C++ Programming

C++

C++ is a general-purpose programming language

Implementations of C++ exist from some of the most modest microcomputers to the largest supercomputers and for almost all operating systems.

C++ is a superset of the C programming language. In addition to the facilities provided by C, C++ provides flexible and efficient facilities for defining new types.

The C++ language support four programming styles:

- Procedural programming
- Data abstraction
- Object-oriented programming
- Generic programming

C++ supports systems programming.

History:

C++ developed by **Bjarne Stroustrup** in 1979 at AT&T Bell Laboratory.

C++ developed using class concept of Simula(already existing language which supported OOP paradigm) and C's efficiency and flexibility for systems programming.

Templates and exceptions added to C++ later.

1979: Work on ''C with Classes'' started. The initial feature set included classes and derived classes, public/private access control, constructors and destructors, and function declarations with argument checking.

1984: ''C with Classes'' was renamed to C++. By then, C++ had acquired virtual functions, function and operator overloading, references, and the I/O stream, number libraries.

1985: First commercial release of C++

1991: The C++ Programming Language, Second Edition, presenting generic programming using templates and error handling based on exceptions

1997: The C++ Programming Language, Third Edition [Stroustrup,1997] introduced ISO C++, including namespaces, dynamic_cast, and many refinements of templates. The standard library added the STL framework of generic containers and algorithms

2011: ISO C++11 standard was formally approved. In 2012 The first complete C++11 implementations emerged. 2012 Work on future ISO C++ standards (referred to as C++14 and C++17) started

2017: the final standard was published in December 2017.[

Need of C++

C++ is a systems programming language used for developing native applications

Software written in C++ is everywhere: It is in your computer, your phone, your car, and other many electronic devices

Many operating systems have been written in C++ like Windows, Apple's OS, Linux, and most portable-device OSs..

C++ is used to write device drivers and other software that rely on direct manipulation of hardware under real-time constraints.

C++ is used to write some critical parts of most widely used systems like Amazon, Google, Facebook.

Many other advanced technologies depends on C++'s performance and reliability in their implementations – JVM of Java Technologies, Web services framework of Microsoft's .NET technologies, JavaScript interpreters of many browsers like Microsoft's Internet Explorer, Mozilla's Firefox, Apple's Safari, and Google's Chrome.

Games has been another major applications area for C++.

C Programming Language	C++ Programming Language
C was developed by Dennis Ritchie between the year 1969 and 1973 at AT&T Bell Labs.	C++ was developed by Bjarne Stroustrup in 1979 at AT&T Bell Labs
C supports procedural programming.	C++ is hybrid language, because it supports procedural and object oriented programming and generic programming.
C is a function driven language	C++ object driven language
Functions are written and data is passed as parameter in functions.	Data and functions are encapsulated together in an object in C++.
Different operators are supported on built-in data types	Different operators are supported on built-in data types as well as user defined data type
malloc(), calloc(), realloc(), free() functions are used for memory management	Operators new and delete are used for memory management
	Features like – namespace, template, exception handling, reference variable, function and operator overloading are supported in C++
Total 32 keywords in Standard C	Total 63 keywords in C++

First C++ Program :

```
#include<iostream>

using namespace std;

int main()

{
   cout<<"Hello World..";

   return 0;
}
```

Tokens in C++ : In C++, tokens can be defined as the smallest building block of C++ programs that the compiler understands. Every word in a C++ source code can be considered a token.

Before translation parser does parsing of each statement and finds out tokens. Each token has some pre-defined meaning in compiler. Compiler translates each token in machine understandable form and returns .exe file. This .exe file is executable file, which can be executed directly on OS prompt.

Types of tokens :

- Identifiers
- Keywords
- Constants
- Datatypes
- Strings
- Operators
- Escape sequence

Identifier : It is user-defined word. Used to define and declare variable_name, function_name, user_defined data types etc..

