



# Index:

---

1.Overflow in 2's complement addition

---

2.computer memory

---

3.pointer

---

4.DMA

---

5.structure

---

6.file

---

7.-----

---

8.-----

---

9.-----

---

10.-----

---

11.-----

---

12.-----

13.-----

---

14.-----

---

15.-----

---

16.-----

---

17.-----

---

18.-----

---

19.-----

---

20.-----

---

21.-----

---

22.-----

---

23.-----

---

24.-----

---

25.-----

## **1. Overflow in 2's complement addition**

---

$$-8 \leq x[4] \leq +7$$

$$-128 \leq x[8] \leq +127$$

$$-32768 \leq x[16] \leq +32767$$

$$-2147483648 \leq x[32] \leq +2147483647$$

What if the result overflows the representation?

If the result of an arithmetic operation is too large (positive or negative) to fit into the resultant bit-group, then arithmetic overflow occurs. It is normally left to the programmer to decide how to deal with this situation.

### **Overflow Rule for addition:**

If 2 Two's Complement numbers are added, and they both have the same sign (both positive or both negative), then overflow occurs if and only if the result has the opposite sign. Overflow never occurs when adding operands with different signs.

i.e. Adding two positive numbers must give a positive result

Adding two negative numbers must give a negative result

Overflow occurs if

- $(+A) + (+B) = -C$
- $(-A) + (-B) = +C$

Example: Using 4-bit Two's Complement numbers ( $-8 \leq x \leq +7$ )

$$\begin{array}{r} (-7) \quad 1001 \\ +(-6) \quad 1010 \\ \hline \end{array}$$

$(-13) \quad 10011 = 3$  : Overflow (largest -ve number is -8)

### **Overflow Rule for Subtraction:**

If 2 Two's Complement numbers are subtracted, and their signs are different, then overflow occurs if and only if the result has the same sign as the subtrahend.

Overflow occurs if

$$(+A) - (-B) = -C$$

$$(-A) - (+B) = +C$$

Example: Using 4-bit Two's Complement numbers ( $-8 \leq x \leq +7$ )

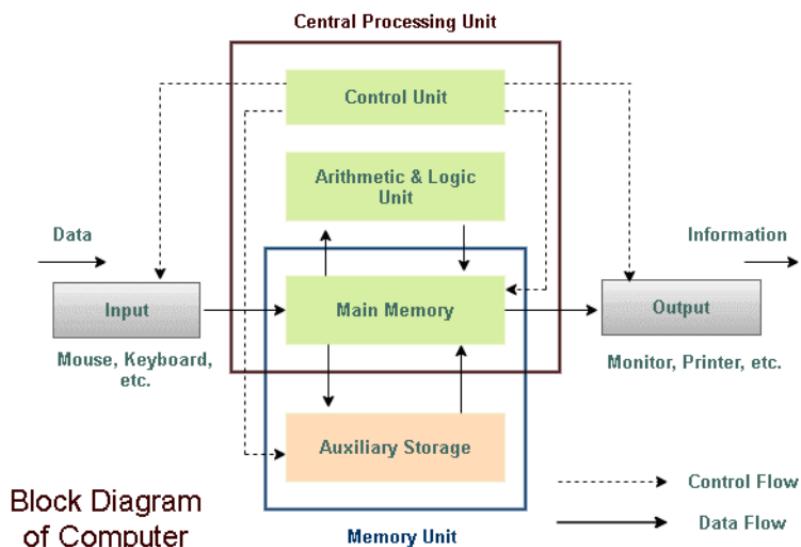
Subtract -6 from +7

$$\begin{array}{r} (+7) \ 0111 \\ -( -6) \ 1010 \rightarrow \text{Negate} \rightarrow +0110 \\ \hline 13 \end{array} \qquad \begin{array}{r} 0111 \\ \hline \end{array}$$

$1101 = -8 + 5 = -3$  : Overflow

## 2. computer memory

A memory is just like a human brain. It is used to store data and instructions. Computer memory is the storage space in the computer, where data is to be processed and instructions required for processing are stored. The memory is divided into large number of small parts called cells. Each location or cell has a unique address, which varies from zero to memory size minus one. For example, if the computer has 64k words, then this memory unit has  $64 * 1024 = 65536$  memory locations. The address of these locations varies from 0 to 65535.



Memory is primarily of three types –

- Cache Memory,
  - Primary Memory/Main Memory,
  - Secondary Memory.
- 
- If there are n number of bit then we can keep  $2^n$  individual number of 0 or 1 in a memory location.
  - It means if we have 2,3 and 4 number of bits then we can keep 4,8 and 16 individual number of 0 or 1.
- 
- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    int i;
    for(i=33;i<=126;i++){
        printf("ASCII code for %c is %d\n",i,i);
    }
    return 0;
}
```

Here,

Remember one thing that if we want to print characters which ASCII value is above 126 can't be possible because from 127 to 255 ASCII value of characters are non-printable.

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    char small_letter,capital_letter;
    printf("Please enter a small letter : ");
    small_letter = getchar();
    capital_letter = small_letter - 32;
    printf("The capital letter is : %c\n",capital_letter);
    return 0;
}
```

Here,

The output will be :

Please enter a small letter : a

The capital letter is : A

Now think about why it happened.

- **Write a function that will take a character type variable as a parameter and if the character is a digit then it will return 1 or if it is not then it will return 0.**

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

int sum(char a)
{
    if (a >= 48 && a <= 57)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int main()
{
    char c;
    printf("Please enter a character : ");
    c = getchar();
    printf("Return value : %d\n", sum(c));
    return 0;
}
```

- **Address of variable :**

Byte is the smallest unit of computer. One after one many bytes are arranged in memory. Every single bytes have a specific address. when we declare a character variable it takes a byte in memory. Again if we declare a integer type variable it will take four byte one after one in memory. But which memory address they will take we can not say it. On the other hand if we declare any array then all the elements of the array will take place in memory one after one.

- **What will be the output of the program given bellow:**

```
#include <iostream>
using namespace std;
int main()
{
    char ch1 = 'A', ch2 = 'B';
    int n1 = 100, n2 = 100000;

    printf("Value of ch1=%c,\t", ch1);
    printf("Address of ch1=%p\n", &ch1);

    printf("Value of ch2=%c,\t", ch2);
    printf("Address of ch2=%p\n", &ch2);

    printf("Value of n1=%c,\t", n1);
    printf("Address of n1=%p\n", &n1);
```

```
printf("Value of n2=%c,\t", n2);
printf("Address of n2=%p\n", &n2);
return 0;
}
```

## **Output:**

Value of ch1=A, Address of ch1=0061FF1F

Value of ch2=B, Address of ch2=0061FF1E

Value of n1=d, Address of n1=0061FF18

Value of n2=á, Address of n2=0061FF14

Here,

%p is used for print the variable address in hexadecimal.

If n is a variable then the address of it is &n.

That's why when we use scanf function we use & before the variables name.

- What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    int ara[5] = {50, 60, 70, 80, 90};

    printf("Value of Array: %d, %d, %d, %d, %d,\n",
ara[0], ara[1], ara[2], ara[3], ara[4]);
    printf("Address of ara is %d\n", ara);
    printf("Address of ara[0] is %d\n", &ara[0]);
```

```
printf("Address of ara[1] is %d\n", &ara[1]);
printf("Address of ara[2] is %d\n", &ara[2]);
printf("Address of ara[3] is %d\n", &ara[3]);
printf("Address of ara[4] is %d\n", &ara[4]);

return 0;
}
```

### **Output:**

Value of Array: 50, 60, 70, 80, 90,  
Address of ara is 6422284  
Address of ara[0] is 6422284  
Address of ara[1] is 6422288  
Address of ara[2] is 6422292  
Address of ara[3] is 6422296  
Address of ara[4] is 6422300

Here we can see that at first the program will print all the arrays elements in the first line. In the second line it will print the starting address of the array, it means the address of memory storage from where the array started. The third line will print as well as the second line do because the starting address of array and the first elements address of array are same. In the fourth line it will print the second elements address of array which is large 4 byte than the first element because the array is an integer data type and we know that every integer data type variable takes 4 byte of memory storage.

- The size of variables of varies data type depends on computer architecture.
- There are a operator in C language called sizeof which can find out the variable size in byte.
- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    int x = 10;

    printf("Value of x is %d\n", x);
    printf("Address of x is %p\n", &x);
    printf("Address of x is %X\n", &x);
    printf("Address of x is %x\n", &x);
    printf("Address of x is %d\n", &x);

    return 0;
}
```

### **Output:**

Value of x is 10  
Address of x is 0061FF1C  
Address of x is 61FF1C  
Address of x is 61ff1c  
Address of x is 6422300

- Write a program that will print size of int, char, double and float data type variables in byte.

```
#include <iostream>
using namespace std;
int main()
{
    int num;
    char ch;
    double d_num;
    float f_num;

    printf("%lu\n",sizeof(int));
    printf("%lu\n",sizeof(double));
    printf("Size of int in byte: %d\n",sizeof(num));
    printf("Size of char in byte: %d\n",sizeof(ch));
    printf("Size of double in byte: %d\n",sizeof(d_num));
    printf("Size of float in byte: %d\n",sizeof(f_num));

    return 0;
}
```

### **Output:**

```
4
8
Size of int in byte: 4
Size of char in byte: 1
Size of double in byte: 8
Size of float in byte: 4
```

- If we want we can declare variable to keep in register in C language. As a example:

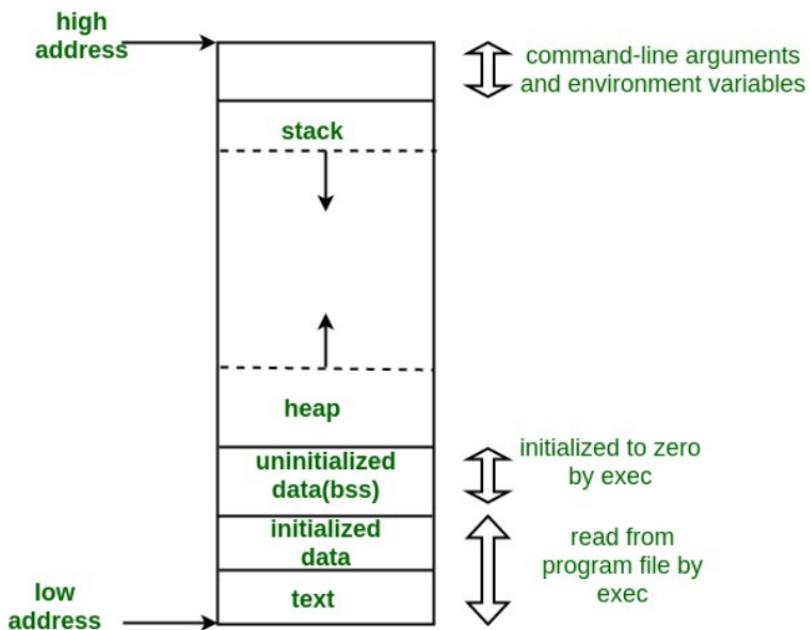
```
register int number;
```

- Register is the fastest storage of all. It is situated in the computer processor and after this cash memory is attached with it. Cash memory is slower than register. On the other hand RAM is attached with the motherboard. RAM is also slower than RAM. There are also a storage called virtual memory. It is more slower than RAM.
- When the storage of RAM isn't enough for processing then computer operating system gives a part of hard disk to use as memory. Of course it is more slower than RAM.

## Memory Layout of C Programs:

A typical memory representation of a C program consists of the following sections.

1. Text segment (i.e. instructions)
2. Initialized data segment
3. Uninitialized data segment (bss)
4. Heap
5. Stack



A typical memory layout of a running process

**1. Text Segment:** A text segment, also known as a code segment or simply as text, is one of the sections of a program in an object file or in memory, which contains executable instructions.

As a memory region, a text segment may be placed below the heap or stack in order to prevent heaps and stack overflows from overwriting it.

Usually, the text segment is sharable so that only a single copy needs to be in memory for frequently executed programs, such as text editors, the C compiler, the shells, and so on. Also, the text segment is often read-only, to prevent a program from accidentally modifying its instructions.

**2. Initialized Data Segment:** Initialized data segment, usually called simply the Data Segment. A data segment is a portion of the virtual address space of a program, which contains the global variables and static variables that are initialized by the programmer.

Note that, the data segment is not read-only, since the values of the variables can be altered at run time.

This segment can be further classified into the initialized read-only area and the initialized read-write area.

For instance, the global string defined by `char s[] = "hello world"` in C and a C statement like `int debug=1` outside the main (i.e. global) would be stored in the initialized read-write area. And a global C statement like `const char* string = "hello world"` makes the string literal "hello world" to be

stored in the initialized read-only area and the character pointer variable string in the initialized read-write area.

Ex: static int i = 10 will be stored in the data segment and global int i = 10 will also be stored in data segment

**3. Uninitialized Data Segment:** Uninitialized data segment often called the “bss” segment, named after an ancient assembler operator that stood for “block started by symbol.” Data in this segment is initialized by the kernel to arithmetic 0 before the program starts executing  
uninitialized data starts at the end of the data segment and contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.  
For instance, a variable declared static int i; would be contained in the BSS segment.  
For instance, a global variable declared int j; would be contained in the BSS segment.

**4. Stack:** The stack area traditionally adjoined the heap area and grew in the opposite direction; when the stack pointer met the heap pointer, free memory was exhausted. (With modern large address spaces and virtual memory techniques they may be placed almost anywhere, but they still typically grow in opposite directions.)  
The stack area contains the program stack, a LIFO structure, typically located in the higher

parts of memory. On the standard PC x86 computer architecture, it grows toward address zero; on some other architectures, it grows in the opposite direction. A “stack pointer” register tracks the top of the stack; it is adjusted each time a value is “pushed” onto the stack. The set of values pushed for one function call is termed a “stack frame”; A stack frame consists at minimum of a return address.

Stack, where automatic variables are stored, along with information that is saved each time a function is called. Each time a function is called, the address of where to return to and certain information about the caller’s environment, such as some of the machine registers, are saved on the stack. The newly called function then allocates room on the stack for its automatic variables. This is how recursive functions in C can work. Each time a recursive function calls itself, a new stack frame is used, so one set of variables doesn’t interfere with the variables from another instance of the function.

**5. Heap:** Heap is the segment where dynamic memory allocation usually takes place.

The heap area begins at the end of the BSS segment and grows to larger addresses from there. The Heap area is managed by malloc, realloc, and free, which may use the brk and sbrk system calls to adjust its size (note that the use of brk/sbrk and a single “heap area” is not

required to fulfill the contract of malloc/realloc/free; they may also be implemented using mmap to reserve potentially non-contiguous regions of virtual memory into the process' virtual address space). The Heap area is shared by all shared libraries and dynamically loaded modules in a process.

Examples:

The size(1) command reports the sizes (in bytes) of the text, data, and bss segments. ( for more details please refer man page of size(1) )

1. Check the following simple C program

```
#include <stdio.h>

int main(void)
{
    return 0;
}
```

```
[narendra@CentOS]$ gcc memory-layout.c -o memory-layout
```

```
[narendra@CentOS]$ size memory-layout
text      data      bss      dec      hex   filename
960       248        8     1216      4c0   memory-layout
```

2. Let us add one global variable in the program, now check the size of bss (highlighted in red color).

