Dynamic memory Allocation:

Storage class:

-storage class is a concept available in c++.

-storage class defines 3 different property of a variable/functions.

1.scope of the variable.-->in which place i can use this variable.

2.lifetime of a variable.-->when that variable will be deallocated/at what time that variable will be removed from the memory.

3.storage location🡪where i need to store my variable

There are total 5 storage classes are available in c++.

Auto,Extern,static,register,mutable

How many different types of memory we can assign to our variables?

1.static memory allocation---->compile time memory allocation means static memory allocation.

int x=100;

int----->4bytes

float--->4bytes

int arr[5];--->20bytes

struct Emp[5]-🡪140bytes

struct Emp

{

Int eid;--------4

Char[20] ename;----20

Float sal;--------4

};

Why we are calling static memory allocation as a compile time memory allocation?

-We are calling static memory allocation as compile time memory allocation because though the size of the datatype is fixed so that before executing our program we can decide the size of that variable.

-and that size is fixed for that variable.

-when we are dealing with fixed size datatype or fixed memory location at that time we need to go for static memory allocation.

2.Dynamic memory allocation:

-When we are dealing with resizable memory at that time we need to go for dynamic memory allocation.

-The memory which can be increased or decreased depending upon our requirement.

-those kind of memory are also known growable memory.

-Incase of c program for DMA we need to go for stdlib.h

-in stdlib.h we have 4 functions for this DMA

1.malloc()—used to allocate DM for the userdefined datatypes(structures and unions)

2.calloc()—used to allocate memory for the derived datatype like Array

3.realloc()—used to increase/decrease the size of an array

Int arr[3];

arr(x)

0 1 2

|  |  |  |
| --- | --- | --- |
| 10 | 20 | 30 |

Apply realloc() to arr

arr(point)

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | 20 | 30 |  |  |

4.free—used to delete memory

Pointer:

-pointer is nothing but it is a derived datatype.

-whenever you will create a pointer variable, pointer variable is responsible to an address.

-it will hold the address of an variable.

int a=100

int\* ptr=&a;

(address--2089)a---->100

\*ptr—2089

|  |
| --- |
|  |

(address🡪3786)Ptr🡪2089

&ptr=3786

ptr=100(ptr🡪&a🡪100)

control--->ptr--->2089--->100

\*🡪value at symbol

&🡪address of

Malloc()

Prototype

Returntype+ functionName+function argument

void\* malloc(size\_t size)

Void\*--it is a generic pointer

-whether my malloc function will return an memory address of an integer value or an float value or an student value or of any datatype.

-generic pointer can able to hold any type of address.

Size\_t—unsighned(positive integer)—size of the memory we need to pass as an argument of malloc()

And this function always expects an positive integer as a memory size

Structure & pointer

-Structure is nothing it is an userdefiend datatype.

-whenever there is a require that we want to represent the userdefined datas or i can say that when i wnt to represent multiple dissimilar/heterogeneous data elements in single unit at that time we need to go for userdefined datatype concepts like structure or union.

-before using any userdefined datatype in our program we need to declare those datatype first but in case of predefined datatype we do not required to declare before using those datatypes in our program.

-for example suppose i want to use an integer predefined datatype.

int x=10;

here not required to any extra information about what is int because my compiler or my controller already knows what is int and when to use int.

But suppose i want to use an userdefined datatype Employee then before using this Employee datatype we need to create the structure of employee by declaring the employee structure.

struct Employee

{

int eid;

char[30] ename;

float esal;

};

Main()

{

struct Employee e1;

}

Pointer

-it is a predefined datatype, this pointer is coming user derived datatype concept.

-Without the help of primitive datatype i can not think about pointer datatype.

-pointer datatype is always represents a memory address of a particular variable.

-my pointer type is depends upon the variable whose address need to store inside that pointer.

-suppose example my pointer variable will store the address of an integer then it is of type int\*

-suppose my pointer variable will store the address of a float then it will be float\*

-suppose my pointer variable will store the address of an Employee variable then it will be Employee\*

-the pointer datatype we can represent by the help of any primitive/userdefined datatype and by the help of \* symbol.

-Basically in pointer we are using two types of symbol

These are &,\*

&🡪Address of operator

\*🡪value at operator

How this is possible that we can able to use the same \* operator for multi purpose:

-here the programming language internally uses the operator overloading concept.

Int\* x;

Int \*x;

Int y;

\*x=&y;--🡪here the value at x is address of y.

X=10;

Types of memory allocation:

-Basically We can perform two types of memory allocations.

1.static memory allocation

2.Dynamic memory alloctaion.

Operator overloading concept:

+ operator

Addition operation---5+7=12

String concatenation—“hello”+” i am bbsr”=hello i am in bbsr

struct Demo

{

Int x,y;

};

Demo operator +( Demo &obj(d1), Demo &obj1(d2))

{

Struct Demo d;

d.x=obj.x+obj1.x;

d.y=obj.y+obj1.y;

return d;

}

Void print(Demo a(d3))

{

Printf(“a.x” + “a.y”);

}

Main()

{

Struct Demo d1;

d1.x=10;

d1.y=20;

Struct Demo d2

d2.x=12;

d2.y=13;

struct Demo d3;

d3=d1+d2;//x=22,y=33//d

d3=d;

print(d3);

}