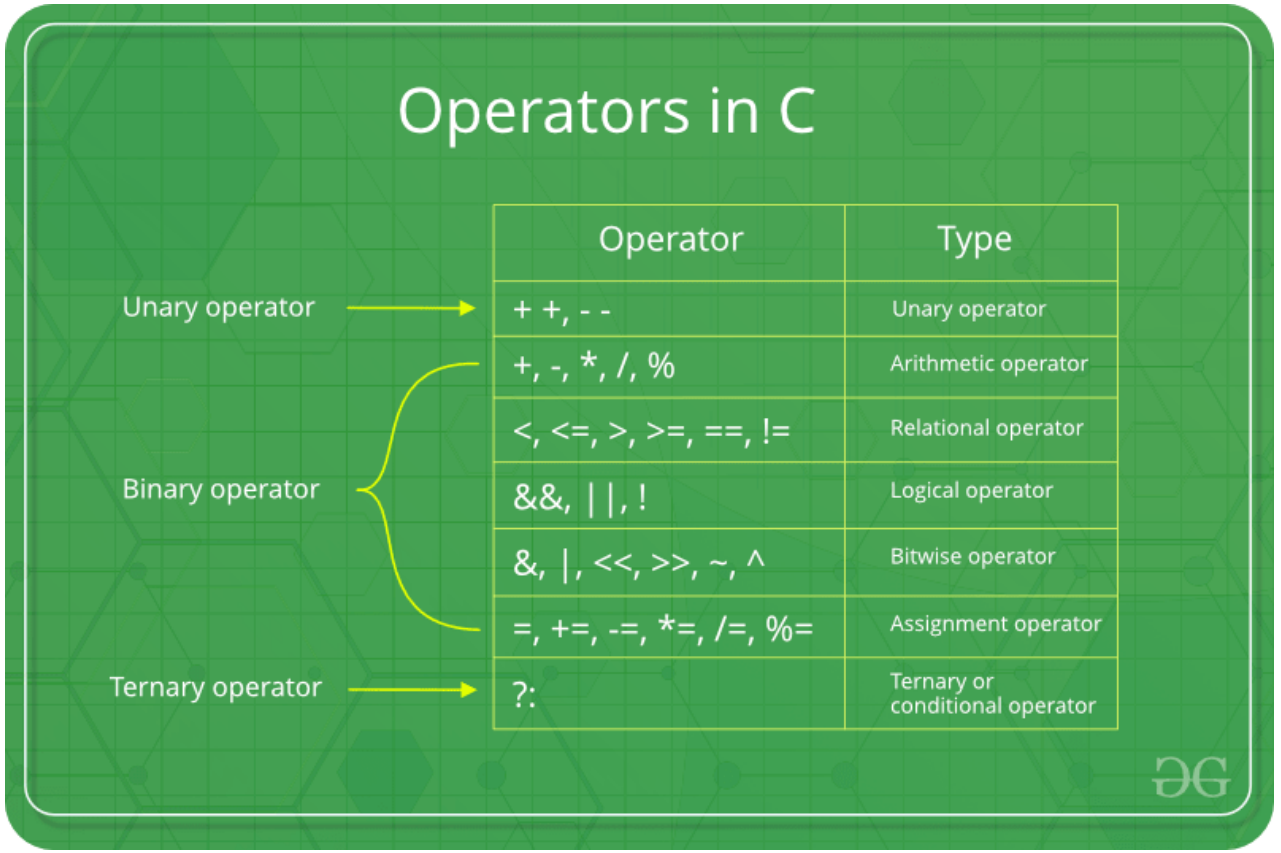
Operator	Operation
&	Bitwise AND
|	Bitwise OR
~	One's Complement
>>	Shift right
<<	Shift left
^	Exclusive OR

