cin and cout

- C++ provides an easier way for input and output.
- The output:
  - cout << “Hello C++”;
- The input:
  - cin >> var;

```
#include<stdio.h>
int main()
{
    printf(“ hellow world”);
}
```

```
#include<iostream>
using namespace std;
int main()
{
   cout<<“hellow world”;
}
```

printf is a function & declared in stdio.h header file. So if we want to use printf it is necessary include stdio.h header file

cout is object of ostream class and ostream class is declared in iostream.h that’s why if u want to use cout object is necessary to include iostream.h header file

operator which is used with cout is called as insertion operator <<

```
int res=30;
printf(“res=%d”, res);
```

```
int res=30;
cout<<“res=”<<res;
```

```
a=10,b=5;
printf(“a=%d b=%d”,a,b);
```

```
int a=10, b=5;
cout<<"a="<<a<<" b="<<b;
```

```
#include<stdio.h>
int main()
{
    int num;
    scanf("%d",&num);
}
```

```
#include<iostream>
using namespace std;
int main()
{   int num;
    cin>>num;
}
```

scanf is a function & declared in stdio.h header file. So if we want to use scanf it is necessary include stdio.h header file

cin is object of istream class and istream class is declared in iostream.hthat's why if u want to use cin object is necessary to include iostream.h header file

operator which is used with cin is called as operator extraction (>>)

```
int  no1, no2;
scanf("%d%d",&no1, &no2);
```

```
int no1, no2;
cin>>no1>> no2;
```

- **Data members**
- Data members of the class are generally made as private to provide the data security.
- The private members cannot be accessed outside the class.
- So these members are always accessed by the member functions.

Constructor

- We can have constructors with
  - No argument : initialize data member to default values
  - One or more arguments : initialize data member to values passed to it
  - Argument of type of object : initialize object by using the values of the data members of the passed object. It is called as copy constructor.

Constructor

- Constructor is a member function of class having same name as that of class and don’t have any return type.
- Constructor get automatically called when object is created i.e. memory is allocated to object.
- If we don’t write any constructor, compiler provides a default constructor.

- Destructor

- Destructor is a member function of class having same name as that of class preceded with ~ sign and don’t have any return type and arguments.
- Destructor get automatically called when object is going to destroy i.e. memory is to be de-allocated.

- Destructor
- If we don’t write, compiler provides default destructor.
- Generally, we implement destructor when constructor of the object is allocating any resource
- e.g. we have pointer as data member, which is pointing to dynamically allocated memory.

- Size of object of empty class is always 1 byte
- When you create object of an class it gets 3 characteristics
  - State
  - Behavior
  - Identity
- When you create object of empty class at that time state of object is nothing. Behavior of that object is also nothing.

  but that object have unique identity(address).

memory in computer is always organized in form of bytes.

Byte is unit of memory. Minimum memory at objects unique address is one byte that's why size of empty class object is one byte.

References

- References are treated as aliases to the variable.
- It can be used as another name for the same variable
- int a=10; int &r = a;
- Thus we can pass arguments to function, by value, by address or by reference.
- Reference is internally treated as constant pointer to the variable, which gets automatically de-referenced.

```
#include<iostream>
using namespace std;
void swap(int &n1, int &n2)
{
     int temp;
     temp=n1;
     n1=n2;
     n2=temp;
    cout<< "\n &n1 ::"<< &n1<<" &n2 ::" << &n2<<endl;
}
void main()
{
   int no1=5, no2=10;
   cout<< "\n no1 ::"<< no1<<" no2 ::" << no2<<endl;
   cout<< "\n &no1::"<< &no2<<"no2::" << &no2<<endl;
   swap(no1, no2);
   cout<< "\n no1 ::"<< no1<<" no2 ::" <<no2<<endl;
   cout<< "\n&no1 ::"<<&no1<<"&no2 ::"<<&no2<<endl;
}
```

Difference between pointer and reference:

	Pointers	References
1	**pointers may not be (not compulsory) initialized at the point of declaration.**	**reference must be initialized at the point of declaration.**
2	**pointers must be derefeneced explicitly using value at (*) operator.**	**References are automatically dereference.**
3	**address stored in a pointer can be modified later.**	**reference keeps referring to the same variable till it goes out of scope.**
4	**we can have pointer arithmetic, null pointers, dangling pointers.**	**such concepts do not exist for references.**
5	**We can initialized pointer to NULL.**	**We can not initialized reference to NULL**
6	**We can create a array of pointer.**	**We can not create a array of reference.**
7	**We can create pointer to pointer**	**Can not create reference to reference.**

Exception Handling:
runtime error which can be handled by programmer is called exception.
In c++ exceptions are handled using try, catch and throw.

```
#include<iostream>
using namespace std;
int main()
{    int x=10,y=0;
     try
     {    if(y==0)
          {    throw 1;
          }
          else
          {              int res=x/y;
               cout<<"res::"<<res<<endl;
          }
     }
     catch(int)
     {    cout<<"Enter Other Than 0"<<endl;
     }
     return  0;
}
```

Try block must have at least one catch handler.
One try block can have multiple catch blocks .
When exception is thrown but matching catch block is not available at that time compiler give call to library defined function terminate which internally give call to abort function.
Catch block which handles all kind of exceptions such type of catch is called generic catch block.

```
catch(...)  (ellipse)
{
        cout<<“inside genric block”<<endl;
}
```

catch handler for ellipse must last handler in exception handling

Dynamic memory allocation

C++ provides operators *new* and *delete* for allocating and de-allocating memory at run-time. The memory is allocated on heap.

