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;

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

int main()
{
        int no, cnt;
        cout<< "Enter How many Number You want ::";
        cin>> no;
        int *ptr= new int[no];
        for( 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;
}
```

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

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.

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.

Swap using call by reference

```
#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:

run time error which can be handled by programmer is called exception.

In c++ exceptions are handled using try, catch and throw.

```
#include<iostream>
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

Copy Constructor:

it is a special member function of a class which is having same of that class and which gets called implicitly.

1. **when we pass an object to the function by value.**
2. **when we return an object from function by value.**
3. **when we assign already created object to the newly created object (object initialization)**

such special member function is called copy constructor of that class. Job of copy constructor is to create new object from existing object.

When class do not contain copy constructor at that time compiler provides one copy constructor for that class by default such copy constructor is called default copy constructor. It is a special member function of a class having same name of a class and which is taking only one argument of same type but as a reference.

```
Complex (const Complex& other) {
        this->real=other._real;
        this->imag=other.imag;
}
int main() {
    Complex  c1(10, 20);
    Complex  c2=c1; // in this case copy constructor will call for object c2;
}
```

Why c++ is not pure object oriented programming language?

Or why it is partial object oriented programming language ?

1. in c++ we can access private data members of class outside that class by using friend function.

2 . in c++ we can access private data members of class outside that class by using pointers.

3.in c++ we can write global functions and main itself a global function

4. in pure object oriented programming lang data types are also having a classes. Such type of class is called wrapper class. Such type of concept is not available in c++ that’s why c++ is not pure object oriented programming language.

Friend Function

it is a non member function of a class which can access private members of class in which it is declared as friend.

Friend function do not have this pointer.

We can declared global function as friend.

```
#include<iostream>
using namespace std;
class Test
{       private:
                        int a;
                        int b;
        public :
                        Test();
                        Test(int a, int b);
                        friend void sum();
};
```

```
Test::Test()
{
		this->a =0;
		this->b =0;
}
Test::Test(int a, int b)
{
		this->a = a;
		this->b = b;
}
void sum()
{
		Test t(10, 20);
		int ans=t.a +t.b;
		cout<<"ans="<<ans<<endl;
}
	int main()
{
		sum();
		return 0;
}
```

The global functions cannot access private data members of the class.
If such function is made as *friend* of the class, then it can access private members of the class.
If class is made as friend of other class, then all functions of friend class can access private members of that class.

Operator overloading:
giving extension to the meaning of operator is called operator overloading.
By using operator overloading we can change the meaning of operator but we should not change the meaning of operator.

```
 class Complex
{
        int real;
        int imag;
    public:
        Complex();
        Complex(int real , int imag);
        Complex Sum(Complex &other1);
        Complex operator+(complex &other1);
        void Output();
};
```

```
Complex Complex::Sum(complex &other1)
{
        Complex temp;
        temp.real =this->real + other1.real ;
        temp.imag =this->imag + other1.imag ;
        return temp;
}
Complex Complex::operator+(Complex other1)
{
        Complex temp;
        temp.real =this->real + other1.real ;
        temp.imag =this->imag + other1.imag ;
        return temp;
}
int main()
{
        Complex c1(10, 20), c2(20, 10);
        Complex c3=c1.Sum (c2);
        c3.Output ();
        Complex c4=c1+c2;
        c4.Output ();
        return 0;
}
```

When we write like c3=c1+c2 at that time compiler resolves call for this statement like c3=c1.operator+(c2); meaning of this statement is that we have to write function by name of ‘ operator+’. Since this function has called by using object name it must be inside class. Operator+ function taking one argument means at the time of function definition it is necessary to pass argument to that function

function definition for statement c3=c1+c2 is

```
TComplex TComplex::operator+(TComplex  &other1)
{       TComplex temp;
                temp._real =this->_real + other1._real ;
                temp._imag =this->_imag + other1._imag ;
                return temp;
}
 c3=c1 – c2; // c3=c1.operator-(c2);
 c3=c1 * c2; // c3=c1.operator*(c2);
 c3=c1 / c2; // c3=c1.operator/(c2);
```

We can overload operator function by using member as well as friend function friend function.
When we write c3=c1+c2; and operator+ function is member function at that time function call will be resolved like c3= c1.operator+(c2)

when we overload operator+ function by friend function implicitly call will be resolved like c3= operator+(c1, c2):
when we overload operator- function by friend function implicitly call will be resolved like c3= operator-(c1, c2):

```
Complex operator-(Complex &other1, Complex &other1)
{
                Complex temp;
                temp.real= other1.real – other2.real;
                temp.imag= other1.imag – other2.imag;
                return temp;
}
```

we can not overload operator function as member function as well as friend function at same time. In this case compiler will confuse and it will give you ambiguity error.

List of function that complier provides by default for any class if it is not available in that class

1. **default parameter less constructor**
2. **default destructor**
3. **default copy constructor**
4. **default assignment operator function**

Limitations of operator Overloading:

in c++ there are some operator that we can not overload using member function as well as friend function.

1. **. (dot) member selection operator**
2. **:: scope resolution operator**
3. **? : conditional operator**
4. **sizeof size of operator**
5. **.* pointer to member selection**
6. **typeid**
7. **Casting Operators (4 types of casting operators)**

There are some operators we can not overload then as friend function

1 **= assignement**
2 **[] subscript or index**
3 **() function call**
4 **-> arraow operator**