Courtesy: cppbetterexplained.com

Operators in C++

- Unary operators
- Binary operators
- Ternary operators

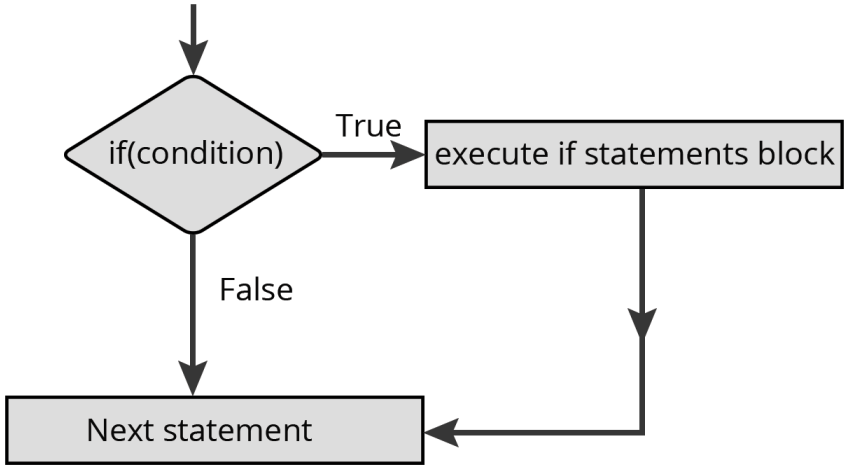
Self study:
- Miscellaneous operators



Courtesy: geeksforgeeks.org

Conditional Statements in C++

- **if-else if-else statement**
- switch-case statement



Syntax:

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

Short hand:

```
int time = 20;
string result = (time < 18) ? "Good day." : "Good evening.";
cout << result;
```

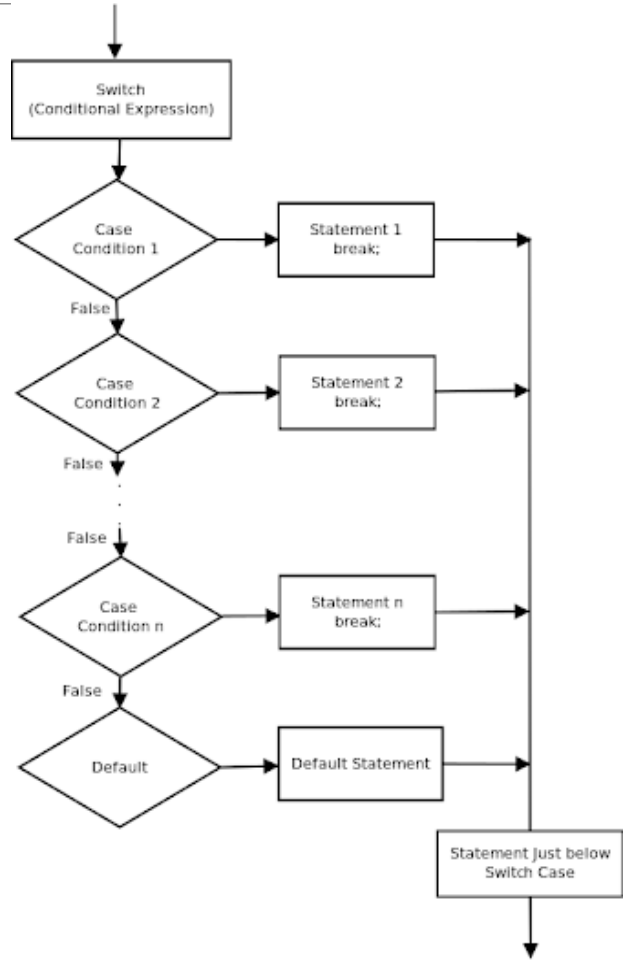
Courtesy: w3schools.com

Conditional Statements in C++

- if-else if-else statement
- **switch-case statement**

Syntax:

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```



Courtesy: w3schools.com; onlinegdb.com

Loops in C++

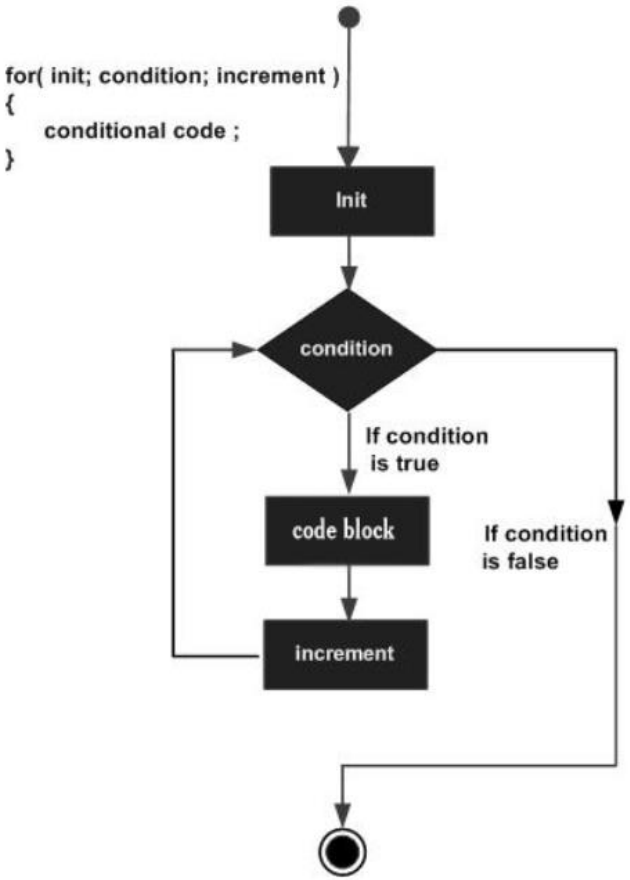
- **for loop**
- while loop
- do-while loop