```
#include <stdio.h>

int global; /* Uninitialized variable stored in bss*/

int main(void)
{
    return 0;
}
```

```
[narendra@CentOS]$ gcc memory-layout.c -o memory-
layout
```

```
[narendra@CentOS]$ size memory-layout
text      data      bss      dec      hex   filename
960       248       12     1220     4c4   memory-layout
```

3. Let us add one static variable which is also stored in bss.

```
#include <stdio.h>

int global; /* Uninitialized variable stored in bss*/

int main(void)
{
    static int i; /* Uninitialized static variable
stored in bss */
    return 0;
}
```

```
[narendra@CentOS]$ gcc memory-layout.c -o memory-layout
```

```
[narendra@CentOS]$ size memory-layout
```

text	data	bss	dec	hex	filename
960	248	16	1224	4c8	memory-layout

4. Let us initialize the static variable which will then be stored in the Data Segment (DS)

```
#include <stdio.h>
```

```
int global; /* Uninitialized variable stored in bss*/
```

```
int main(void)
```

```
{
```

```
    static int i = 100; /* Initialized static variable stored in DS*/
```

```
    return 0;
```

```
}
```

```
[narendra@CentOS]$ gcc memory-layout.c -o memory-layout
```

```
[narendra@CentOS]$ size memory-layout
```

text	data	bss	dec	hex	filename
960	252	12	1224	4c8	memory-layout

5. Let us initialize the global variable which will then be stored in the Data Segment (DS)

```
#include <stdio.h>

int global = 10; /* initialized global variable stored in DS*/

int main(void)
{
    static int i = 100; /* Initialized static variable stored in DS*/
    return 0;
}
```

```
[narendra@CentOS]$ gcc memory-layout.c -o memory-layout
```

```
[narendra@CentOS]$ size memory-layout
text      data      bss      dec      hex filename
960       256        8     1224      4c8  memory-layout
```

### **3. pointer**

---

- A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before you can use it to store any variable address.
- The general form of a pointer variable declaration is:

```
dataType *var_name;
```

Here,

- dataType is the pointer's base type; it must be a valid C data type(i.e., int, float, char etc).
- var\_name is the name of the pointer variable.
- The asterisk \* you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer.

- Pointers are powerful features of C and C++ programming that differentiates it from other popular programming languages like : Java and Python.
- Using pointer makes the software more efficient cause it works with memory management.
- But excessive usage may make the application less understandable.
- If you want to be proficient in the writing of code in the C programming language, you must have a thorough working knowledge of how to use pointers.
- Some C programming tasks are performed more easily without pointers, and other tasks, such as dynamic memory allocation, cannot be performed without using pointers. So it becomes necessary to learn pointers to become a perfect C programmer.
- As you know, every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory. Consider the following example:

Following are the valid pointer declaration:

```
int    *ip;     /* pointer to an integer */
double *dp;     /* pointer to a double */
float  *fp;     /* pointer to a float */
char   *ch;     /* pointer to a character */
```

- The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address.
- The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

### **Use of pointers :**

- There are few important operations, which we will do with the help of pointers very frequently.
  - 1.we define a pointer variable
  - 2.assign the address of a variable to a pointer and
  - 3.finally access the value at the address available in the pointer variable by dereferencing.
- Dereferencing is done by using unary operator \* that returns the value of the variable located at the address specified by its operand.

- Dereferencing a pointer means getting the value stored in the memory at the address which the pointer “points” to.
  - The **\*** is the value-at-address operator, also called the **indirection operator**. It is used both when declaring a pointer and when dereferencing a pointer.
- 
- **Following example makes use of these operations:**

```
#include<stdio.h>
int main()
{
    int var = 20; /* actual variable declaration */
    int *ip;    /* pointer variable declaration */
    ip = &var; /* store address of var in pointer
                variable */

    printf("Address of var variable: %x\n",&var);

    /* address stored in pointer variable */
    printf("Address stored in ip variable: %x\n",ip);

    /* access the value using the pointer */
    printf("Value of *ip variable: %d\n",*ip);
    return 0;
}
```

## **Output:**

Address of var variable: 61ff18

Address stored in ip variable: 61ff18

Value of \*ip variable: 20

- the & is the address-of operator and is used to reference the memory address of a variable.
- By using the & operator in front of a variable name we can retrieve the memory address-of that variable. It is best to read this operator as address-of operator.
- Following code shows some common notations for the value-at-address (\*) and address-of (&) operators.

```
// declare an variable ptr which holds the value-at-address of an int type
int *ptr;
// declare assign an int the literal value of 1
int val = 1;
// assign to ptr the address-of the val variable
ptr = &val;
// dereference and get the value-at-address stored in ptr
int deref = *ptr;
printf("%d\n", deref);
```

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    int x = 10;

    printf("Value of x is %d\n", x);
    printf("Address of x is %p\n", &x);
    return 0;
}
```

### **Output:**

Value of x is 10  
Address of x is 0061FF1C

- **What will be the output of the program given below:**

```
#include <iostream>

using namespace std;

int main(void)
{
    int ar[5] = {1, 6, 4, 11, 9};

    printf("%d\n", ar);
    printf("%d\n", ar+0);
    printf("%d\n", ar+1);
    printf("%d\n", ar+2);
    printf("%d\n", ar+3);
    printf("%d\n", ar+4);

}
```

**Output:**

6422284  
6422284  
6422288  
6422292  
6422296  
6422300

Here,

We can understand that array is nothing but a pointer. And the concept of the array comes from pointer.

- **What will be the output of the program given below:**

```
#include <iostream>

using namespace std;

int main(void)
{
    int ar[5] = {1, 6, 4, 2, 9};

    printf("Value of ar[%d] = %d | Memory address =
%d\n",0 , *(ar+0), ar+0);
    printf("Value of ar[%d] = %d | Memory address =
%d\n",1 , *(ar+1), ar+1);
    printf("Value of ar[%d] = %d | Memory address =
%d\n",2 , *(ar+2), ar+2);
    printf("Value of ar[%d] = %d | Memory address =
%d\n",3 , *(ar+3), ar+3);
    printf("Value of ar[%d] = %d | Memory address =
%d\n",4 , *(ar+4), ar+4);

}
```

### **Output:**

Value of ar[0] = 1 | Memory address = 6422284  
Value of ar[1] = 6 | Memory address = 6422288  
Value of ar[2] = 4 | Memory address = 6422292  
Value of ar[3] = 2 | Memory address = 6422296  
Value of ar[4] = 9 | Memory address = 6422300

- **What will be the output of the program given below:**

```
#include <iostream>

using namespace std;

int main(void)
{
    int ar[5] = {1, 6, 4, 2, 9};

    printf("Value of ar[%d] = %d | Memory address =
%d\n",0 , ar[0], ar+0);
    printf("Value of ar[%d] = %d | Memory address =
%d\n",1 , ar[1], ar+1);
    printf("Value of ar[%d] = %d | Memory address =
%d\n",2 , ar[2], ar+2);
    printf("Value of ar[%d] = %d | Memory address =
%d\n",3 , ar[3], ar+3);
    printf("Value of ar[%d] = %d | Memory address =
%d\n",4 , ar[4], ar+4);

}
```

### **Output:**

Value of ar[0] = 1 | Memory address = 6422284  
Value of ar[1] = 6 | Memory address = 6422288  
Value of ar[2] = 4 | Memory address = 6422292  
Value of ar[3] = 2 | Memory address = 6422296  
Value of ar[4] = 9 | Memory address = 6422300

- **What will be the output of the program given below:**

```
#include <iostream>

using namespace std;

int main(void)
{
    int ar[5] = {1, 6, 4, 2, 9};

    printf("Memory address of ar = %d\n",ar);
    printf("Value of ar[%d] = %d | Memory address = %d\n",0 , ar[0], &ar[0]);
    printf("Value of ar[%d] = %d | Memory address = %d\n",1 , ar[1], &ar[1]);
    printf("Value of ar[%d] = %d | Memory address = %d\n",2 , ar[2], &ar[2]);
    printf("Value of ar[%d] = %d | Memory address = %d\n",3 , ar[3], &ar[3]);
    printf("Value of ar[%d] = %d | Memory address = %d\n",4 , ar[4], &ar[4]);

}
```

### **Output:**

Memory address of ar = 6422284  
Value of ar[0] = 1 | Memory address = 6422284  
Value of ar[1] = 6 | Memory address = 6422288  
Value of ar[2] = 4 | Memory address = 6422292  
Value of ar[3] = 2 | Memory address = 6422296  
Value of ar[4] = 9 | Memory address = 6422300

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    int x=10;
    int *p;
    p=&x;
    printf("*p=%d\n",*p);
    printf("Value of p is %d\n",p);
    return 0;
}
```

**Output:**

\*p=10  
Value of p is 6422296

Here,

int \*p is called "integer pointer p" and \*p is called "content of p".

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    int x=10;
    int *p=&x;

    printf("Value of x: %d\n",x);
    *p=20;
    printf("Value of x: %d\n",x);
    printf("Value of p: %d\n",*p);
    printf("Address of x: %d\n",&x);
    printf("Value of p: %d\n",p);
    return 0;
}
```

### **Output:**

Value of x: 10  
Value of x: 20  
Value of p: 20  
Address of x: 6422296  
Value of p: 6422296

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    int x=10;
    int *p;
    printf("Value of x: %d\n",x);
    p=&x;
    *p=20;
    printf("Value of x: %d\n",x);
    x=15;
    printf("Value of x: %d\n",x);
    printf("Value stored at location %p is
%d\n",p,*p);
    printf("Address of x: %p\n",&x);
    printf("Value of p: %p\n",p);
    return 0;
}
```

### **Output:**

Value of x: 10  
Value of x: 20  
Value of x: 15  
Value stored at location 0061FF18 is 15  
Address of x: 0061FF18  
Value of p: 0061FF18

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    int x=10;
    int y;
    int *p;
    printf("Value of x: %d\n",x);
    p=&x;
    y=*p;
    *p=15;
    printf("Value of x: %d\n",x);
    printf("Value of y: %d\n",y);
    printf("Value of *p:%d\n",*p);
    printf("Address of x: %p\n",&x);
    printf("Address of x: %p\n",&y);
    printf("Value of p: %p\n",p);
    return 0;
}
```

### **Output:**

Value of x: 10  
Value of x: 15  
Value of y: 10  
Value of \*p:15  
Address of x: 0061FF18  
Address of x: 0061FF14  
Value of p: 0061FF18

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    int x=10,y;
    int *p,*q;
    p=&x;
    q=&y;
    y=*p;
    *p=15;
    *q=20;
    printf("Value of x: %d\n",x);
    printf("Value of y: %d\n",y);
    printf("Value of *p: %d\n",*p);
    printf("Value of *q: %d\n",*q);
    return 0;
}
```

### **Output:**

Value of x: 15  
Value of y: 20  
Value of \*p: 15  
Value of \*q: 20

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    int x=10,y;
    int *p,*q;
    p=&x;
    y=*p;
    *p=15;
    *q=20;
    printf("Value of x: %d\n",x);
    printf("Value of y: %d\n",y);
    printf("Value of *p: %d\n",*p);
    printf("Value of *q: %d\n",*q);
    return 0;
}
```

Here,

The output will be error because we do not indicate q to point y. So, if we want to do content exes of any pointer we must assign a variable address in this pointer first.

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    int x=10,y;
    int *p,*q;
    p=&x;
    q=&y;
    y=*p;
    *p=15;
    *q=20;
    printf("Value of x: %d\n",x);
    printf("Value of y: %d\n",y);
    printf("Value of *p: %d\n",*p);
    printf("Value of *q: %d\n",*q);
    return 0;
}
```

### **Output:**

Value of x: 15  
Value of y: 20  
Value of \*p: 15  
Value of \*q: 20

- **Make a program which will print the address of the variables defined in hexadecimal.**

```
#include <iostream>
using namespace std;
int main()
{
    int var1;
    char var2[10];
    printf("Address of var1 variable: %x\n",&var1);
    printf("Address of var2 variable: %x\n",&var2);
    printf("Address of var2[0] variable:
%x\n",&var2[0]);
    return 0;
}
```

### **Output:**

Address of var1 variable: 61ff1c  
Address of var2 variable: 61ff12  
Address of var2[0] variable: 61ff12

- **Make a program which will print the address of the variables defined in decimal/integer.**

```
#include <iostream>
using namespace std;
int main()
{
    int var1;
    char var2[10];
    printf("Address of var1 variable: %d\n",&var1);
    printf("Address of var2 variable: %d\n",&var2);
    printf("Address of var2[0] variable:
%d\n",&var2[0]);
    return 0;
}
```

### **Output:**

```
Address of var1 variable: 6422300
Address of var2 variable: 6422290
Address of var2[0] variable: 6422290
```

or,

```
#include <iostream>
using namespace std;
int main()
{
    int a,b=7;
    int *ptr1,*ptr2;

    ptr1=&a;
    ptr2=&b;

    printf("Address of a variable : %d\n",ptr1);
    printf("Address of b variable : %d\n",ptr2);
}
```

- **Make a program which will show a variable address and print the variable using pointer and also show the pointer variable address.**

```
#include <iostream>
using namespace std;
int main()
{
    int x=5;
    int *ptr;
    ptr = &x;

    printf("Value of x = %d\n",x);
    printf("Memory address of x = %d\n",&x);
    printf("Memory address of ptr = x = %d\n",ptr);
    printf("Value of x using pointer = %d\n",*ptr);
    printf("Memory address of ptr = %d\n",&ptr);
    return 0;
}
```

### **Output:**

Value of x = 5

Memory address of x = 6422300

Memory address of ptr = x = 6422300

Value of x using pointer = 5

Memory address of ptr = 6422296

- **Make a program that will show pointer pointing to different variable.**