Rules to define identifier:

1. It should start with a alphabet or _ (underscore)
2. White space and special characters are allowed in name of identifier
3. It can have alphabet, digits and _ (underscore)
4. It cannot be a keyword
5. It must be unique (can only be declared once in a namespace)
6. C++ is case sensitive, so be careful.

Keywords : C++ is case sensitive. All keywords are in lower-case.

asm	auto	bool	break	case	catch	char
class	const	const_cast	continue	default	delete	do
double	dynamic_cast	else	enum	explicit	export	extern
false	float	for	friend	goto	if	inline
int	long	mutable	namesapce	new	operator	private
protected	public	register	reinterprete_cast	return	short	signed
sizeof	static	static_cast	struct	switch	template	this
throw	true	try	typedef	typeid	typename	union
unsigned	using	virtual	void	volatile	wchar_t	while

#define MAX 100

Preprocessor directive, works like find and replace. Before compilation all occurrences of word MAX in program are replaced by 100.

Constants :

const int i = 10;

The value of i will remain unchanged for entire program.

constexpr double area = 3.14 * r * r;

The expression is evaluated at compile time and value is kept in area name, which is constant.

Demonstration: constDemo.cpp
Discuss Assignment #1

Datatypes:

We use the variables to store values in programs, but the OS has to know what kind of data we want to store in them, since OS is going to allocate some amount of memory to store a value.

A declaration is statement that introduces a variable name into the program. It specifies a datatype for the variable

A datatype defines a set of possible values and a set of operations we can perform on those values.

C++ offers a variety of primitive (built-in) datatypes.

Datatypes supported in C++ : Size and range depends on the system the program is compiled for,

Name	**Description**	**Size***	**Range***
char	Character or small integer.	1byte signed:	-128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value.	1byte	true or false
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	Wide character.	2 or 4 bytes	1 wide character

Discuss Assignment #2

string in C++ :

Variables that can store non-numerical values that are longer than one single character are known as strings.

The C++ language library provides support for strings through the standard string class. This is not a primitive type, but it behaves like a primitive types in its most basic usage.

Demonstration: stringDemo.cpp
Discuss Assignment #3

Operator: C++ has very rich set of operators.

Assignment operator =

The part at the left of the assignment operator (=) is known as the lvalue (left value) and the right one as the rvalue (right value).
The lvalue has to be a variable whereas the rvalue can be either a constant, a variable, the result of an operation or any combination of these.
The most important rule when assigning is the right-to-left rule:
The assignment operation always takes place from right to left, and never the other way:

```
a = b;
```

This statement assigns to variable a (the lvalue) the value contained in variable b (the rvalue).
The value that was stored until this moment in a is not considered at all in this operation, and in fact that value is lost.

A property that C++ has is that the assignment operation can be used as the rvalue (or part of an rvalue) for another assignment operation.
For example: a = 2 + (b = 5);

is equivalent to:

b = 5; a = 2 + b;

that means: first assign 5 to variable b and then assign to a the value 2 plus the result of the previous assignment of b (i.e. 5), leaving a with a final value of 7.

The following expression is also valid in C++:

a = b = c = 5;   It assigns 5 to the all the three variables: a, b and c

Arithmetic operators (+, -, *, /, %) : Need 2 operands.

The five arithmetical operations supported by the C++ language are:

+ addition
- subtraction
* multiplication
/ division
% modulo

Compound assignment (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

Increment and decrement (++, --)

Relational and equality operators (==, !=, >, >=, <, <=)

Logical operators (!, &&, ||)

Conditional operator **or** **Ternary operator** (? :)

Comma operator (,)

Bitwise Operators (&, |, ^, ~, <>)

Explicit type casting operator

sizeof() operator

dot operator (.)