Example:

```
/*To print all even integers
from 0 to 20 */
#include <iostream>
using namespace std;

int main() {
    for(int i=0; i<=20; i+=2)
    {
        cout << i << '\n';
    }
    return 0;
}
```

Syntax:



Courtesy: tutorialspoint.com

Loops in C++

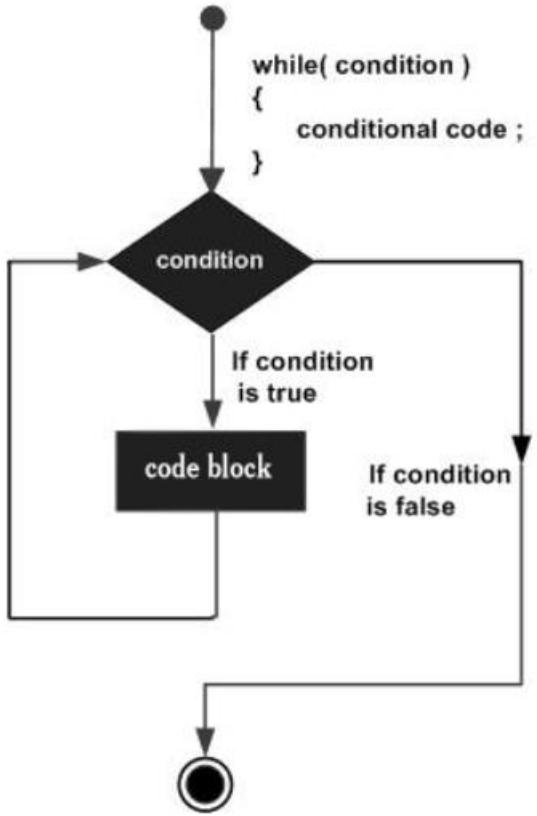
- for loop
- **while loop**
- do-while loop

Example:

```
/*To print all even
integers from 0 to 20 */
#include <iostream>
using namespace std;

int main() {
    int i=0;
    while(i<=20){
        cout << i << '\n';
        i+=2;
    }
    return 0;
}
```

Syntax:



Courtesy: tutorialspoint.com

Loops in C++

- for loop
- while loop
- **do-while loop**

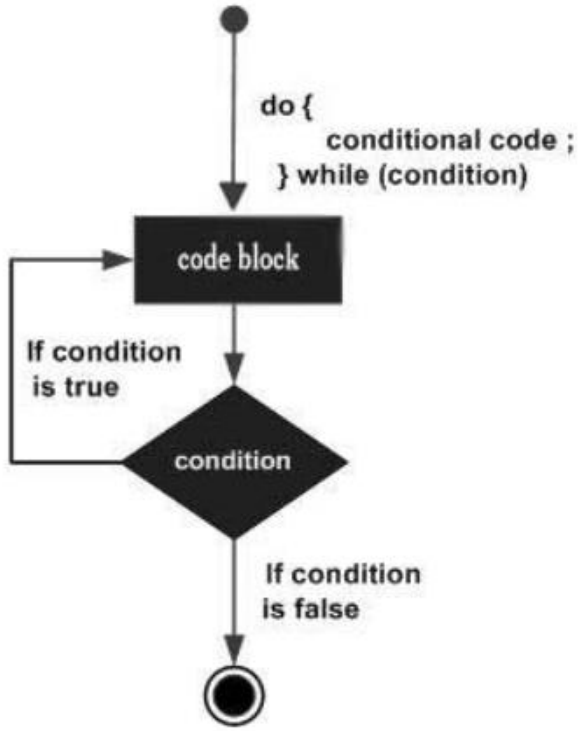
What is the difference between the *do-while* and *while* loops?