```
#include <iostream>
using namespace std;
int main()
{
    int x=10,y=20,z=30;
    int *ptr;

    ptr = &x;
    printf("Value of x = %d\n",*ptr);

    ptr = &y;
    printf("Value of x = %d\n",*ptr);

    ptr = &z;
    printf("Value of x = %d\n",*ptr);

    return 0;
}
```

or,

```
#include <iostream>
using namespace std;
int main()
{
    int x=10,y=20,z=30;
    int *ptr1=&x,*ptr2=&y,*ptr3=&z;

    printf("Value of x = %d\nValue of y =
%d\nValue of z = %d",*ptr1,*ptr2,*ptr3);
    return 0;
}
```

- **Make a program that will add two numbers using pointer.**

```
#include <iostream>
using namespace std;
int main()
{
    int x=10,y=20,sum;
    int *ptr1,*ptr2;
    ptr1 = &x;
    ptr2 = &y;

    sum = *ptr1 + *ptr2;
    printf("Sum = %d\n",sum);
    return 0;
}
```

or,

```
#include <iostream>
using namespace std;
int main()
{
    int x=10,y=20,sum;
    int *ptr1=&x,*ptr2=&y;

    sum = *ptr1 + *ptr2;
    printf("Sum = %d\n",sum);
    return 0;
}
```

## **Output:**

Sum = 30

- **swapping two numbers using pointer.**

```
#include <iostream>
using namespace std;
int main()
{
    int x=10,y=20,temp;
    int *ptr1,*ptr2;

    ptr1 = &x;
    ptr2 = &y;

    // swapping
    temp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = temp;
    printf("x = %d\n",x);
    printf("y = %d\n",y);
    return 0;
}
```

### **Output:**

x = 20  
y = 10

- **swapping two numbers using pointer and function.**

```
#include <iostream>
using namespace std;
void swapping(int *ptr1, int *ptr2)
{
    int temp;
    temp=*ptr1;
    *ptr1=*ptr2;
    *ptr2=temp;
}
int main()
{
    int x=10,y=20;
    printf("Before swapping : x=%d, y=%d\n",x,y);
    swapping(&x,&y);
    printf("After swapping : x=%d, y=%d\n",x,y);
    return 0;
}
```

### **Output:**

Before swapping : x=10, y=20

After swapping : x=20, y=10

- **Access array elements using pointer.**

```
#include <iostream>
using namespace std;
int main()
{
    int a[5]={10,20,30,40,5000000};
    int *ptr;
    int i;

    ptr=&a[0];

    for(i=0;i<5;i++)
    {
        printf("%d\n",*ptr);
        ptr++;
    }
}
```

or,

```
#include <iostream>
using namespace std;
int main()
{
    int a[5]={10,20,30,40,5000000};
    int *ptr;
    int i;
    for(i=0;i<5;i++)
    {
        ptr=&a[i];
        printf("%d\n",*ptr);
    }
}
```

## **NULL Pointers in C++ :**

- It is always a good practice to assign a NULL value to a pointer variable in case you do not have exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer.
- The NULL pointer is a constant with a value of zero defined in several standard libraries.

**Consider the following program:**

```
#include <iostream>
using namespace std;
int main()
{
    int *ptr = NULL;
    printf("The value of ptr is : %x\n",ptr);
    return 0;
}
```

### **Output:**

The value of ptr is : 0

- When the above code is compiled and executed, it produces the following result:

```
#include<conio.h>
#include<stdio.h>
int main()
{
    int *ptr1,*ptr2,a,b;
    clrscr();
    printf("Enter two numbers\n");
    scanf("%d%d",&a,&b);
    printf("Given numbers are %d and %d\n",a,b);
    ptr1=&a;
    ptr2=&b;
    printf("Address of a is %x and that of b is %x\n",ptr1,ptr2);
    printf("Sum of %d and %d is %d\n",a,b,*ptr1+*ptr2);
    getch();
    return 0;
}
```

Variable name ----->



Address ----->

- On most of the operating systems, programs are not permitted to access memory at address 0 because that memory is reserved by the operating system. However, the memory address 0 has special significance; it signals that the pointer is not intended to point to an accessible memory location. But by convention, if a pointer contains the null (zero) value, it is assumed to point to nothing.
- To check for a null pointer you can use an if statement as follows:

```
if (ptr)      /* succeeds if p is not null */
if (!ptr)     /* succeeds if p is null */
```

```
#include <iostream>
using namespace std;
int main()
{
    int *ptr=NULL;
    printf("The value of ptr is %d\n",ptr);
    if(ptr){
        printf("I am going to die\n");
    }
    else if(!ptr){
        printf("I am not going to die\n");
    }
    return 0;
}
```

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    int x=100;
    int *p = NULL;
    printf("The value of x: %d\n",x);
    printf("The value of *p: %d\n",*p);
    return 0;
}
```

### **Output:**

The value of x: 100

But the value of p does not printed because p isn't pointing any variable but we are trying to print \*p.

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    int x=100;
    int *p = NULL;
    printf("The value of x: %d\n",x);
    p=&x;
    printf("The value of *p: %d\n",*p);
    return 0;
}
```

### **Output:**

The value of x: 100  
The value of \*p: 100

- **Strings and pointer :**
- **Write a program that will print a string and address of the strings.**

```
#include <iostream>
using namespace std;
int main()
{
    char s[]="Bangladesh";
    printf("Name of our country is , s = %s\n",s);
    printf("Memory Address of s: %d\n",s);
    printf("Memory Address of s: %d\n",&s[0]);
    printf("Memory Address of s: %d\n",&s[1]);
    return 0;
}
```

### **Output:**

Name of our country is , s = Bangladesh  
Memory Address of s: 6422293  
Memory Address of s: 6422293  
Memory Address of s: 6422294

Here

And we do not need any ampersand (&) sign before string variable because it take ampersand (&) sign automatically but when you want to watch the single character address of string you must have to give the ampersand sign infornt of s[l] (s[0]) otherwise it will show the assci value of the character of the string.

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    char s[]="Bangladesh";
    char *p;
    p=s;
    printf("Name of our country : %s\n",p);
    printf("Memory address of s : %d\n",p);
    printf("Memory address of s : %d\n",s);
    return 0;
}
```

### **Output:**

Name of our country : Bangladesh  
Memory address of s : 6422289  
Memory address of s : 6422289

Here,

And we do not need any ampersand (&) sign before string variable for assigning the address of s in p because string variable take ampersand (&) sign automatically. Here we also do not need to give '\*' sign before p to print the string. If we use '\*' sign before p then there will be no output.

- **What will be the output of the program given bellow:**

```
#include <iostream>
using namespace std;
int main()
{
    char c1='A',c2='B',c3='C';
    char *p1,*p2,*p3;
    p1=&c1;
    p2=&c2;
    p3=&c3;
    printf("%c,%c,%c\n",*p1,*p2,*p3);
    return 0;
}
```

### **Output:**

A,B,C

- **What will be the output of the program given bellow:**

```
#include <iostream>
using namespace std;
int main()
{
    char *ptr="Bangladesh";
    printf("Name of our country is , ptr = %s\n",ptr);
    printf("Memory Address of ptr: %p\n",ptr);
    return 0;
}
```

**Output:**

Name of our country is , ptr = Bangladesh  
Memory Address of ptr: 00405064

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    char *ptr[2];
    printf("Enter your name: ");
    scanf(" %[^\n]",&ptr);
    printf("Your name is = %s\n",ptr);
    printf("Memory Address of ptr: %p\n",ptr);
    return 0;
}
```

**Output:**

Enter your name: Hamim Talukder  
Your name is = Hamim Talukder  
Memory Address of ptr: 0061FF18

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    char *ptr[2][30];
    printf("Enter your name : ");
    scanf(" %[^\n]",&ptr[0]);
    printf("Enter your future wife name : ");
    scanf(" %[^\n]",&ptr[1]);
    printf("%s + %s",ptr[0],ptr[1]);
    return 0;
}
```

### **Output:**

Enter your name : Hamim Talukder  
Enter your future wife name : Hridi Chowdhury  
Hamim Talukder + Hridi Chowdhury

- **Pointer's pointer :**
- **Write a program that will print a variable address and a pointer address which containing the address of the variable.**

```
#include <iostream>
using namespace std;
int main()
{
    char c='A';
    char *p;
    p=&c;
    printf("Address of c : %p\n",p);
    printf("Address of p : %p\n",&p);
    return 0;
}
```

**Output:**

Address of c : 0061FF1F  
Address of p : 0061FF18

- Write a program that will print a character type variable's value using Pinter's pointer.

```
#include <iostream>
using namespace std;
int main()
{
    char c='A';
    char *p,**q;
    p=&c;
    q=&p;
    printf("Value of c : %c\n",c);
    printf("Value of c : %c\n",*p);
    printf("Value of c : %c\n",**q);
    printf("Memori address of c : %d\n",p);
    printf("Memori address of c : %d\n",*q);
    printf("Memori address of p : %d\n",&p);
    printf("Memori address of p : %d\n",q);
    printf("Memori address of q : %d\n",&q);

    return 0;
}
```

### **Output:**

Value of c : A  
Value of c : A  
Value of c : A  
Memori address of c : 6422303  
Memori address of c : 6422303  
Memori address of p : 6422296  
Memori address of p : 6422296  
Memori address of q : 6422292

- **What will be the output of the program given below:**

```
#include <iostream>
using namespace std;
int main()
{
    char c='A';
    char *p,**q;
    p=&c;
    q=&p;
    **q='B';
    printf("Value of c : %c\n",c);
    printf("Value of c : %c\n",*p);
    printf("Value of c : %c\n",**q);
    return 0;
}
```

### **Output:**

Value of c : B  
Value of c : B  
Value of c : B

- **What will be the output of the program given bellow.**

```
#include <iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    int ara[] = {100,300,500,700,900};
    int *p;
    p = ara;
    printf("*p : %d\n",*p);
    printf("*p + 1 : %d\n",*p+1);
    printf("*(p+1) : %d\n",*(p+1));
    printf("*p+2 : %d\n",*p+2);
    printf("*(p+2) : %d\n",*(p+2));
    return 0;
}
```

- **What will be the output of the program given bellow.**

```
#include <iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    char *str = "Bangladesh";
    printf("%c,%c,%c,%c\n",*str,*(str+1),*(str+2),*(str+3));

    printf("%c,%c,%c,%c\n",*str,*str+1,*str+2,*str+3);
    return 0;
}
```

- **What will be the output of the program given bellow.**

```
#include <iostream>
using namespace std;
int main()
{
    char *p, a = 10;
    int *q, b = 'F';
    double *r, c = 302.64;
    p = &a;
    q = &b;
    r = &c;
    printf("Size of char: %d byte\n", sizeof(char));
    printf("p : %d\n", p);
    printf("p+1: %d\n", p + 1);
    printf("p+2: %d\n", p + 2);

    printf("Size of int: %d byte\n", sizeof(int));
    printf("q : %d\n", q);
    printf("q+1: %d\n", q + 1);
    printf("q+2: %d\n", q + 2);

    printf("Size of double: %d byte\n",
           sizeof(double));
    printf("r : %d\n", r);
    printf("r+1: %d\n", r + 1);
    printf("r+2: %d\n", r + 2);
    return 0;
}
```

- **Void pointer:**

A void pointer is a pointer that has no associated data type with it. A void pointer can hold address of any type and can be typecasted to any type.

**void pointer declaration:** void \*ptr;

In void pointer to dereference/print the value which we pointed by pointer, we have to type cast the pointer variable, otherwise, it can not be possible to print.

**Pointer type cast system:** \*((int\*) ptr)

- **What will be the output of the program given below.**

```
#include <iostream>
using namespace std;
int main()
{
    void *vp;
    int n=10;
    vp=&n;
    printf("Address of n: %p\n",&n);
    printf("Value of vp: %p\n",vp);
    printf("Content of vp: %d\n",*((int*)vp));

    return 0;
}
```

- **Function pointer:**

In C, like normal data pointers (int \*, char \*, etc), we can have pointers to functions.

**Function pointer:** int (\*fnc) (int, int);

- **What will be the output of the program given below.**

```
#include <iostream>
using namespace std;
int add(int n1, int n2)
{
    return n1 + n2;
}

int sub(int n1, int n2)
{
    return n1 - n2;
}

int main()
{
    int (*fnc)(int, int);
    int n1=10, n2=5;
    fnc=&add;
    printf("Result : %d\n", fnc(n1,n2));
    fnc=&sub;
    printf("Result : %d\n", fnc(n1,n2));
    return 0;
}
```

Here,

The main advantage of a function pointer is that we can use any function as the parameter of other function.

- **What will be the output of the program given below.**

```
#include <iostream>
using namespace std;
int add(int n1, int n2)
{
    return n1 + n2;
}

int sub(int n1, int n2)
{
    return n1 - n2;
}

int operate (int (*op) (int, int), int a, int b)
{
    return op(a, b);
}

int main()
{
    int n1 = 10, n2 = 5;
    printf("Result : %d\n", operate(&add, n1, n2));
    printf("Result : %d\n", operate(&sub, n1, n2));
    return 0;
}
```

```
or,  
#include <iostream>  
using namespace std;  
int add(int n1, int n2)  
{  
    return n1 + n2;  
}  
  
int sub(int n1, int n2)  
{  
    return n1 - n2;  
}  
  
int operate (int (*op) (int, int), int a, int b)  
{  
    return op(a, b);  
}  
  
int main()  
{  
    int n1 = 10, n2 = 5;  
    printf("Result : %d\n", operate(add, n1, n2));  
    printf("Result : %d\n", operate(sub, n1, n2));  
    return 0;  
}
```

- **qsort and bsearch:**

You all must know that if we have multiple data, then we can sort them in a particular order. This sort of work is called sorting. There are many different algorithms for sorting, one of the most popular being QuickSort. When you read Algorithms, you will surely learn QuickSort Algorithm and be impressed by its beauty. Until then, you can use the function called qsort in the stdlib.h header file. The prototype of the function looks like this:

```
void qsort (void* base, size_t items, size_t size,  
int (*compar) (const void*, const void*));
```

The function name is qsort, the return type is void (ie the function will not return anything). Now let's look at the parameters of the function.