Scope resolution operator (::)

Arrow operator (->)

- **OperatorDemo.cpp**
- **Implicit_Explicit_Casting.cpp**

Discuss Assignment #4

Special Characters and Escape Sequence :

Sequence	Meaning
\a	Alert.
\b	Backspace.
\f	Form feed.
\n	Newline.
\r	Carriage return.
\t	Horizontal tab.
\v	Vertical tab.
\\	Backslash (\ ).
\′	Single quote (').
\"	Double quote (").
\?	Question mark (?).

Demo: escSequence.cpp

Control Structures in C++:

Condition Structure:

    if..else if..else

Loop Structures:

    for loop
    while loop
    do..while loop

Keywords work with loop structure:

    break
    continue

** Assignments based on condition and loop structures **

Discuss Assignment #5 and #6

Functions in C++ :

A function is a group of statements that is executed when it is called from some point of the program.

Functions are used to structure our programs in a more modular way.

Gives feature of reusability because of its write once and call multiple times mechanism.

They are written to perform a specific task and are also called as module or sub-program or procedure or
sub-routine.

A function can optionally define input parameters that enable callers to pass arguments into the function.

A function can optionally return a value as output.

Every function should be called directly or indirectly from main(). main() function is entry point function for C++ program.

type function_name (parameter1, parameter2, ...) { statements }

where:

- type is the data type specifier of the data returned by the function.
- name is the identifier by which it will be possible to call the function.
- parameters (as many as needed): Each parameter consists of a data type specifier followed by an identifier, like any regular variable declaration (for example: int x) and which acts within the function as a regular local variable. They allow to pass arguments to the function when it is called. The different parameters are separated by commas.

- statements is the function's body. It is a block of statements surrounded by braces { }.

Different forms of functions:

- Function returning nothing and taking no arguments.
- Function return a value and taking no arguments.
- Function returning nothing and taking arguments.
- Function returning a value and taking arguments.

Function declaration, definition and call should match to avoid syntax error.

Function parameters with default value: Demonstration (DefaultParameter.cpp)

Global variable:

- Declared outside any structure or class or function.
- Accessible from through out the program.
- Memory allocation is on data segment.

Local variable:

- Declared inside the function.
- Accessibility is within the function only.
- Memory allocation is on stack segment.

Static variable:

- Declared inside class or function.
- Scope depends on how and where it is declared
- Memory allocation is on data segment.
- Initialization is done only one time.

Demonstrate: GlobalStaticLocal.cpp

Pointers in C++:

It is a variable, which holds address of another variable. The address should be from same process's address space.

Syntax: int *ptr;

Operators used with pointer: addressOf(&) and valueAt operators(*)

Usages:
1. To return more than one values from function. (call by address)
2. To store the address returned by malloc (memory allocation function) at run time.
3. Array is a constant pointer.
4. To access a variable which has been declared outside the function (declared in caller() etc)
5. To write fast and efficient programs.

Pointer comes with a lot of responsibility.

Types of pointers in C++ :

NULL pointer :

1. To initialize a pointer variable when that pointer variable hasn’t been assigned any valid memory address yet.

2. 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 a pointer variable only if it’s not NULL.

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

4. A NULL pointer is used in data structures like trees, linked lists, etc. to indicate the end.

```
int *ptr = NULL;
```

void pointer:

1. generic pointer.
2. does not belong to any datatype.
3. Arithmetic not possible.
4. dereferencing not possible.
5. It should be typecasted before use.
6. Lower level functions used for memory allocation returns void pointer.

dangling pointer: pointer holding an address which is released and not a part of process's address space. Following 3 scenarios where a pointer can become dangling –

1. De-allocation of memory
2. Function returning a pointer
3. Variable goes out of scope

Wild Pointer :

1. A pointer, which is not initialized to valid address (not even NULL) is known as wild pointer.
2. The pointer may have a non-NULL garbage value that may not be a valid address.
3. It is not good to have any wild pointer in our programs.
4. It will cause our program execution crash.