Example:

```
/*To print all even
integers from 0 to 20 */
#include <iostream>
using namespace std;

int main() {
    int i=0;
    do {
        cout << i << '\n';
        i+=2;
        } while(i<=20);
    return 0;
}
```

Syntax:



Courtesy: tutorialspoint.com

Nested Loops in C++

```
// To print all prime numbers from 2 to 100 using a nested for and while loop
#include <iostream>
using namespace std;

int main() {
    int i,j,num_f;
    for(i=2;i<=100;i++) {
        j=2; num_f=0;
        while(j<i) {
            if(i%j==0) num_f++;
            j++; }
        if(num_f == 0)   cout << i << " is a prime number\n"; }
    return 0;
}
```

Control Statements in C++

- **break statement**
- continue statement
- goto statement

Example:

```
/*To print all integers from 0 to 15 */
#include <iostream>
using namespace std;

int main() {
    int i=0;
    while(i<=20){
        cout << i << endl;
        if(i==15) break;
        i++;
    }
    return 0;
}
```

Courtesy: tutorialspoint.com

Control Statements in C++

- break statement
- **continue statement**
- goto statement

Example:

```
/*To print all integers from 0 to 20,
skipping 15 */
#include <iostream>
using namespace std;

int main() {
    for(int i=0; i<=20; i++){
        if(i==15) continue;
        cout << i << endl;
    }
    return 0;
}
```

Courtesy: tutorialspoint.com

Try Problem 3: Enter a positive integer and calculate its factorial.

Topics to be covered

- Functions
- Arrays
- Reference and dereference operators, pointers
- Call by value and call by reference
- Dynamic memory – new, delete

Functions in C++

- It is a block of statements which are executed when the function is called.
- Primary advantage: Reusability, i.e., a single block of code written once can be executed many times
- C++ has many predefined functions, such as main(), min(), max(), pow(), clock(), etc.
- User-defined functions have to defined before the calling function is defined. (Or the declaration has to be done before the calling function, in case the definition is done after the calling function.)
- Pre-defined functions can be called from anywhere, given the respective header file has been included in the pre-processor directive.

Functions in C++

```
<return type> function_name(<formal parameters>)  {  //Function declaration,
//Rest is function definition
//Function body
    return <return value>; // For void type, no return statement reqd.
}

int main() {
    result = function_name(<actual parameters>); //Function call
    return 0;
}
```

- The formal arguments can be set as default parameters so that if no parameters are passed to the function, the default value is considered.
- This method of passing values of the arguments is called **Call by value**.

Problem 2: Calculate factorial of a positive integer using a function.

Arrays in C++

- Array is a contiguous set of memory locations.
- It is used to store multiple values in a single variable.
- **Array index starts from 0**.

Declaration:

```
int num[5]; // Declares an array consisting of 20bytes(for 5 int values)
```

Declaration + Initialisation:

```
int num[] = {1,2,3,4,5};   // No need to mention the array size
```

Access:

```
num[0] = 1; num[1] = 2; num[2] = 3; num[3] = 4; num[4] = 5;

for(int i=0;i<5;i++){
    cout << num[i] << "\t"; }
```

Arrays in C++

- 2D and 3D arrays can be defined and accessed in a similar manner.

Declaration:

```
int num[3][2]; // Declares an array of 3 rows and 2 columns
```

Declaration + Initialisation:

```
int num[][2] = {{1,2},{3,4},{5,6}};//Last dimension size needs to be
mentioned
```

Access:

```
num[0][0] = 1; num[0][1] = 2; num[1][0] = 3; num[1][1] = 4; num[2][0] = 5;
num[2][1] = 6; // a[i][j] is a_ij, i.e. i=row index, j=column index
```

```
for(int i=0;i<3;i++){
    for(int j=0;j<2;j++){
        cout << num[i][j] << "\t"; }
    cout << "\n"; }
```

C++ stores data in a **row major** format. So, taking i (row index loop) as outer loop and j (column index loop) as inner loop is faster.