The first parameter is a void pointer called base. That's the array we want to sort

So, the pointer to its first element is this base. But the qsort function doesn't know any

array of types to be sorted. So here the void pointer is being used as if it were an integer

We can sort any array like array, float array, structure array etc.

The second parameter is nitems , which indicates the total number of elements in the array. The third parameter is size, the number of bytes an element occupies in the array to be sorted must be stored in size.

The fourth parameter is a pointer to a function. The function is to compare two objects. will be used. The function will take two const void\* as parameters and return an integer. The reason for using void\* is that we don't know what type of object to compare, so we can pass any type of variable. And putting const before it means that the values we send will not be changed.

Now let's write a program to see how to use qsort.

- **What will be the output of the program given bellow.**

```
#include <iostream>
#include<stdlib.h>
using namespace std;
int comparefunc (const void * a, const void * b)
{
    return (*(int*) a - *(int*)b);
}

int main(){
int i, n=5;
int values[] = { 65, 6, 100, 1, 250 };
qsort(values, 5, sizeof(int), comparefunc);
```

```
for (i=0; i < n; i++) {  
    printf("%d ", values[i]);  
}  
printf("\n");  
return 0;  
}
```

If you run the program you will see the output: 1  
6 65 100 250 .

One thing to note is that inside the comparefunc we are on an integer pointer converted because we now know that our array is an array of integers.

If we wanted to sort in ascending to descending order, we would write b - a instead of a - b inside comparefunc (of course the pointer must be cast).

Another such function is bsearch, which performs a binary search. Binary search is discussed in detail in my book Computer Programming Volume 1, if you forget that book you can check from it. The prototype of the function looks like this:

```
void bsearch(const void *key, const void  
*base, size_t nitems, size_t size, int (*compar)  
(const void *, const void *));
```

Also see void pointer and function pointer used here. The return type of the function is void pointer. Because the function will return the data type of the variable we are searching for, and since there is no way to know in advance what type of variable will be returned, void pointers are used.

The first parameter of the function is a void pointer (key) that we will search for in the array. And the next things are same as above qsort function. Here also we need to write a comparer function. Let's see a complete example.

- **What will be the output of the program given bellow.**

```
#include <iostream>
#include<stdlib.h>
using namespace std;

int comparefunc (const void * a, const void * b)
{
    return (* (int*) a - *(int*)b);
}

int main()
{
    int key, *item, n=5;
    int values[]={1, 2, 5, 8, 10};

    while (1) {
```

```
printf("Enter the value of the key (0 to exit): ");
scanf("%d", &key);
if (key == 0) {
    break;
}
item = (int *) bsearch (&key, values, n,
sizeof(int), comparefunc);

if (item != NULL)
{
    printf("Item found: %d\n", *item);
}

else {
    printf("Item not found in array\n");
}
}
return 0;
}
```

Here,

If you take a good look at the program, the array in which we need to find something, but that array must be sorted beforehand. Otherwise binary search cannot be performed. Now your task with writing a program that first sorts an array using `qsort`, then searches the array using `bsearch`. And in the examples I've shown, I've used integer arrays. But you can use any type of array, even an array of structures.

## C Pointers in Detail :

Concept	Description
C - Pointer arithmetic	There are four arithmetic operators that can be used on pointers: ++, --, +, -
C - Array of pointers	You can define arrays to hold a number of pointers.
C - Pointer to pointer	C allows you to have pointer on a pointer and so on.
Passing pointers to functions in C	Passing an argument by reference or by address both enable the passed argument to be changed in the calling function by the called function.
Return pointer from functions in C	C allows a function to return a pointer to local variable, static variable and dynamically allocated memory as well.

### C - Pointer arithmetic :

- C pointer is an address, which is a numeric value. Therefore, you can perform arithmetic operations on a pointer just as you can a numeric value.
- There are four arithmetic operators that can be used on pointers: ++, --, +, and -
- To understand pointer arithmetic, let us consider that ptr is an integer pointer which points to the address 1000. Assuming 32-bit integers, let us perform the following arithmetic operation on the pointer:  
$$++\text{ptr}$$
- Now, after the above operation, the ptr will point to the location 1004 because each time ptr is incremented, it will point to the next integer location which is 4 bytes next to the current location.
- If ptr points to a character whose address is 1000, then above operation will point to the location 1001 because next character will be available at 1001.

## **Incrementing a Pointer :**

- We prefer using a pointer in our program instead of an array because the variable pointer can be incremented, unlike the array name which cannot be incremented because it is a constant pointer.

- Make a program that will increments the variable pointer to access each succeeding element of the array.

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main()
{
    int var[]={10,100,200};
    int i, *ptr;
    /* let us have array address in pointer */
    ptr = var;
    for(i=0;i<MAX;i++)
    {
        printf("Address of var[%d] = %x\n",i,ptr);
        printf("Value of var[%d] = %d\n",i,*ptr);

        /* move to the next location */
        ptr++;
    }
    return 0;
}
```

### **Output:**

Address of var[0] = 61ff0c  
Value of var[0] = 10  
Address of var[1] = 61ff10  
Value of var[1] = 100  
Address of var[2] = 61ff14  
Value of var[2] = 200

## **Decrementing a Pointer :**

- Make a program which will decreases its value by the number of bytes of its data.**

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main()
{
    int var[]={10,100,200};
    int i, *ptr;
    /* let us have array address in pointer */
    ptr = &var[MAX-1];
    for(i=MAX;i>0;i--)
    {
        printf("Address of var[%d] = %x\n",i,ptr);
        printf("Value of var[%d] = %d\n",i,*ptr);

        /* move to the next location */
        ptr--;
    }
    return 0;
}
```

## **Output:**

```
Address of var[3] = 61ff14
Value of var[3] = 200
Address of var[2] = 61ff10
Value of var[2] = 100
Address of var[1] = 61ff0c
Value of var[1] = 10
```

## Pointer \* and ++ :

Expression	Meaning
*p++ or * (p++)	Value of expression is *p before increment; increment p later
(*p)++	Value of expression is *p before increment; increment *p later
++*p or * (++p)	Increment p first; value of expression is *p after increment
++*p or ++ (*p)	Increment *p first; value of expression is *p after increment

## Pointer Comparisons :

- Pointers may be compared by using relational operators, such as ==, <, and >.
- If p1 and p2 point to variables that are related to each other, such as elements of the same array, then p1 and p2 can be meaningfully compared.
- **Make a program that will modifies the previous example one by incrementing the variable pointer as long as the address to which it points is either less than or equal to the address of the last element of the array, which is &var[MAX - 1].**

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main()
{
    int var[]={10,100,200};
    int i, *ptr;
    /* let us have address of the first element in
pointer */
    ptr = var;
    i=0;
```

```
while(ptr <= &var[MAX -1])
{
    printf("Address of var[%d] = %x\n",i,ptr);
    printf("Value of var[%d] = %d\n",i,*ptr);

    /* point to the previous location */
    ptr++;
    i++;
}
return 0;
}
```

### **Output:**

Address of var[0] = 61ff0c  
Value of var[0] = 10  
Address of var[1] = 61ff10  
Value of var[1] = 100  
Address of var[2] = 61ff14  
Value of var[2] = 200

### **Array of pointers :**

- **Before we understand the concept of arrays of pointers, let us consider the following example, which makes use of an array of 3 integers:**

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main()
{
    int var[]={10,100,200};
    int i;
    for(i=0;i<MAX;i++)
    {
        printf("Value of var[%d] = %d\n",i,var[i]);
    }
    return 0;
}
```

### **Output:**

Value of var[0] = 10  
Value of var[1] = 100  
Value of var[2] = 200

- There may be a situation when we want to maintain an array, which can store pointers to an int or char or any other data type available. Following is the declaration of an array of pointers to an integer:

```
int *ptr[ ];
```

- This declares ptr as an array of integer pointers. Thus, each element in ptr, now holds a pointer to an int value. Following example makes use of three integers, which will be stored in an array of pointers as follows:

- **Make a program that will show use of three integers, which will be stored in an array of pointers.**

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main()
{
    int var[]={10,100,200};
    int i,*ptr[MAX];

    for(i=0;i<MAX;i++)
    {
        ptr[i] = &var[i]; /* assign the address of
                           integer. */
    }

    for(i=0;i<MAX;i++)
    {
        printf("Value of var[%d] = %d\n",i,*ptr[i]);
    }
    return 0;
}
```

### **Output:**

Value of var[0] = 10  
Value of var[1] = 100  
Value of var[2] = 200

- **Make a program that will store a list of strings using an array of pointers to character.**

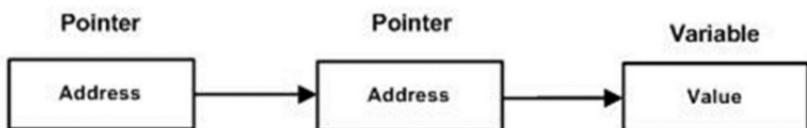
```
#include <iostream>
using namespace std;
const int MAX = 4;
int main()
{
    char *names[] = {
        "Hamim Ali",
        "Hameam Ali",
        "Haameem Ali",
        "Hameem Ali",
    };
    for(int i=0;i<MAX;i++)
    {
        printf("Value of var[%d] = %s\n",i,names[i]);
    }
    return 0;
}
```

### **Output:**

Value of var[0] = Hamim Ali  
Value of var[1] = Hameam Ali  
Value of var[2] = Haameem Ali  
Value of var[3] = Hameem Ali

## **Pointer to Pointer :**

- A pointer to a pointer is a form of multiple indirection, or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.



- A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example, following is the declaration to declare a pointer to a pointer of type int:

```
int **var;
```

- **When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the example:**

```
#include <iostream>
using namespace std;
const int MAX = 4;
int main()
{
    int var;
    int *ptr;
    int **pptr;
    var = 3000;

    ptr = &var;

    pptr = &ptr;

    printf("Value of var = %d\n",var);
    printf("Value available at *ptr = %d\n",*ptr);
    printf("Value available at **pptr = %d\n",**pptr);
    return 0;
}
```

### **Output:**

Value of var = 3000  
Value available at \*ptr = 3000  
Value available at \*\*pptr = 3000

## **Passing pointers to functions :**

- C programming language allows you to pass a pointer to a function. To do so, simply declare the function parameter as a pointer type.
- **Following is a simple example where we pass an unsigned long pointer to a function and change the value inside the function which reflects back in the calling function:**

```
#include <iostream>
#include<time.h>
using namespace std;

void getSeconds(unsigned long *par);
int main()
{
    unsigned long sec;
    getSeconds(&sec);

    /* print the actual value */
    printf("Number of seconds: %ld\n",sec);
    return 0;
}

void getSeconds(unsigned long *par)
{
    /* get the current number of seconds */
    *par = time (NULL);
    return;
}
```

## **Output:**

Number of seconds: 1662029220

- The function, which can accept a pointer, can also accept an array as shown in the following example:**

```
#include <iostream>
using namespace std;
/* function declaration */
double getAverage(int *arr, int size);
int main()
{
    /* an int array with 5 elements */
    int balance[5]={1000,2,3,17,50};
    double avg;

    /* pass pointer to the array as an argument */
    avg = getAverage(balance,5);

    /* output the return value */
    printf("Average value is :%f\n",avg);

    return 0;
}
double getAverage(int *arr,int size)
{
    int i,sum = 0;
    double avg;
```

```
for(i = 0; i<size; ++i){  
    sum += arr[i];  
}  
avg = (double)sum/size;  
return avg;  
}
```

## Return pointer from functions :

- C allows us to return a pointer from a function. To do so, you would have to declare a function returning a pointer as in the following example:

```
int * myFunction()  
{  
    .  
    .  
    .  
}
```

- Second point to remember is that, it is not good idea to return the address of a local variable to outside of the function so you would have to define the local variable as static variable.

- Consider the following function which will generate 10 random numbers and return them using an array name which represents a pointer i.e., address of first array element.

```
#include <iostream>
#include<time.h>
using namespace std;

/* function to generate and return random
numbers.*/
int *getRandom()
{
    static int r[10];
    int i;

    /* set the seed */
    srand((unsigned)time(NULL));
    for (i = 0; i < 10; ++i)
    {
        r[i] = rand();
        printf("%d\n", r[i]);
    }
    return r;
}

/* main function to call above defined function */
int main()
{
    /* &a pointer to an int */
```

```
int *p;
int i;
p = getRandom();
for (i = 0; i < 10; i++){
    printf(*(p+[%d]) : %d\n", i, *(p + i));
}
return 0;
}
```

## **Output:**

20989  
3705  
2994  
20654  
3506  
11499  
7648  
10965  
16750  
28264  
\*(p+[0]) : 20989  
\*(p+[1]) : 3705  
\*(p+[2]) : 2994  
\*(p+[3]) : 20654  
\*(p+[4]) : 3506  
\*(p+[5]) : 11499  
\*(p+[6]) : 7648  
\*(p+[7]) : 10965  
\*(p+[8]) : 16750  
\*(p+[9]) : 28264

## 4. DMA

---

- **Dynamic Memory Allocation in C++ using malloc(), calloc(), free() and realloc():**

Since C is a structured language, it has some fixed rules for programming. One of them includes changing the size of an array. An array is a collection of items stored at contiguous memory locations.

40	55	63	17	22	68	89	97	89
0	1	2	3	4	5	6	7	8

-> Array Indices

Array Length = 9  
First Index = 0  
Last Index = 8

As it can be seen that the length (size) of the array above made is 9. But what if there is a requirement to change this length (size). For Example,

If there is a situation where only 5 elements are needed to be entered in this array. In this case, the remaining 4 indices are just wasting memory in this array. So there is a requirement to lessen the length (size) of the array from 9 to 5.

Take another situation. In this, there is an array of 9 elements with all 9 indices filled. But there is a need to enter 3 more elements in this array. In this case, 3 indices more are required. So the length (size) of the array needs to be changed from 9 to 12.

This procedure is referred to as Dynamic Memory Allocation in C.

Therefore, C Dynamic Memory Allocation can be defined as a procedure in which the size of a data structure (like Array) is changed during the runtime.

C provides some functions to achieve these tasks. There are 4 library functions provided by C defined under `<stdlib.h>` header file to facilitate dynamic memory allocation in C programming.

They are:

- `malloc()`
- `calloc()`
- `free()`
- `realloc()`

### **C malloc() method:**

The “malloc” or “memory allocation” method in C is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form. It doesn’t Initialize memory at execution time so that it has initialized each block with the default garbage value initially.

### **Syntax:**

`ptr = (cast-type*) malloc(byte-size)`

For Example:

`ptr = (int*) malloc(100 * sizeof(int));`

Since the size of int is 4 bytes, this statement will allocate 400 bytes of memory. And, the pointer

ptr holds the address of the first byte in the allocated memory.

### **malloc function:**

prototype:

```
void * malloc ( size_t size )
```

Input :

size\_t size : It takes the memory space in bytes as input.

Return value :

void \* : After doing memory allocation it returns void pointer of that address.

Action/work :

size takes bytes equivalent space from memory and returns a void pointer in that address.

- **What will be the output of the program given bellow.**

```
#include <iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    int *marks;
    marks = (int *) malloc(sizeof(int));
    printf("Enter your age : ");
    scanf("%d",marks);
    printf("Your age is : %d\n", *marks);
    return 0;
}
```

**Output:**

Enter your age : 23  
Your age is : 23

- What will be the output of the program given below:

```
#include <stdio.h>

int *getvalue()
{
    int x;
    x = 6;
    return &x;
}

int main()
{
    int *var = getvalue();
    printf("%d\n", var);
    printf("%d\n", *var);
}
```

### **Output:**

Hamim.cpp: In function 'int\* getvalue()':

Hamim.cpp:5:9: warning: address of local variable 'x'  
returned [-Wreturn-local-addr]

```
int x;  
^
```

0

Here,

All functions take place in the stack section of RAM one by one. First main() function then getvalue(). So when getvalue() function is called it creates a reference variable ( x ) in it and our expectation was that it will return the variable x address in the main() function's pointer variable \*var and it will print the memory address and also the value of this address in the main() function. But in the output, we see that it is showing us an error and also printing a NULL pointer value (0).

Now the question is why it is happening.

The answer is when stack memory starts popping (deleting) memory address first it delete getvalue() function. So the memory address of variable x gets destroyed in the getvalue. So in the main() function we actually haven't any variable address of variable x in var pointer variable. That's why we get the error in the output.

Now the question is how we can solve this problem.

For solving this problem we have to do dynamic memory allocation. Because dynamic memory allocation functions allocate the memory address outside the stack. It allocates the memory in the heap section of RAM in runtime. So when in stack

section the getvalue() function got deleted with variable x it will not effect variable x value. Because the memory for this value 6 is stored in the heap section. That's why we will get our desired output.

- **After doing dynamic memory allocation we get our desired output.**

```
#include<stdio.h>
#include<stdlib.h>

int* getvalue(){
    int* x=(int*)malloc(4);
    *x=6;
    return x;
}

int main(){
    int *var = getvalue();
    printf("Memory address of x variable is %d\n", var);
    printf("Value of x is %d\n", *var);
}
```

### **Output:**

Memory address of x variable is 12595112  
Value of x is 6

- Write a program that can take the marks of each student of a class and show them using pointer.

```
#include <iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    int *marks;
    int i, n;
    printf("Please enter the number of students: ");
    scanf("%d",&n);
    // now allocate memory
    marks = (int *) malloc(n * sizeof(int));
    printf("Enter the marks for each student: \n");
    for(i=0;i<n;i++){
        scanf("%d", &marks[i]);
    }

    printf("Now here you can see the values:\n");
    // now print the marks array
    for(i=0;i<n;i++){
        printf("%d\n",marks[i]);
    }
    return 0;
}
```

Here,

The malloc function will return a pointer and gives us the starting memory address from the allocated memory address. But if anything goes wrong doing the memory allocation then it will return NULL. If memory allocation is done in this way then it will take place in the heap segment of memory.

Here we use (int\*) before malloc for doing type cast and convert the type of allocated memory address into integer form.

### **C calloc() method:**

"calloc" or "contiguous allocation" method in C is used to dynamically allocate the specified number of blocks of memory of the specified type. It is very much similar to malloc() but has two different points and these are:

1. It initializes each block with a default value '0'.
2. It has two parameters or arguments as compare to malloc().

## Syntax:

```
ptr = (cast-type*)calloc(n, element-size);
```

here, n is the no. of elements and element-size is the size of each element.

For Example:

```
ptr = (float*) calloc(25, sizeof(float));
```

This statement allocates contiguous space in memory for 25 elements each with the size of the float.

- **Write a program that can take the marks of each student of a class and show them using pointer.**

```
#include <iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    int *marks;
    int i, n;
    printf("Please enter the number of students: ");
    scanf("%d",&n);
    // now allocate memory
    marks = (int *) calloc(n,sizeof(int));
    printf("Enter the marks for each student: \n");
    for(i=0;i<n;i++){
        scanf("%d", &marks[i]);
    }
    printf("Now here you can see the values:\n");
```

```
// now print the marks array
for(i=0;i<n;i++){
    printf("%d\n",marks[i]);
}
return 0;
}
```

### **C free() method:**

“free” method in C is used to dynamically de-allocate the memory. The memory allocated using functions malloc() and calloc() is not de-allocated on their own. Hence the free() method is used, whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.

### **Syntax:**

```
free(ptr);
```

### **Prototype:**

```
void free ( void * ptr)
```

Input:

void \* ptr : It takes the address of allocated memory using malloc, calloc function.

return value:

void ; it returns no value .

action:

ptr free up the allocated space of memory location ( deallocation).

- Write a program that can take the marks of each student of a class and show them using pointer after then free up the allocated memory space.

```
#include <iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    int *marks;
    int i, n;
    printf("Please enter the number of students: ");
    scanf("%d",&n);
    // now allocate memory
    marks = (int *) malloc( n * sizeof(int));
    printf("Enter the marks for each student: \n");
    for(i=0;i<n;i++){
        scanf("%d", &marks[i]);
    }
    printf("Now here you can see the values:\n");
    // now print the marks array
    for(i=0;i<n;i++){
        printf("%d\n",marks[i]);
    }
    // now free up the memory
    free (marks);
    printf("After free-up/deallocate the allocated memory
: \n");
    for(i=0;i<n;i++){
        printf("%d\n",marks[i]);
    }
    return 0;
}
```

## **C++ realloc() method**

“realloc” or “re-allocation” method in C++ is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to dynamically re-allocate memory. re-allocation of memory maintains the already present value and new blocks will be initialized with the default garbage value.

### **Syntax:**

`ptr = (cast-type*) realloc(ptr, newSize);`  
where ptr is reallocated with new size 'newSize'.  
If space is insufficient, allocation fails and returns a NULL pointer.

### **Example:**

```
#include <iostream>
#include <stdlib.h>
using namespace std;
int main()
{
    int *ptr;
    int n, i;
    n = 5;
    printf("Enter number of elements: %d\n", n);
    ptr = (int *)calloc(n, sizeof(int));

    if (ptr == NULL)
    {
```

```
printf("Memory not allocated.\n");
exit(0);
}
else
{
printf("Memory successfully allocated using
calloc.\n");

for (i = 0; i < n; ++i)
{
ptr[i] = i + 1;
}

printf("The elements of the array are: ");

for (i = 0; i < n; ++i)
{
printf("%d, ", ptr[i]);
}

n = 10;
printf("\n\nEnter the new size of the array:
%d\n", n);

ptr = (int *) realloc(ptr, n * sizeof(int));

printf("Memory successfully re-allocated using
realloc.\n");

for (i = 5; i < n; ++i)
{
```

```
ptr[i] = i + 1;  
}  
  
printf("The elements of the array are: ");  
  
for (i = 0; i < n; ++i)  
{  
    printf("%d, ", ptr[i]);  
}  
  
free(ptr);  
}  
  
return 0;  
}
```

- Write a program that can take the marks of each student of class one to five and show them using pointer.

```
#include <iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    int *ara[5], num[5];
    int i, j, n;
    for(i=0;i<5;i++){
        printf("Enter the number of stdunts for class
%d : ",i+1);
        scanf("%d",&n);
        num[i] = n;
        ara[i] = (int *) malloc( n * sizeof(int) );
        if(ara[i] == NULL){
            printf("Memory allocation failed\n");
            return 1;
        }
        for(j=0;j<n;j++){
            printf("Enter marks for student %d: ",j+1);
            scanf("%d", &ara[i][j]);
        }
    }
    // now print the results
    printf("Output:\n");
    for(i=0;i<5;i++){
        printf("Class %d : ",i+1);
        for(j=0;j<num[i];j++){
            printf("%4d",ara[i][j]);
        }
    }
}
```

```
    printf("\n");
}
return 0;
}
```

- **Write a program that can take the marks of each student of any classes and show them using pointer.**

```
#include <iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    int **ara, num[12];
    int i, j, total_classes, n;
    printf("Enter the total number of classes : ");
    scanf("%d",&total_classes);
    ara = (int **)malloc(total_classes * sizeof(int *));
    if(ara == NULL){
        printf("Memory allocation failed\n");
        return 1;
    }
    for(i=0;i<total_classes;i++){
        printf("Enter the number of students for
class %d : ",i+1);
        scanf("%d",&n);
        num[i] = n;
        ara[i] = (int *) malloc ( n * sizeof(int) );
        if(ara[i] == NULL){
            printf("Memory allocation failed\n");
            return 1;
        }
    }
}
```

```
for(j=0;j<n;j++){
    printf("Enter marks for student %d: ",j+1);
    scanf("%d",&ara[i][j]);
}
// now print the results
printf("Output:\n");
for(i=0;i<total_classes;i++){
    printf("Class %d : ",i+1);
    for(j=0;j<num[i];j++){
        printf("%4d",ara[i][j]);
    }
    printf("\n");
}
return 0;
}
```

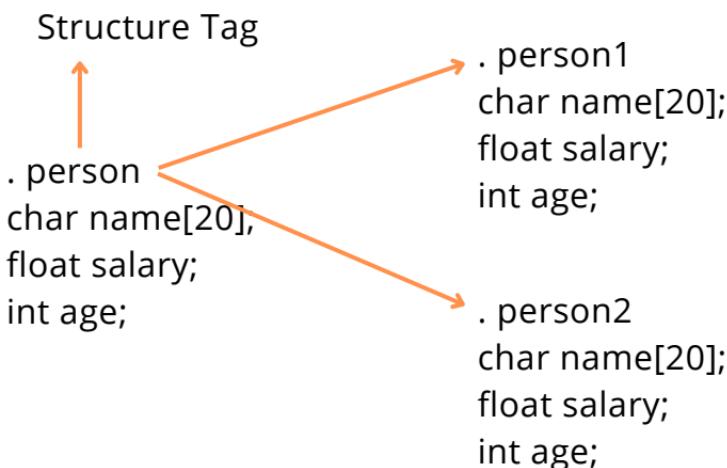
## 4. structure

---

Structure is a collection of variables of different types under a single type.

A structure is a user defined data type in C/C++.

Structure is a collection of different types under a single name.



- **What will be the output of the program bellowed?**

```
#include <iostream>
using namespace std;
// global structre
struct Person
{
    int age;
    float salary;
};

int main()
{
    struct Person man1, man2; // local variable
    man1.age=22;
    man1.salary=12300.12;

    printf("man1 :\n");
    printf("Age = %d\n",man1.age);
    printf("Salary = %.2f\n",man1.salary);

    printf("\n");

    man2.age=25;
    man2.salary=23500.12;

    printf("man2 :\n");
    printf("Age = %d\n",man2.age);
    printf("Salary = %.2f\n",man2.salary);
    return 0;
}
```

Here,

Person is a structure tag.

or,

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

```
int main()
```

```
{
```

```
    // local structure
```

```
    struct Person
```

```
{
```

```
        int age;
```

```
        float salary;
```

```
};
```

```
    struct Person man1, man2; // local variable
```

```
    man1.age=22;
```

```
    man1.salary=12300.12;
```

```
    printf("man1 :\n");
```

```
    printf("Age = %d\n",man1.age);
```

```
    printf("Salary = %.2f\n",man1.salary);
```

```
    printf("\n");
```

```
    man2.age=25;
```

```
    man2.salary=23500.12;
```

```
    printf("man2 :\n");
```

```
    printf("Age = %d\n",man2.age);
```

```
    printf("Salary = %.2f\n",man2.salary);
```

```
    return 0;
```

```
}
```

```
or,  
#include <iostream>  
using namespace std;  
// global structure  
struct Person  
{  
    int age;  
    float salary;  
};  
  
struct Person man1, man2; // global variable  
  
int main()  
{  
    man1.age=22;  
    man1.salary=12300.12;  
  
    printf("man1 :\n");  
    printf("Age = %d\n",man1.age);  
    printf("Salary = %.2f\n",man1.salary);  
  
    printf("\n");  
  
    man2.age=25;  
    man2.salary=23500.12;  
  
    printf("man2 :\n");  
    printf("Age = %d\n",man2.age);  
    printf("Salary = %.2f\n",man2.salary);  
    return 0;  
}
```

**Output:**

man1 :

Age = 22

Salary = 12300.12

man2 :

Age = 25

Salary = 23500.12

- **Make a program that will show given below using structure.**

**Enter information for man1:**

**Enter Age: 23**

**Enter salary: 65600**

**man1 :**

**Age = 23**

**Salary = 65600.00**

**Enter information for man2:**

**Enter Age: 45**

**Enter salary: 65000.889**

**man2 :**

**Age = 45**

**Salary = 65000.89**

```
#include <iostream>
using namespace std;
// global structure
struct Person
{
    int age;
    float salary;
};

int main()
{
    struct Person man1, man2; // local variable

    printf("Enter information for man1:\n");
    printf("Enter Age: ");
    scanf("%d",&man1.age);
    printf("Enter salary: ");
    scanf("%f",&man1.salary);

    printf("\nman1 :\n");
    printf("Age = %d\n",man1.age);
    printf("Salary = %.2f\n",man1.salary);
    printf("\n");

    printf("Enter information for man2:\n");
    printf("Enter Age: ");
    scanf("%d",&man2.age);
    printf("Enter salary: ");
    scanf("%f",&man2.salary);

    printf("\nman2 :\n");
    printf("Age = %d\n",man2.age);
    printf("Salary = %.2f\n",man2.salary);
    return 0;
}
```

- **What will be the output of the program bellowed?**

```
#include<iostream>

using namespace std;

struct Student{
    int age;
    char section;
};

int main(){
    cout<<sizeof(struct Student);
}
```

**Output:**

8

- **What will be the output of the program bellowed?**