Constant Pointer: you cannot change the address stored in pointer variable.

```
int i=10,j;
int *const ptr=&i;

ptr=&j; //compile time error
*ptr=*ptr+10;
```

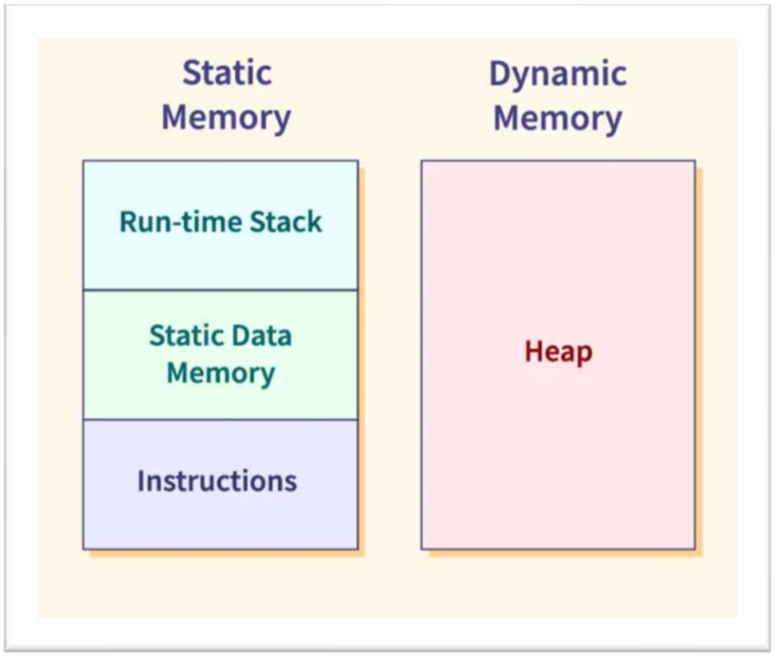
Pointer to constant: you cannot change value using value at operator.

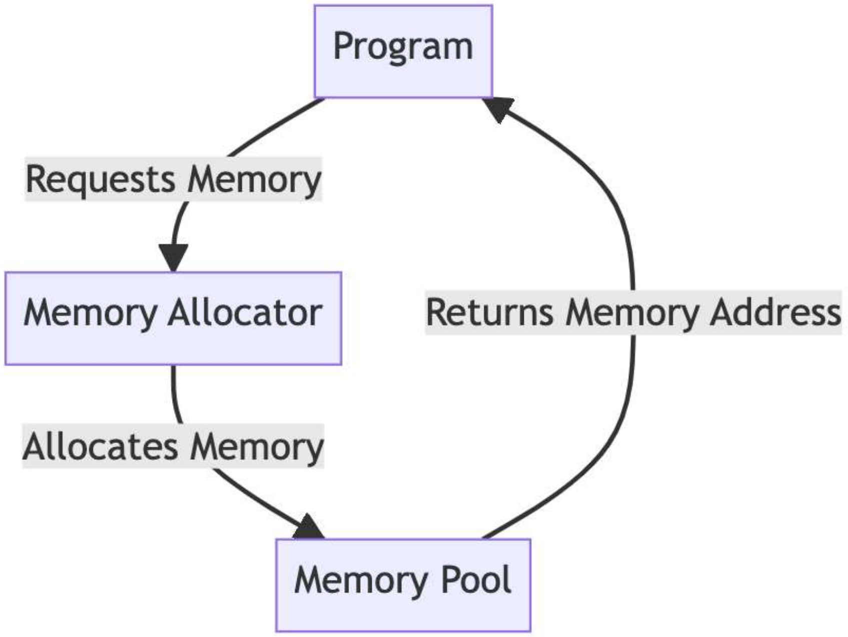
```
const int *p=&i;

*p=*p+10 //error
p=&j;
```

Constant Pointer to constant: you have effect of both, neither can change address nor

```
const int* const pc = &i;
```