- int *ptr=NULL; ptr=new int;
  delete ptr; ptr=NULL;
- char *str=NULL; str = new char[10];
  delete[] str; str=NULL:

1. Allocating memory dynamically for a single variable by using new operator

```
int main()
{
        int *ptr=new int;
        *ptr=45;  // cin>>*ptr;
        cout<<ptr<<endl;
        delete ptr;
        ptr=NULL;
        return 0;
}
```

Allocate Memeory for array using New Operator

```
#include<iostream.h>
void main()
{
    int no;
    cout<< "Enter How many Number You want ::";
    cin>> no;
    int *ptr= new int[no];
    for(int cnt=0;cnt<no;cnt++)
    {
        cout<<"Enter number :: ";
        cin>>ptr[cnt];
    }
    for( cnt=0;cnt<no;cnt++)
    {
        cout<<" "<<ptr[cnt] << endl;
    }
    delete [] ptr;
    ptr=NULL;
    cout<<"memory freed"<<endl;
}
```

Malloc

- sizeof operator is required
- When we deallocate memory by using free function which is allocated for an object by using malloc function at that time implicitly destructor will not be called .

New

- sizeof operator is not required
- When we deallocate memory by using delete operator which is allocated for an object by using new operator at that time implicitly destructor of that class will be called.

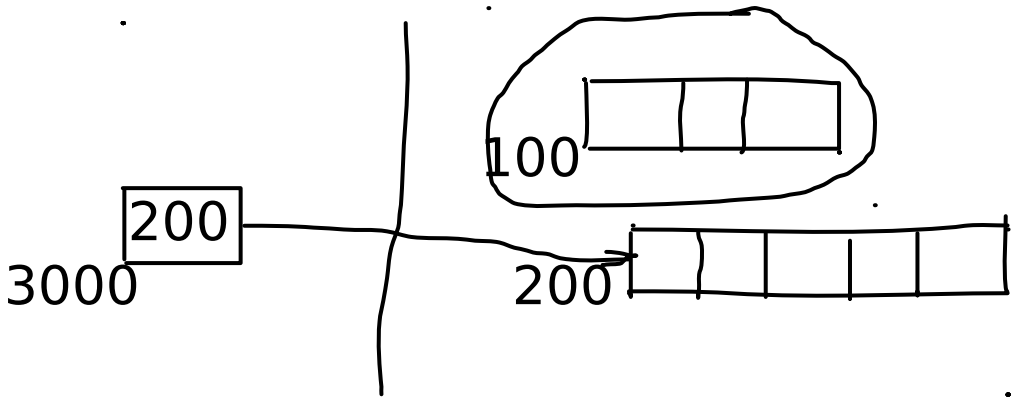
Difference between malloc and new :

	malloc	**new**
1	malloc is a function	new is a operator
2	Allocate memory using malloc constructor is not called i.e.. malloc is not aware of ctor	Allocate memory using new constructor is called i.e.. new is aware of ctor
3	If malloc fails return NULL	if new fails it throws bad_alloc exception
4	Need to specify number of bytes and typecasting is required	Need to specify number of objects and typecasting is not required

Memory Leakage

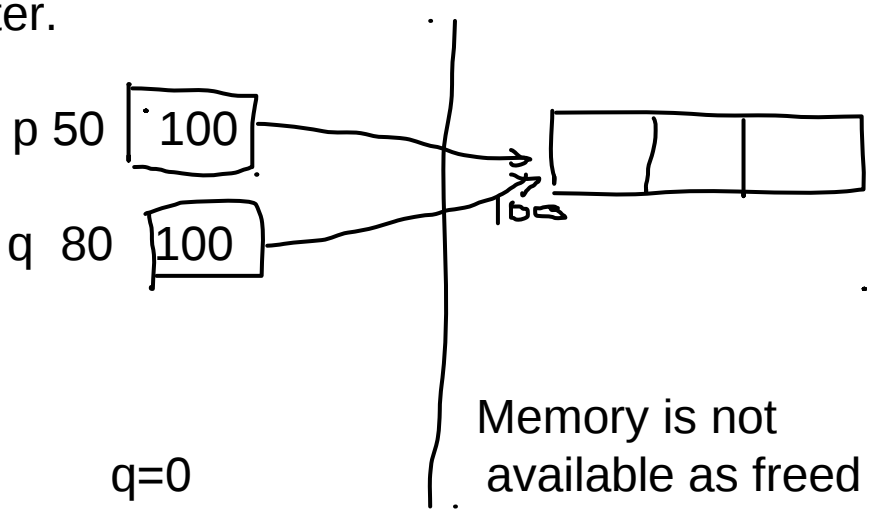
Memory is allocated but not freed after it's used. Then there is no way to reach at that memory such type of wastage of memory is called memory leakage.

```
int *ptr= new int [3];
p=new int [5];
```



Dangling Pointer : pointer pointing to memory which is not available such type of pointer is called as dangling pointer.

```
int *p=new int [3];
int  *q=p;
```




```
delete[] q;
q=NULL;
p becomes dangling pointer
```

Memory is not available as freed it by dele