```
#include<iostream>

using namespace std;

struct __attribute__((packed))Student{
    int age;
    char section;
};

int main(){
    cout<<sizeof(struct Student);
}
```

### **Output:**

5

- **What will be the output of the program bellowed?**

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

struct employee
{
    char *name;
    int age;
    int salary;
};

int manager()
{
    struct employee manager;
    manager.age = 27;
    if(manager.age>30)
        manager.salary = 65000;
    else
        manager.salary = 55000;
    return manager.salary;
}

int main()
{
    struct employee emp1;
    struct employee emp2;
    printf("Enter the salary of employee1: ");
    scanf("%d", &emp1.salary);
    printf("Enter the salary of employee2: ");
    scanf("%d", &emp2.salary);
```

```
printf("Employee 1 salary is: %d\n",emp1.salary);
printf("Employee 2 salary is: %d\n",emp2.salary);
printf("Manager's salary is %d", manager());
}
```

or,

```
#include <iostream>
using namespace std;
struct
{
    char *name;
    int age;
    int salary;
} emp1, emp2;
```

```
int manager()
```

```
{
    struct
    {
        char *name;
        int age;
        int salary;
    } manager;
```

```
    manager.age = 27;
    if(manager.age>30)
        manager.salary = 65000;
    else
        manager.salary = 55000;
    return manager.salary;
}
```

```
int main()
{
    printf("Enter the salary of employee1: ");
    scanf("%d", &emp1.salary);
    printf("Enter the salary of employee2: ");
    scanf("%d", &emp2.salary);
    printf("Employee 1 salary is: %d\n", emp1.salary);
    printf("Employee 2 salary is: %d\n", emp2.salary);
    printf("Manager's salary is %d", manager());
}
```

or,

```
#include <iostream>
using namespace std;
struct employee
{
    char *name;
    int age;
    int salary;
}emp1;
```

```
int manager()
{
    struct employee manager;
    manager.age = 27;
    if(manager.age>30)
        manager.salary = 65000;
    else
        manager.salary = 55000;
    return manager.salary;
}
```

```
int main()
{
    struct employee emp2;
    printf("Enter the salary of employee1: ");
    scanf("%d", &emp1.salary);
    printf("Enter the salary of employee2: ");
    scanf("%d", &emp2.salary);
    printf("Employee 1 salary is: %d\n",emp1.salary);
    printf("Employee 2 salary is: %d\n",emp2.salary);
    printf("Manager's salary is %d", manager());
}
```

### **Output:**

```
Enter the salary of employee1: 12000
Enter the salary of employee2: 13000
Employee 1 salary is: 12000
Employee 2 salary is: 13000
Manager's salary is 55000
```

- **typedef keyword:**

typedef gives freedom to the user by allowing them to create their own types.

Syntax:

```
typedef existing_data_type new_data_type
```

- **What will be the output of the program bellowed?**

```
#include <iostream>
using namespace std;
typedef int integer;

int main()
{
    integer var1 = 100;
    int var2 = 200;
    printf("var1 = %d\n",var1);
    printf("var2 = %d\n",var2);
    return 0;
}
```

**Output:**

var1 = 100  
var2 = 200

- **What will be the output of the program given below.**

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

typedef struct car {
    char engine[50];
    char fuel_type[10];
    int fuel_tank_cap;
    int seating_cap;
    float city_milage;
}car;

int main()
{
    car c1, c2;
    printf("Enter c1 car engine name: ");
    fflush(stdin);
    gets(c1.engine);
    printf("Enter c2 car engine name: ");
    scanf(" %[^\n]",&c2.engine);
    printf("c1 car engine name: %s\n",c1.engine);
    printf("c2 car engine name: %s\n",c2.engine);
    return 0;
}
```

### **Output:**

```
Enter c1 car engine name: VIVIT_14
Enter c2 car engine name: VIVIT_15
c1 car engine name: VIVIT_14
c2 car engine name: VIVIT_15
```

- **What will be the output of the program bellowed?**

```
#include <iostream>
using namespace std;
// global structure
struct Person
{
    int age;
    float salary;
};

int main()
{
    struct Person man1 = {27,25550.256}; // local
variable
    struct Person man2, man3;

    // element wise assignment
    man2.age = 25;
    man2.salary = 27896.78;

    // structure variable assignment
    man3 = man2;

    printf("\nman1 :\n");
    printf("Age = %d\n",man1.age);
    printf("Salary = %.2f\n",man1.salary);

    printf("\n");

    printf("man2 :\n");
```

```
printf("Age = %d\n",man2.age);
printf("Salary = %.2f\n",man2.salary);
printf("\n");

printf("man3 :\n");
printf("Age = %d\n",man3.age);
printf("Salary = %.2f\n",man3.salary);
return 0;
}
```

**Output:**

man1 :  
Age = 27  
Salary = 25550.26

man2 :  
Age = 25  
Salary = 27896.78

man3 :  
Age = 25  
Salary = 27896.78

- **What will be the output of the program bellowed?**

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

```
struct car {
    char engine[50];
    char fuel_type[10];
    int fuel_tank_cap;
    int seating_cap;
    float city_milage;
}car;
```

```
int main()
{
    struct car c1={"DDis 190 Engine","Diesel", 37, 5,
19.74};
    struct car c2={"Kappa 200 Engine","Petrol", 22,
8, 14.74};
    printf("car c1 data:\n");
    printf("Engine: %s\n",c1.engine);
    printf("Fuel type: %s\n",c1.fuel_type);
    printf("Fuel tank capacity:
%d\n",c1.fuel_tank_cap);
    printf("Seating capacity: %d\n",c1.seating_cap);
    printf("City milage: %.2f\n",c1.city_milage);
    printf("\n");
    printf("car c2 data:\n");
    printf("Engine: %s\n",c2.engine);
    printf("Fuel type: %s\n",c2.fuel_type);
    printf("Fuel tank capacity:
%d\n",c2.fuel_tank_cap);
```

```
printf("Seating capacity: %d\n",c2.seating_cap);
printf("City milage: %.2f\n",c2.city_milage);
return 0;
}

or,
#include <iostream>
using namespace std;

struct car {
    char engine[50];
    char fuel_type[10];
    int fuel_tank_cap;
    int seating_cap;
    float city_milage;
}car;

int main()
{
    struct car c1={ .engine = "DDis 190 Engine",
    .fuel_type = "Diesel", .fuel_tank_cap = 37,
    .seating_cap = 5, .city_milage = 19.74};
    struct car c2={ .engine = "Kappa 200 Engine",
    .fuel_type = "Petrol", .fuel_tank_cap = 22,
    .seating_cap = 8, .city_milage = 14.74};
    printf("car c1 data:\n");
    printf("Engine: %s\n",c1.engine);
    printf("Fuel type: %s\n",c1.fuel_type);
    printf("Fuel tank capacity:
%d\n",c1.fuel_tank_cap);
    printf("Seating capacity: %d\n",c1.seating_cap);
```

```
printf("City milage: %.2f\n",c1.city_milage);
printf("\n");
printf("car c2 data:\n");
printf("Engine: %s\n",c2.engine);
printf("Fuel type: %s\n",c2.fuel_type);
printf("Fuel tank capacity: %d\n",c2.fuel_tank_cap);
printf("Seating capacity: %d\n",c2.seating_cap);
printf("City milage: %.2f\n",c2.city_milage);
return 0;
}
```

## **Output:**

car c1 data:  
Engine: DDis 190 Engine  
Fuel type: Diesel  
Fuel tank capacity: 37  
Seating capacity: 5  
City milage: 19.74

car c2 data:  
Engine: Kappa 200 Engine  
Fuel type: Petrol  
Fuel tank capacity: 22  
Seating capacity: 8  
City milage: 14.74

- Make a program that will show whether the structural elements of different structure variables are equal or not.

```
#include <iostream>
using namespace std;
// global structure
struct Person
{
    int age;
    float salary;
};

int main()
{
    struct Person man1 = {27,25550.256}; // local
variable
    struct Person man2;

    // element wise assignment
    man2.age = 27;
    man2.salary = 25550.256;

    if(man1.age == man2.age && man1.salary ==
man2.salary){
        printf("man1 equal to man2");
    }
    else
        printf("man1 not equal to man2");
    return 0;
}
```

- Make a program that will show given below using structure.

**Enter information for Person1**

**Enter age : 20**

**Enter salary : 12500.678**

**Enter information for Person2**

**Enter age : 22**

**Enter salary : 14500**

**Enter information for Person3**

**Enter age : 23**

**Enter salary : 45900.89**

**Enter information for Person4**

**Enter age : 40**

**Enter salary : 67000.67**

**Information for Person1**

**Age : 20**

**Salary : 12500.68**

**Information for Person2**

**Age : 22**

**Salary : 14500.00**

**Information for Person3**

**Age : 23**

**Salary : 45900.89**

**Information for Person4**

**Age : 40**

**Salary : 67000.67**

```
#include <iostream>
using namespace std;
// global structure
struct Person
{
    int age;
    float salary;
};

int main()
{
    struct Person person[4];
    int i;

    for(i=0;i<4;i++){
        printf("Enter information for
Person%d\n",i+1);
        printf("Enter age : ");
        scanf("%d",&person[i].age);
        printf("Enter salary : ");
        scanf("%f",&person[i].salary);
    }
    for(i=0;i<4;i++){
        printf("\n\nInformation for
Person%d\n",i+1);
        printf("Age : %d\n",person[i].age);
        printf("Salary : %.2f\n",person[i].salary);
    }
    return 0;
}
```

- Make a program that will show given below using structure.

**Enter information for Person1**

**Enter Name : Hamim Talukder**

**Enter age : 22**

**Enter salary : 34000.3535**

**Enter information for Person2**

**Enter Name : Safin**

**Enter age : 24**

**Enter salary : 23787.89**

**Enter information for Person3**

**Enter Name : Atik**

**Enter age : 45**

**Enter salary : 34000**

**Enter information for Person4**

**Enter Name : Munmun**

**Enter age : 23**

**Enter salary : 25678.896**

**Information for Person1**

**Name : Hamim Talukder**

**Age : 22**

**Salary : 34000.35**

**Information for Person2**

**Name : Safin**

**Age : 24**

**Salary : 23787.89**

### **Information for Person3**

**Name : Atik**

**Age : 45**

**Salary : 34000.00**

### **Information for Person4**

**Name : Munmun**

**Age : 23**

**Salary : 25678.90**

```
#include <iostream>
using namespace std;
// global structre
struct Person
{
    char name[50];
    int age;
    float salary;
};

int main()
{
    struct Person person[4];
    int i;

    for(i=0;i<4;i++){
        printf("Enter information for Person%d\n",i+1);
        printf("Enter Name : ");
        fflush(stdin);
        gets(person[i].name);
        printf("Enter age : ");
        scanf("%d",&person[i].age);
```

```
    printf("Enter salary : ");
    scanf("%f",&person[i].salary);
    printf("\n");
}
for(i=0;i<4;i++){
    printf("\n\nInformation for Person%d\n",i+1);
    printf("Name : %s\n",person[i].name);
    printf("Age : %d\n",person[i].age);
    printf("Salary : %.2f\n",person[i].salary);
}
return 0;
}
```

Here,

fflush(stdin) is a library function that is used to solve the string assigning problem.

- **Passing structure variable to function:**

- **What will be the output of the program bellowed?**

```
#include <iostream>
#include<string.h>
using namespace std;

// global structre
struct Person
{
    char name[50];
    int age;
    float salary;
};

void display(struct Person p)
{
    printf("Name : %s\n",p.name);
    printf("Age : %d\n",p.age);
    printf("Salary : %.2f\n",p.salary);
}

int main()
{
    struct Person person1,person2;
    strcpy(person1.name,"Hamim Talukder");
    person1.age = 27;
    person1.salary = 12250.50;
    display(person1);
    return 0;
}
```

## **Output:**

Name : Hamim Talukder

Age : 27

Salary : 12250.50

- What will be the output of the program bellowed?**

```
#include <iostream>
#include<string.h>
using namespace std;
// global structure
struct Person
{
    char name[50];
    int age;
    float salary;
};

void display(struct Person p)
{
    printf("Name : %s\n",p.name);
    printf("Age : %d\n",p.age);
    printf("Salary : %.2f\n",p.salary);
}

int main()
{
    struct Person person1,person2;
    strcpy(person1.name,"Hamim Talukder");
    person1.age = 22;
    person1.salary = 92250.50;
    display(person1);
```

```
strcpy(person2.name,"Hridi");
person2.age = 23;
person2.salary = 81250.50;
display(person2);
return 0;
}

or,
#include <iostream>
#include<string.h>
using namespace std;

// global structure
struct Person
{
    char name[50];
    int age;
    float salary;
};

void display(struct Person p)
{
    printf("Name : %s\n",p.name);
    printf("Age : %d\n",p.age);
    printf("Salary : %.2f\n",p.salary);
}

int main()
{
    struct Person person1;
    strcpy(person1.name,"Hamim Talukder");
    person1.age = 22;
```

```
person1.salary = 92250.50;  
display(person1);  
  
strcpy(person1.name,"Hridi");  
person1.age = 23;  
person1.salary = 81250.50;  
display(person1);  
return 0;  
}
```

### **Output:**

Name : Hamim Talukder

Age : 22

Salary : 92250.50

Name : Hridi

Age : 23

Salary : 81250.50

- **Pointer structure :**

- **What will be the output of the program bellowed?**

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

struct abc{
    int x;
    int y;
};

int main()
{
    struct abc a = {0, 1};
    struct abc *ptr = &a;

    printf("%d %d",(*ptr).x, (*ptr).y);
}
```

```
or,  
#include <iostream>  
using namespace std;  
struct abc{  
    int x;  
    int y;  
};  
int main()  
{  
    struct abc a = {0, 1};  
    struct abc *ptr = &a;  
  
    printf("%d %d",ptr->x, ptr->y);  
}
```

```
or,  
#include <iostream>  
using namespace std;  
struct abc{  
    int x;  
    int y;  
};  
int main()  
{  
    struct abc a = {0, 1};  
    struct abc *ptr = &a;  
  
    printf("%d %d",(*&a).x, (*&a).y);  
}
```

```
or,  
#include <iostream>  
using namespace std;  
struct abc{  
    int x;  
    int y;  
};  
int main()  
{  
    struct abc a = {0, 1};  
    struct abc *ptr = &a;  
  
    printf("%d %d", a.x, a.y);  
}
```