Dynamic Memory Allocation





Dynamic memory allocation in C and C++

calloc ( )	malloc ( )	realloc ( )
It is used to dynamically allocate specified number of blocks of memory of the specified type	It is used to dynamically allocate a single large block of memory with the specified size.	It is used to reallocate memory, if previously allocated memory is insufficient
It returns a pointer of type void which can be cast into a pointer of any form.	It returns a pointer of type void which can be cast into a pointer of any form.	It returns a pointer of type void which can be cast into a pointer of any form.
It initialize memory at execution time to 0. It has two parameters or arguments as compare to malloc().	It doesn't Initialize memory at execution time so that it has initialized each block with the default garbage value initially.	Re-allocation of memory maintains the already present value and new blocks will be initialized with the default garbage value.
If space is insufficient, allocation fails and returns a NULL pointer.	If space is insufficient, allocation fails and returns a NULL pointer.	If space is insufficient, allocation fails and returns a NULL pointer.
ptr = (cast-type*)calloc(n, element-size);	*ptr = (int*) malloc(100 * sizeof(int));*	**ptr = (int *) realloc(ptr , 400*sizeof(int))**

Releasing memory after use:

1. Dynamic memory allocation is done by OS on heap on demand at run-time.
2. It is programmers responsibility to release memory after use.
3. Assign NULL to pointer variable after releasing memory.
4. Function free is used to release memory

```
int *ptr = (int *)malloc(4 * sizeof(int));

free(ptr);
ptr = NULL;
```

Demonstration: mallocCallocRealloc.cpp

new and delete operators in C++

- new operator is used to allocate memory at run-time.
- It allocates memory, typecast it and returns address of allocated memory.
- You can allocate and initialize the content on allocated memory in one statement (since C++11)

```
 int *pi = new int (6); // *pi set to 6
 double * pd = new double (99.99); // *pd set to 99.99

int * ar = new int [4] {2,4,6,7};
```

- delete operator is used to release allocated memory.
- pointer variable should be assigned to NULL once memory is released.

```
delete pi;                    delete [ ]ar;
```

Array:

1. collection of values of similar data type
2. contiguous memory allocation
3. fixed size
4. Array name itself is base address of array
5. Array is constant pointer
6. We can access array elements using index or subscript
7. Array index starts from 0 to size-1
8. It can be 1-d, 2-d, n-d
9. Random access is possible

Syntax : datatype nameOfArray[size of array];

Discuss assignment #7

Understand 2-d array and Complex pointer or pointer to array:

Array of arrays is 2-d array.

```
Syntax:    int arr[3][3];
      OR  int (*ptr)[3];
      OR  int ptr[ ][3];
```

Demonstrate 2-d array – addition and multiplication of 2 matrices

Understand 3-d array :

Array of arrays of arrays is 3-d array.

```
Syntax:    int arr[2][2][2];
       OR  int (*ptr)[2][2];
       OR  int ptr[ ][2][2];
```

Demonstrate 3-d array – print all values of 3-d array

Points to remember:

1. Array are never pass by value, they are always pass by address
2. C++ never checks for array index out of bound. It uses garbage value if program tries to go beyond index. The program may crash.

Discuss Assignment #8 and #9

Function returning pointer

The return type of function is pointer.

```
int *fun( )
{
     int var = 15;
     return &var;
}
int main( )
{
     int *ptr;
     ptr = fun( );    //dangling pointer
     cout << *ptr;
     return 0;
 }
```

//error: address of local var is returned

Solution is

```
int *fun(int *p )
 {
      *p = 15;
      return p;
 }
 int main( )
 {
      int *ptr, num = 10;
      ptr = fun(&num);
      cout << *ptr;
      return 0;
  }
```

Function Pointer:

- The pointer which holds address of a function is called as function pointer.
- The advantage of this type of pointer is, it can hold addresses of different functions at different point of tir and we can call different functions with same function call.
- Polymorphism in C++ is possible because of this feature.
- We cannot allocate or de-allocate memory for this type of pointer.