Referencing and Dereferencing

- A variable is a chunk of memory somewhere in the RAM.
- A pointer points to the location of the variable.
- * is the dereferencing operator and & is the referencing operator.

```
int n = 65;  // Variable declaration
int* ptr = &n;    // Pointer declaration

// Reference: Output the memory address of n with the pointer (0x6dfed4)
cout << ptr << "\n";

// Dereference: Output the value of integer with the pointer (65)
cout << *ptr << "\n";
```

Functions in C++ : Call by Value

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

void swap_int(int, int);
int main() {
	int a = 10, b = 20;
	swap(a, b);
	return 0;
}
void swap_int(int x, int y)  {
	x = x+y;
	y = x-y;
	x = x-y;
	cout << "a = " << x << ", b = " << y << endl;
}
```

Functions in C++ : Call by Reference

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

void print_array(int n, int* arr);
int main() {
	int a[] = {50,41,32,74,54};
	int a_size = sizeof(a)/sizeof(a[0]);
	int* ptr;
	ptr = a;
	print_array(a_size, ptr);
	return 0;
}
void print_array(int n, int* arr)  {
	for(int i=0;i<n;i++) {
		cout << arr[i] << "\n";  }
}
```

Passing the pointer to the variable to the function is **Call by reference.**

Functions in C++ : Call by Reference

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

void print_array(int n, int arr[]);
int main() {
      int a[] = {50,41,32,74,54};
      int a_size = sizeof(a)/sizeof(a[0]);
      print_array(a_size, a);
      return 0;
}
void print_array(int n, int arr[])  {
      for(int i=0;i<n;i++) {
            cout << arr[i] << "\n";  }
}
```

Arrays are passed as **call by reference**, by default.

Functions in C++ : Call by Reference

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

void print_array(int m, int n, int** arr);
int main() {
	int** a = new int*[3];
	for (int i=0;i<3;i++)	a[i] = new int[2];
	// Input values to a
	print_array(3,2,a);
	return 0;
}
void print_array(int m, int n, int** arr)  {
	for(int i=0;i<m;i++) {
		for(int j=0;j<n;j++) {
			cout << arr[i][j] << "\t";  }
		cout << "\n"; }
}
```

Problem 3: Pass a 2D array to a function which *returns* its transpose.

Static Memory

- Memory which is allocated during compilation and stays allocated until the program finishes execution.
- The allocated memory can’t be freed, altered or reused for other purposes.
- It leads to low efficiency, but is faster during execution.

```
char c = ‘P’;
int total[5];
float percent = 97.5;
```

Dynamic Memory in C++

- Memory which is allocated during execution of the code.
- Allows reusability and change in size.
- Slower, but more efficient.

```
pointer-variable = new data-type(value);
pointer-variable = new data-type[size];

int* n = new int(25); // Allocates 4 bytes of memory with pointer n pointing
to it
int* arr = new int[5]; // Allocates 20 bytes (5 units) of memory with pointer
arr pointing to the first unit
```

Dynamic Memory in C++

- Memory which is allocated during execution of the code.
- Allows reusability and change in size.
- Slower, but more efficient.

```
int** arr2d = new int*[3]; //Defining a pointer to pointer
for(int i=0;i<3;i++){
    arr2d[i] = new int[3];}
```

```
delete n;
delete arr;
delete[] arr;
delete arr2d[2];
```

arr2d

arr2d[0]
arr2d[1]
arr2d[2]

arr2d[0][0]	arr2d[0][1]	arr2d[0][2]
arr2d[1][0]	arr2d[1][1]	arr2d[1][2]
arr2d[2][0]	arr2d[2][1]	arr2d[2][2]

Static vs Dynamic Memory

Static Memory	Dynamic Memory
Allocation during compilation	Allocation during runtime
Memory size fixed	Changeable memory size
No memory reusability and less efficient	Memory can be freed and reused, hence, more efficient
Execution is faster	Execution is slower
Memory stays allocated till termination of code	Memory can be freed, whenever required

Problem 1: Define a 1D integer array dynamically and check for the largest absolute integer.

Further learning

1) Stroustrup, B., 2000. *The C++ programming language*. Pearson Education India.
2) Kanetkar, Y.P., 2010. *Let us C++*. BPB Publications.
3) https://www.w3schools.com/cpp
4) https://www.tutorialspoint.com/cplusplus
5) https://www.geeksforgeeks.org/c-plus-plus
6) https://www.programiz.com/cpp-programming
7) https://www.cplusplus.com/doc/tutorial