**Output:**

0 1

- **What will be the output of the program bellowed?**

```
#include<iostream>
#include<string.h>
#include<stdlib.h>

using namespace std;

typedef struct Person {
    char name[20];
    int age;
    float height;
} Person;

int main(void){
    Person* p1 = (Person*) malloc(sizeof(Person));
    strcpy((*p1).name , "Hridi Chowdhury");
    (*p1).age = 27;
    (*p1).height = 5.65;

    cout<<"Name = "<<(*p1).name<<endl;
    cout<<"Age = "<<(*p1).age<<endl;
    cout<<"Height = "<<(*p1).height<<endl;
}
```

### **Output:**

Name = Hridi Chowdhury  
Age = 27  
Height = 5.65

```
or,  
#include<iostream>  
#include<string.h>  
#include<stdlib.h>  
  
using namespace std;  
  
typedef struct Person {  
    char name[20];  
    int age;  
    float height;  
} Person;  
  
int main(void){  
    Person* p1 = (Person*) malloc(sizeof(Person));  
    strcpy(p1->name , "Hridi Chowdhury");  
    p1-> age = 27;  
    p1->height = 5.65;  
  
    cout<<"Name = "<<p1->name<<endl;  
    cout<<"Age = "<<p1->age<<endl;  
    cout<<"Height = "<<p1->height<<endl;  
}
```

### **Output:**

Name = Hridi Chowdhury  
Age = 27  
Height = 5.65

- **What will be the output of the program bellowed?**

```
#include <iostream>
#include <string.h>
#include <stdlib.h>

using namespace std;

typedef struct Person
{
    char name[20];
    int age;
    float height;
} Person;

int main(void)
{
    Person *p1 = (Person *)malloc(3 * sizeof(Person));

    for (int i = 0; i < 3; i++)
    {
        cout << i + 1 << " no person : "<<endl;
        cout <<"Name : ";
        cin >> p1[i].name;
        cout <<"Age : ";
        cin >> p1[i].age;
        cout <<"Height : ";
        cin >> p1[i].height;
    }
}
```

```
for (int i = 0; i < 3; i++)
{
    cout << i + 1 << " no person information : \n";
    cout << "Name = " << p1[i].name << endl;
    cout << "Age = " << p1[i].age << endl;
    cout << "Height = " << p1[i].height << endl<<endl;
}
}
```

**Output:**

1 no person :

Name : Hamim

Age : 23

Height : 5.9

2 no person :

Name : Hridi

Age : 24

Height : 5.56

3 no person :

Name : Jim

Age : 18

Height : 5.5

1 no person information :

Name = Hamim

Age = 23

Height = 5.9

2 no person information :

Name = Hridi

Age = 24

Height = 5.56

3 no person information :

Name = Jim

Age = 18

Height = 5.5

- **What will be the output of the program bellowed?**

```
#include <iostream>
#include <string.h>
#include <stdlib.h>

using namespace std;

typedef struct Person
{
    char name[20];
    int age;
    float height;
} Person;

int main(void)
{
    Person *p1 = (Person *)malloc(3 * sizeof(Person));

    for (int i = 0; i < 3; i++)
    {
        cout << i + 1 << " no person : "<<endl;
        cout <<"Name : ";
        scanf(" %[^\n]", &p1[i].name);
        cout <<"Age : ";
        cin >> p1[i].age;
        cout <<"Height : ";
        cin >> p1[i].height;
    }
}
```

```
for (int i = 0; i < 3; i++)
{
    cout << i + 1 << " no person information : \n";
    cout << "Name = " << p1[i].name << endl;
    cout << "Age = " << p1[i].age << endl;
    cout << "Height = " << p1[i].height << endl<<endl;
}
}
```

**Output:**

1 no person :

Name : Hamim Talukder

Age : 23

Height : 5.9

2 no person :

Name : Hridi Chowdhury

Age : 24

Height : 5.56

3 no person :

Name : Jim Talukder

Age : 18

Height : 5.5

1 no person information :

Name = Hamim Talukder

Age = 23

Height = 5.9

2 no person information :

Name = Hridi Chowdhury

Age = 24

Height = 5.56

3 no person information :

Name = Jim Talukder

Age = 18

Height = 5.5

- **Predict the output of the following C program.**

```
#include<stdio.h>
struct Point
{
    int x, y, z;
};
int main()
{
    struct Point p1 = {.y = 0, .z = 1, .x = 2};
    printf("%d %d %d\n", p1.x, p1.y, p1.z);
    return 0;
}
```

**Output:**

2 0 1

- **Predict the output of the following C program.**

```
#include<stdio.h>
struct Ournode
{
    char x, y, z;
};
int main()
{
    struct Ournode p = {'1', '0', 'a'+2};
    struct Ournode *q = &p;
    printf("%c, %c", *((char*)q+1), *((char*)q+2));
    return 0;
}
```

**Output:**

0, c

- **Predict the output of the following C program.**

```
#include<stdio.h>
int main()
{
    struct student {
        int id;
        char *name;
    };
    struct student one;
    one.id = 1;
    one.name = "Hamim Talukder";
    printf("ID: %d\nName: %s\n",one.id,one.name);
    return 0;
}
```

or,

```
#include<stdio.h>
#include<string.h>
int main()
{
    struct student {
        int id;
        char name[50];
    };
    struct student one;
    one.id = 1;
    strcpy(one.name,"Hamim Talukder");
    printf("ID: %d\nName: %s\n",one.id,one.name);
    return 0;
}
```

## **Output:**

ID: 1

Name: Hamim Talukder

- **Make a program that will show given below using structure.**

**Enter your ID: 5369**

**Enter your Name: Hamim Talukder**

**ID: 5369**

**Name: Hamim Talukder**

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

```
int main()
{
    struct student {
        int id;
        char name[ 50];
    };
    struct student one;
    printf("Enter your ID: ");
    scanf("%d",&one.id);
    printf("Enter your Name: ");
    scanf(" %[^\n]",&one.name);
    printf("ID: %d\nName: %s\n",one.id,one.name);
    return 0;
}
```

- **Predict the output of the following C program.**

```
#include<stdio.h>
#include<string.h>
struct nametype{
    char first[20];
    char last[20];
};
struct student{
    int id;
    struct nametype name;
};
int main()
{
    struct student one;
    printf("Enter ID: ");
    scanf("%d",&one.id);
    printf("Enter your first name: ");
    scanf("%s",one.name.first);
    printf("Enter your last name: ");
    scanf("%s",&one.name.last);
    printf("ID: %d\n",one.id);
    printf("Name: %s %s\n",one.name.first,
    one.name.last);
    return 0;
}
```

### **Output:**

Enter ID: 221

Enter your first name: Hamim

Enter your last name: Talukder

ID: 221

Name: Hamim Talukder

- Predict the output of the following C program.

```
#include <iostream>
#include<string.h>
using namespace std;

struct nametype{
    char first[20];
    char last[20];
};

struct student{
    int id;
    struct nametype name;
};

int main()
{
    struct student one;
    printf("Enter ID: ");
    scanf("%d",&one.id);
    printf("Enter your first and last name: ");
    scanf("%s%s",&one.name.first, &one.name.last);
    printf("ID: %d\n",one.id);
    printf("Name: %s %s\n",one.name.first,
    one.name.last);
    return 0;
}
```

```
or,  
#include <iostream>  
#include<string.h>  
using namespace std;  
  
struct nametype{  
    char first[20];  
    char last[20];  
};  
struct student{  
    int id;  
    struct nametype name;  
};  
int main()  
{  
    struct student one;  
    printf("Enter ID: ");  
    scanf("%d",&one.id);  
    printf("Enter your first and last name: ");  
    scanf("%s%s",one.name.first, one.name.last);  
    printf("ID: %d\n",one.id);  
    printf("Name: %s %s\n",one.name.first,  
    one.name.last);  
    return 0;  
}
```

### **Output:**

Enter ID: 221

Enter your first and last name: Hamim Talukder

ID: 221

Name: Hamim Talukder

- **Predict the output of the following C program.**

```
#include <iostream>
#include<string.h>
using namespace std;
struct nametype{
    char first[20];
    char last[20];
};
struct student_type{
    int id;
    struct nametype name;
};
int main()
{
    printf("Enter student number: ");
    int i, n;
    scanf("%d",&n);
    struct student_type student[n];
    for(i=0;i<n;i++){
        printf("%d no. student\n",i+1);
        printf("Enter ID: ");
        scanf("%d",&student[i].id);
        printf("Enter your first and last name: ");
        scanf("%s%s",student[i].name.first,
        student[i].name.last);
    }
    printf("Output:\n\n");
    for(i=0;i<n;i++){
        printf("%d no. student\n",i+1);
```

```
    printf("ID: %d\n",student[i].id);
    printf("Name: %s %s\n",student[i].name.first,
student[i].name.last);
}
return 0;
}
```

- **What will be the output of the program given below.**

```
#include <iostream>
#include<string.h>
using namespace std;

typedef struct
{
    char first[20];
    char last[20];
} nametype;

typedef struct
{
    int id;
    nametype name;
    char grade[3];
} studenttype;
```

```
void calculate_grade(studenttype *s, int mark)
{
    if (mark >= 80)
    {
        strcpy(s->grade, "A+");
    }
}
```

```
else if (mark >= 70)
{
    strcpy(s->grade, "A");
}
else if (mark >= 60)
{
    strcpy(s->grade, "A-");
}
else if (mark >= 50)
{
    strcpy(s->grade, "B");
}
else if (mark >= 40)
{
    strcpy(s->grade, "C");
}
else
{
    strcpy(s->grade, "F");
}
```

```
int main()
{
    studenttype student[5];
    int i, n = 5;
    int marks[] = {72, 82, 60, 20, 50};

    for (i = 0; i < n; i++)
    {
```

```
printf("Enter the ID for student %d: ", i + 1);
scanf("%d", &student[i].id);
printf("Enter the first name for student %d: ", i
+ 1);
scanf("%s", student[i].name.first);
printf("Enter the last name for student %d: ", i
+ 1);
scanf("%s", student[i].name.last);
strcpy(student[i].grade, "");
printf("\n");
}

for (i = 0; i < n; i++)
{
    calculate_grade(&student[i], marks[i]);
}

printf("Output: \n\n");

for (i = 0; i < n; i++)
{
    printf("ID: %d\n", student[i].id);
    printf("Name: %s %s\n", student[i].name.first,
student[i].name.last);
    printf("Grade: %s\n", student[i].grade);
}

return 0;
}
```

**Output:**

Enter the ID for student 1: 1

Enter the first name for student 1: Hamim

Enter the last name for student 1: Talukder

Enter the ID for student 2: 2

Enter the first name for student 2: Jim

Enter the last name for student 2: islam

Enter the ID for student 3: 3

Enter the first name for student 3: Hridi

Enter the last name for student 3: Chowdhuri

Enter the ID for student 4: 4

Enter the first name for student 4: Mim

Enter the last name for student 4: Islam

Enter the ID for student 5: 5

Enter the first name for student 5: Shuruvi

Enter the last name for student 5: Khatun

**Output:**

ID: 1

Name: Hamim Talukder

Grade: A

ID: 2

Grade: A+

ID: 3

Name: Hridi Chowdhuri

Grade: A-

ID: 4

Name: Mim Islam

Grade: F

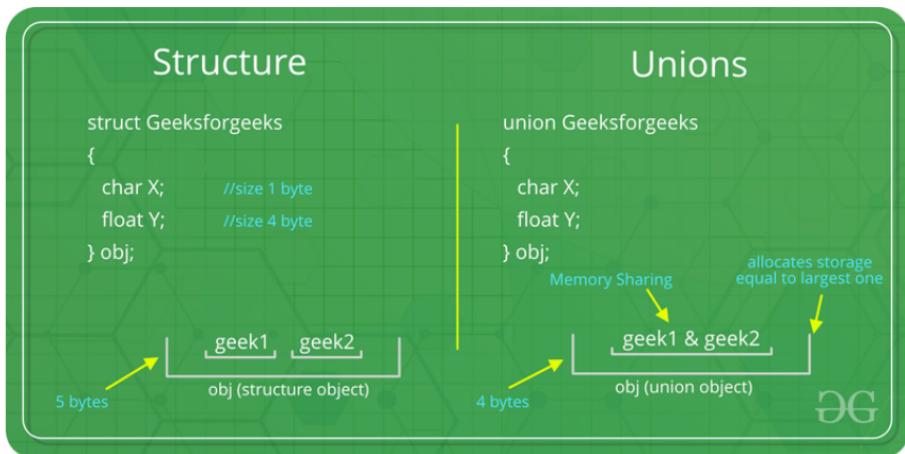
ID: 5

Name: Shuruvi Khatun

Grade: B

## Unions:

Union is a user defined data type but unlike structures, union members share same memory location.



**For example in the following C program, both x and y share the same location. If we change x, we can see the changes being reflected in y:**

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

```
// Declaration of union is same as structures
union test {
    int x, y;
};
```

```
int main()
{
    // A union variable t
    union test t;
```

```
t.x = 2; // t.y also gets value 2
printf("After making x = 2:\n x = %d, y = %d\n\n",
t.x, t.y);

t.y = 10; // t.x is also updated to 10
printf("After making y = 10:\n x = %d, y =
%d\n\n",
t.x, t.y);
return 0;
}
```

### **Output:**

After making x = 2:  
x = 2, y = 2

After making y = 10:  
x = 10, y = 10

### **How is the size of union decided by compiler?**

Size of a union is taken according the size of largest member in union.

```
#include <iostream>
using namespace std;
union test1 {
    int x;
    int y;
} Test1;
```

```
union test2 {
    int x;
    char y;
} Test2;
```

```
union test3 {  
    int arr[10];  
    char y;  
} Test3;  
  
int main()  
{  
    printf("sizeof(test1) = %lu, sizeof(test2) = %lu, "  
        "sizeof(test3) = %lu",  
        sizeof(Test1),  
        sizeof(Test2), sizeof(Test3));  
    return 0;  
}
```

### **Output:**

sizeof(test1) = 4, sizeof(test2) = 4, sizeof(test3) = 40

### **Pointers to unions?**

Like structures, we can have pointers to unions and can access members using the arrow operator (->). The following example demonstrates the same.

```
#include <iostream>  
using namespace std;  
union test {  
    int x;  
    char y;  
};  
int main()  
{  
    union test p1;  
    p1.x = 65;
```

```
// p2 is a pointer to union p1
union test* p2 = &p1;

// Accessing union members using pointer
printf("%d %c", p2->x, p2->y);
return 0;
}
```

### **Output:**

65 A

### **What are applications of union?**

Unions can be useful in many situations where we want to use the same memory for two or more members. For example, suppose we want to implement a binary tree data structure where each leaf node has a double data value, while each internal node has pointers to two children, but no data. If we declare this as:

```
struct NODE {
    struct NODE* left;
    struct NODE* right;
    double data;
};
```

then every node requires 16 bytes, with half the bytes wasted for each type of node. On the other hand, if we declare a node as following, then we can save space.

```
struct NODE {  
    bool is_leaf;  
    union {  
        struct  
        {  
            struct NODE* left;  
            struct NODE* right;  
        } internal;  
        double data;  
    } info;  
};
```

## **5. File**

---

### **File in C :**

In C programming, File is place on disk where a group of related data is stored.