Syntax - int (*fun_ptr)(int, int);

fun_ptr can hold address of function, which returns int and takes 2 int as parameters

Demonstrate function pointer

Discuss Assignment #10

Understand following :

```
int (*ptr)[5];       /*Can point to an array of 5 integers*/

int board[8][8]; // an array of arrays of int
int ** ptr; // a pointer to a pointer to int
int  *risks[10]; // a 10-element array of pointers to integer
int (* rusks)[10]; // a pointer to an array of 10 integers
int  *oof[3][4]; // a 3 x 4 array of pointers to integer
int (*uuf)[3][4]; // a pointer to a 3 x 4 array of integers
int (*uof[3])[4]; // a 3-element array of pointers to 4-element arrays of integer
int *f(void); // Function that returns an int pointer
int (*fp)(int); // a pointer to a function takes one int as argument and returns integer
int (*fp[4])(void);  // An array of 4 pointers to functions, each function returns an integer
float *(*fp)(int, float); // Pointer to a function, function takes two arguments int and float
and returns float pointer
float *(*fp[4])(int, float); // An array of 4 pointers to functions, each function takes two
arguments of int and float type and returns float pointer
```

Enumeration in C++

- It is a user defined datatype to declare symbolic names to represent integer constants.
- Enumerated types is to enhance the readability of a program.
- By default, the constants in the enumeration list are assigned the integer values 0, 1, 2, and so on.
- If you assign a value to one constant but not to the following constants, the following constants will be numbered sequentially.
  For example, suppose you have this declaration:
  enum color {red, green = 10, blue, yellow};
  Then red is 0, by default, and green, blue and yellow are 10, 11, and 12, respectively.
- It can be used in switch..case structure also.

Demonstrate enumDemo.cpp, enumDemo2.cpp

Reference variable:

- It is an alias declared for already declared variable in C++.
- No separate memory is allocated for reference variable.
- It must initialized at the time of declaration.
- Once a reference refers to a variable it is stuck. It cannot be reassigned.
- It is used for passing parameters into function.

Difference between pointer and reference variable

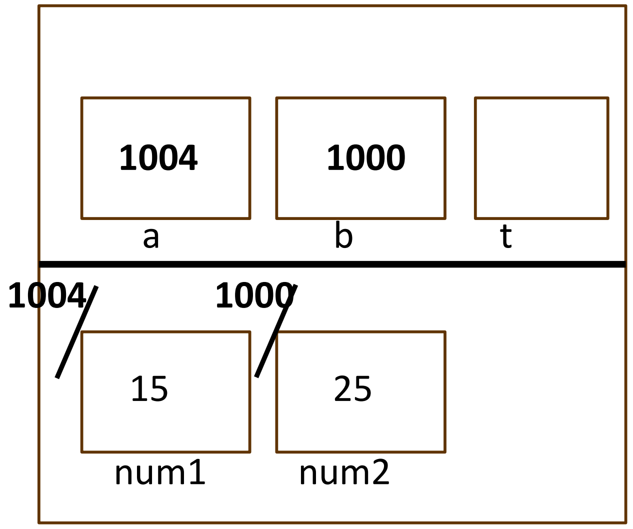
Comparison	Pointer	Reference

Initialization	Its not necessary to initialize the pointer at the time of declaration. Like
	
	int a = 10;
	int *P = &a; //It is not necessary
	Another way is :
	int a = 10;
	int *P;
	P = &a;
		Its necessary to initialize the Reference at the time of declaration. Like
		
		int num=10
		int &a = num;
		
		int &a; //Error here but not in case of Pointer.

Array	You can create the array of Pointer. int *p[5];	You can not create the Array of reference.
NULL value	You can assign NULL to the pointer like int *P = NULL; //Valid	You can not assign NULL to the reference like int &a = NULL; //Error
Pointer to pointer	You can use pointer to pointer.	You can not use reference to reference.

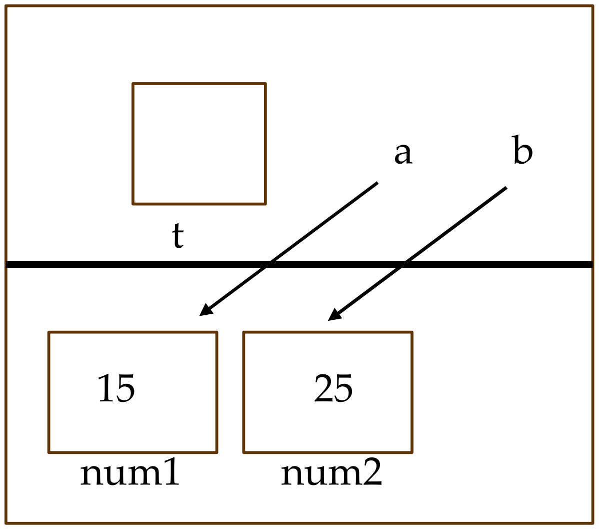
Sample code to understand **call by address**

```
void swap(int *, int *);   // function prototype
int main()
{
   int num1 = 15, num2 = 25;
   swap(&num1,&num2);  //function call
   cout << "\n After swap by address num1 = “ << num1 << " and num2 = “ << num2;
   return 0;
}

void swap(int *a, int *b)  // int *a = &num1   int *b = &num2
{
  int t;
  t = *a;
  *a = *b;
  *b = t;
}
```



Sample code to understand **call by reference**

```
void swap(int & ,int &);
int main()
{
  int num1 = 15, num2 = 25;
  swap(num1, num2);
  cout<<"\n After swap by reference num1 = "<<num1<<" and num2 = "<<num2;
  return 0;
}

void swap(int &a, int &b)  // int &a = num1    int &b = num2
{
     int t;
     t = a;
     a = b;
     b = t;
}
```



typedef:

Used to define our own types based on other existing data types.

Syntax :

typedef existing_type new_type_name ;

where existing_type is a C++ fundamental or compound type and new_type_name is the name for the new type we are defining.

- typedef does not create different types. It only creates synonyms of existing types.
- typedef can be useful to define an alias for a type that is frequently used within a program.
- typedef can be used, if a type we need to use has a name that is too long or confusing.

Examples: typedef unsigned char BYTE;

Function overloading

- We can write more than one function with the same name having different type and number of parameters.
- Used to improve readability of program, by function overloading we can design a family of functions that do essentially the same task but using different argument lists.
- Used for writing variety of constructors in a class.

Examples:

```
void print(const char * str, int width);  // #1
void print(double d, int width);          // #2
void print(long l, int width);            // #3
void print(int i, int width);             // #4
void print(const char *str);              // #5
```

```
double cube(double x);             #1
double cube(double & x);           #2

        cout << cube(x);
```

The x argument matches both the double x prototype and the double &x prototype

The function-matching process does discriminate between const and non-const variables.

Consider the following prototypes:

```
void print(char *s); // overloaded     #1
void print (const char *s); // overloaded    #2
```

Declarations:

```
const char p1[20] = "How's the weather?";
char p2[20] = "How's business?";
```

Function call:

```
print(p2);    // overloaded function #1 is called
print(p1);   // overloaded function #2 is called
```

Function return type is not considered in function prototype

Name Decoration or Name Mangling:

To keep track of overloaded function, C++ compiler does name mangling.
The different functions with different signatures would result in a different name decorated by compiler.
Different compilers would use different conventions for their efforts at decorating.

Demonstration: functionOverloading.cpp

```
D:\cpp\>objdump -d overloading.exe > overloading.asm
```

Discuss Assignment #11