File is a structure which is stored in stdio.h header.

### **Declaration of file:**

```
FILe *name_of_file;
```

In C, you can perform four major operation on the file, either text or binary:

1. Creating a new file,
2. Opening an existing file,
3. Closing a file,
4. Reading from and writing information to a file.

### **To write something in a file:**

```
fputc(),  
fputw(),  
fputs(),  
fprintf(),  
fwrite().
```

### **To read something in a file:**

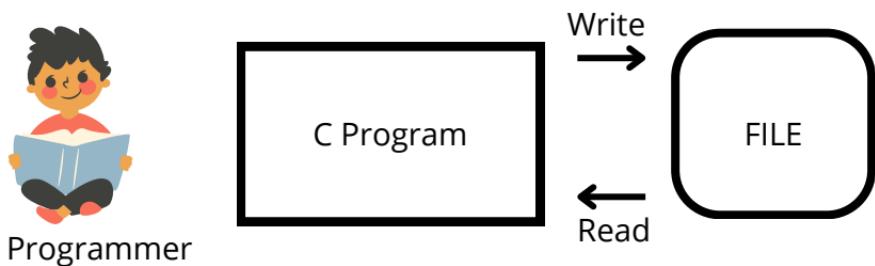
```
fgetc(),  
fgetw(),  
fgets(),  
fscanf(),  
fread().
```

## **File Opening Models:**

<b>Mode</b>	<b>Description</b>
r	Opening an existing text file for reading purpose.
w	Opens a text file for writing. If it does not exist, then a new file is created. Here your program will start writing content from the beginning of the file.
a	Opens a text file for writing in appending mode. If it does not exist , then a new file is created your program will start appending content in the existing file content.
r+	Opens a text file for both reading and writing. If the file exists, loads it loads it into memory and set up a pointer to the first character in it. If file doesn't exist it returns null.
w+	Opens a text file for both reading and writing. If file is present, it first destroys the file to zero length if it exists, otherwise create a file if it does not exist.
a+	Opens a text file for both reading and writing. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.

The Random Access Memory is volatile and its content is lost once the program terminates in order to persist the data former we use files.

A file is data stored in a storage device. A C program can talk to file by reading content from it and writing to it.



### **File pointer:**

The "FILE" is a structure which needs to be created for opening the file.

A file pointer is a pointer to this structure to the file.

FILE pointer is needed for communication between the file and the program.

A FILE pointer can be created as follows:

```
FILE *ptr;
```

```
ptr = fopen ("file_name.txt" , "mode") ;
```

## **FILE opening modes in C:**

C offers the programmers to select a mode for opening a file.

Following modes are primarily used in C FILE I/O :

"r" : Open for reading → If the file does not exist  
fopen returns NULL

"rb" : Open for reading in binary

"w" : Open for writing → If the file exist, the contents will be overwritten

"wb" : Open for writing in binary

"a" : Open for append → If the file does not exist it will be created

## **Types of Files:**

there are two types of files:

- 1) Text files ( .txt , .c)
- 2) Binary files ( .jpg , .dat)

## **Reading a file:**

A file can be opened for reading as follows:

```
FILE *ptr;  
ptr = fopen ("Hamim.txt") ;  
int num;
```

Let us assume that "Hammim.txt" contains an integer we can read that integer using:  
fscanf(ptr , "%D" , &num);

This will read an integer from file in num variable.

Quick Quing: Modify the program above to check whether the file exist or not before opening the file.

### **Closing the file:**

It is very important to close the file after read or write. This is achieved using close as follows:

```
fclose(ptr) ;
```

This will tell the compiler that we are done working with this file and the associated resources called be freed.

### **Write to a file:**

We can write to a file in a very similar manner like we read the file.

```
FILE *fptr ;  
fptr = fopen ("Hamim.txt" , "w") ;  
int num = 432;
```

```
fprintf (fptr , "%d" , num) ;  
fclose (fptr) ;
```

### **fgetc() and fputc() :**

fgetc() and fputc() are used to read and write a character from/to a file.

fgetc(ptr) => Used to read a character from file.

fputc('c' , ptr) ; => Used to write character 'c' to the file.

### **EOF (End of File) :**

fgetc returns EOF when all the characters from a file have been read. So we can write a check like bellow to detect end of file.

```
while (1) {                      => When all the content  
    ch = fgetc (ptr) ;           of a file has been  
    if (ch == EOF) {            read, break the loop!  
        break ;  
    }  
    // code  
}
```

- **Open a file and then close it.**

```
#include <stdio.h>
int main()
{
    FILE *file;
    file = fopen("1test.txt", "w");
    if (file == NULL)
    {
        printf("File doesn't exist\n");
    }
    else
    {
        printf("File is opened\n");
        fclose(file);
    }
    return 0;
}
```

- Write your name in a file using fputc() function.

```
#include <stdio.h>
#include<string.h>
int main()
{
    FILE *file;
    char name[20]={"Hamim Talukdaar"};
    int str_length=strlen(name),i;
    file = fopen("1test.txt", "w");
    if (file == NULL)
    {
        printf("File doesn't exist\n");
    }
    else
    {
        printf("File is opened\n");
        for(i=0;i<str_length;i++){
            fputc(name[i],file);
        }
        printf("File is writtent successfully.\n");
        fclose(file);
    }
    return 0;
}
```

- **Write your name in a file using fputs() function.**

```
#include <stdio.h>
#include<string.h>
int main()
{
    FILE *file;
    char name[50];
    int str_length=strlen(name),i;
    file = fopen("1test.txt", "w");
    if (file == NULL)
    {
        printf("File doesn't exist\n");
    }
    else
    {
        printf("File is opened\n");
        printf("Enter your name: ");
        gets(name);
        fputs(name,file);
        printf("File is written successfully.\n");
        fclose(file);
    }
    return 0;
}
```

- **Make a file program that can take your name and your favorite person name then can keep it in a file using fputs() function.**

```
#include <stdio.h>
#include<string.h>
int main()
{
    FILE *file;
    char name[50];
    int str_length=strlen(name),i;
    file = fopen("1test.txt", "a");
    if (file == NULL)
    {
        printf("File doesn't exist\n");
    }
    else
    {
        printf("File is opened\n");
        printf("Enter your name: ");
        gets(name);
        fputs(name,file);
        fputs("\n",file);
        printf("Enter your favourite person name: ");
        gets(name);
        fputs(name,file);
        fputs("\n",file);
        printf("File is written successfully.\n");
        fclose(file);
    }
    return 0;
}
```

- Write your name in a file using `fprintf()` function.

```
#include <stdio.h>
#include<string.h>
int main()
{
    FILE *file;
    char name[50];
    int str_length=strlen(name),i;
    int age;
    file = fopen("1test.txt", "w");
    if (file == NULL)
    {
        printf("File doesn't exist\n");
    }
    else
    {
        printf("File is opened\n");
        printf("Enter your name: ");
        scanf(" %[^\n]",name);
        printf("Enter your age: ");
        scanf("%d",&age);
        fprintf(file,"Name = %s, Age = %d\n",name,age);
        printf("File is writtent successfully.\n");
        fclose(file);
    }
    return 0;
}
```

- **Reading a file using fgetc() function.**

```
#include <stdio.h>
#include<string.h>
int main()
{
    FILE *file;
    char ch;
    file = fopen("1test.txt", "r");
    if (file == NULL)
    {
        printf("File doesn't exist\n");
    }
    else
    {
        printf("File is opened\n");
        while(!feof(file)) // After end of the file
        {
            ch = fgetc(file);
            printf("%c",ch);
        }
        fclose(file);
    }
    return 0;
}
```

- **Reading a file using fgets() function.**

```
#include <stdio.h>
#include<string.h>
int main()
{
    FILE *file;
    char ch[50];
    file = fopen("1test.txt", "r");
    if (file == NULL)
    {
        printf("File doesn't exist\n");
    }
    else
    {
        printf("File is opened\n");
        while(!feof(file))
        {
            fgets(ch,33, file); // 39 = per line size
            printf("%s\n",ch);
        }
        fclose(file);
    }
    return 0;
}
```

- **Reading a file using fscanf() function.**

```
#include <stdio.h>
#include<string.h>
int main()
{
    FILE *file;
    char ch[50];
    file = fopen("1test.txt", "r");
    if (file == NULL)
    {
        printf("File doesn't exist\n");
    }
    else
    {
        printf("File is opened\n");
        fscanf(file, " %[^\n]",&ch);
        printf("%s\n",ch);
        fclose(file);
    }
    return 0;
}
```

- **Reading a file using fscanf() function.**

```
#include <stdio.h>
#include<string.h>
int main()
{
    FILE *file;
    char first_name[50];
    char last_name[50];
    file = fopen("1test.txt", "r");
    if (file == NULL)
    {
        printf("File doesn't exist\n");
    }
    else
    {
        printf("File is opened\n");
        fscanf(file,"%s %s",&first_name,&last_name);
        printf("%s %s\n",first_name,last_name);
        fclose(file);
    }
    return 0;
}
```

- Open a file and write students name, age and phone number like given bellow

Name	Age	Phone number
------	-----	--------------

.....	.....	.....
-------	-------	-------

Hamim	22	1731767273
-------	----	------------

Jim	18	1731767274
-----	----	------------

Rahim	18	1731767275
-------	----	------------

```
#include <stdio.h>
#include <string.h>
int main()
{
    FILE *file;
    char name[50];
    int age, phone_Number, num, i;
    file = fopen("student.txt", "a");
    if (file == NULL)
    {
        printf("File doesn't exist\n");
    }
    else
    {
        printf("File is opened\n");
        printf("Enter number of students: ");
        scanf("%d",&num);
        for(i=1;i<=num;i++)
        {
```

```
printf("Enter student Name : ");
scanf("%s",&name);
printf("Enter student Age : ");
scanf("%d",&age);
printf("Enter student Phone number : ");
scanf("%d",&phone_Number);

fprintf(file,"\n%s\t\t%d\t%d\n",name,age,phone_
Number);
}

fclose(file);
}
return 0;
}
```

- Write a program to read three integers from a file.
- Write a program to generate multiplication table of a given number in text formate. Make sure that the file is readable and well formatted.
- Write a program to read a text file character by character and write its content twice in a separate file.
- Take name and salary of two employees as input from the user and write them to a text file in the following format:

name1, 3300

name2, 7700

- Write a program to modify a file containing an integer to double its value.

2 => 4

prev. file      new file

- **Make a program that can open a file.**

```
#include<stdio.h>
int main()
{
    FILE *file;
    file = fopen("test.txt","w");
    if(file==NULL)
        printf("File doesn't exist");
    else
        printf("File is opend");
    fclose(file);
    return 0;
}
```

- **Make a program in which we can write something using file.**

```
#include<stdio.h>
#include<string.h>
int main()
{
    FILE *file;
    char name[20]="Hamim Talukder";
    int length = strlen(name), i;
    file = fopen("test.txt","w");
    if(file==NULL)
    {
        printf("File doesn't exist");
    }
    else
```

```
{  
    printf("File is open\n");  
    for(i=0;i<length;i++){  
        fputc(name[i],file);  
    }  
    printf("File is written successfully.\n");  
    fclose(file);  
}  
return 0;  
}
```

## **File in C++ :**

File is used to store data permanently.

- cin and cout method requires <iostream> library.
- To read and write into a file we need <fstream> library.
- ofstream data type used to create and write information to files.
- ifstream data type used to read information from files.

- **how to create and write into a file :**

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

int main()
{
    ofstream file;
    file.open("student.txt");
    file<<"I am Hamim Talukder. I am 28 years old. I am
from Tangail\n";
    file.close();
    return 0;
}
```

```
or,  
#include<iostream>  
#include<fstream>  
#include<string>  
using namespace std;  
  
int main()  
{  
    string name;  
    ofstream file;  
    file.open("student.txt");  
    cout<<"Enter your name : ";  
    getline(cin,name);  
    file<<"Welcome "<<name<<endl;  
    file.close();  
    return 0;  
}  
  
#include<iostream>  
#include<fstream>  
#include<string>  
using namespace std;  
  
int main()  
{  
    string name;  
    ofstream file;  
    file.open("student.txt",ios::out|ios::app);  
    cout<<"Enter your name : ";  
    getline(cin,name);  
    file<<"Welcome "<<name<<endl;
```

```
file.close();
cout<<"Data is stored"<<endl;
return 0;
}
```

- **Storing student details in a file :**

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

int main()
{
    string name;
    int age;
    ofstream file;
    file.open("student_details.txt",ios::out|ios::app);
    cout<<"Enter your name : ";
    getline(cin,name);
    file<<"Welcome "<<name<<endl;
    cout<<"Enter your age : ";
    cin>>age;
    file<<"Age = "<<age<<endl;
    file.close();
    cout<<"Data is stored"<<endl;
    return 0;
}
```

```
or,  
#include<iostream>  
#include<fstream>  
#include<string>  
using namespace std;  
  
int main()  
{  
    string name;  
    int age;  
    ofstream file;  
    file.open("student_details.txt",ios::out | ios::app);  
    for (int i = 0; i <= 3; i++)  
    {  
        cout<<"Enter your name : ";  
        getline(cin,name);  
        file<<"Welcome "<<name<<endl;  
        cout<<"Enter your age : ";  
        cin>>age;  
        file<<"Age = "<<age<<endl;  
        cin.ignore();  
        cout<<"Data is stored"<<endl;  
    }  
    file.close();  
  
    return 0;  
}
```

- **How to read from a file :**

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

int main()
{
    string line;
    ifstream file("student_details.txt");
    while(getline(file,line))
    {
        cout<<line<<endl;
    }
    file.close();

    return 0;
}
```









# C++ LANGUAGE (SECOND PART) T.I.M. HA-MEAM