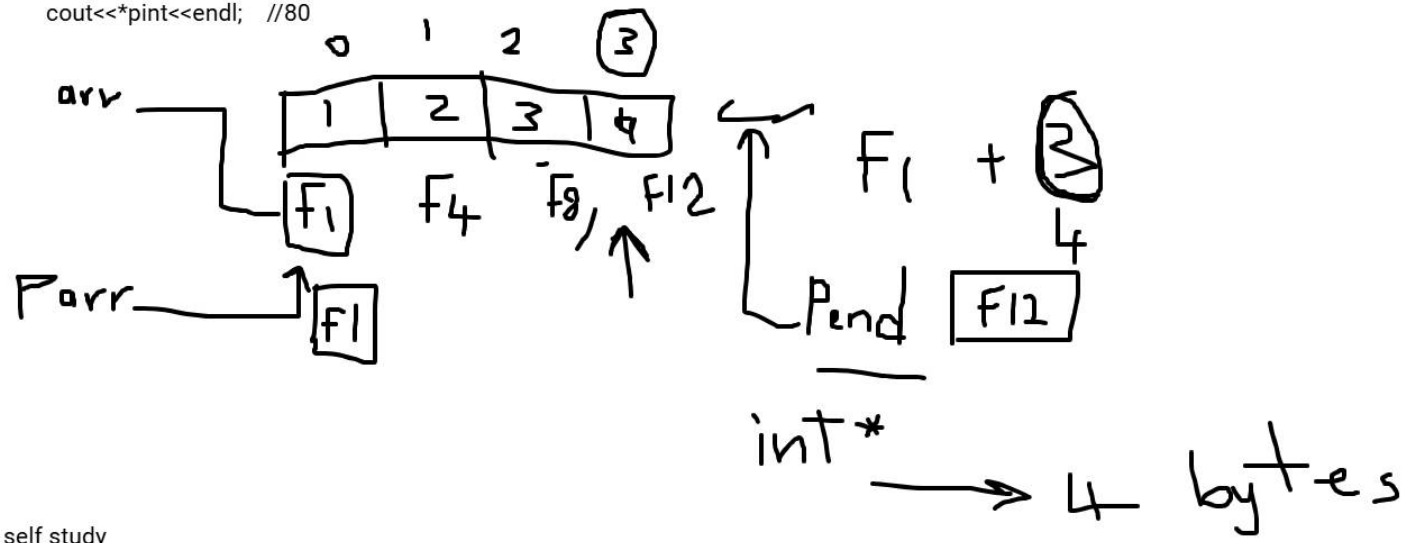
pointer is a variable store memory address of another variable
declared by *

```
int z=99;
int *pint=&z;    //
int x=7;

pint=&x;

pint=nullptr; // seting to null

*pint=10;
cout<<x<<endl;   //x is 10
cin>>*pint;   //user enters 80
cout<<*pint<<endl;    //80
```

```
char ch='a';
char *pch=&ch;  //always an integer because store
memory address

Employee * pemp;//refer to data of type struct emp

cout<<sizeof(pch)<<"  "<<sizeof (pemp)<<endl; //4
bytes

cout<<sizeof(Employee);//# of bytes reserved for
employee struct
```



int* → 4 bytes

self study

table of percedence & associativity

show different cases of pointers with operators →

```
//  sum array elemebts using pointers
 int sum=0;
 for(int i=0,*pcurr=arr;i<4;i++)//,pcurr++
 {
   cout<<"Element"<<i+1<<":"; //<<*pcurr
   cin>>*pcurr;
   sum+=*pcurr++;//++*pcurr//(*pcurr)++; show difference in run
time//
   //pcurr++;
 }
 for(int i=0,*pcurr=arr;i<4;i++,pcurr++)//,pcurr++
 {
   cout<<"Element"<<i+1<<":"<<*pcurr<<endl; //
   //pcurr++;
 }
```

pass by reference , pass by address

reference is not a pointer another name for variable outside its scope

```
void swap( int & a , int&  b)
{
    int temp=a;
    a=b;
    b=temp;
}
//call from main
x=5,y=7;
swap(x,y);
```

reference type can't handle nullability

```
void swap( int *px ,int *py)//pointers
{
    int temp;
    temp=*pa;
    *pa=*pb;
     *pb=temp;
}

//call
x=5,y=7

swap(&x,&y); //passing by address
swap(NULL,NULL);
```

Dynamic Memory :

//memory reserved at runtime not compile time
//reserving Heap memory(shared memory)
//reserve memory --->you should release memory at end of program or end of certain block of code

new ,delete Keyword used to create or delete memory at run time

```
//create array at run time using raw pointers
int size;
cin>>size;
int* parr=new int(size);//i shouldn't leave start of this area..
//parr++;//leave start position
int *pcurrent=parr;//pcurrent change location

for(int i=0;i<size;i++,pcurrent++)
    cin>>*pcurrent;
int sum=0;
for(int i=0,pcurrent=parr;i<size;i++,pcurrent++)
    sum+=*pcurrent;

cout<<sum;

delete parr;//free memory i can remove [ ] primitive type
delete pcurrent;

parr=nullptr;    //avoid dangling pointer (using ptr after delete)

//cout<<*parr;//undefined behavior
```

Unique_ptr (exclusive ownership)
 1- only one pointer can own resource
 2-Automatic memory Mangment (free ,null,...
 3-can't be copied but we can move it to another pointer if needed by using std::move()

```
//Modern Pointers
unique_ptr<int> p=make_unique<int>(40);
cout<<"value :"<<*p<<endl;
//int *pp=p;
//unique_ptr<int> pp=p;//exclusive ownership can't copy its data
unique_ptr<int> pp=move(p);
//p pointer after move p=nullptr
if(!p)
   cout<<"pointer p no longer owns integer above"<<endl;
cout<<"New pointer pp now own integer exclusively:"<<*pp<<endl;

//unique_ptr<int>
int size;
cout<<"enter size :";cin>>size;
auto arr=make_unique<int[]>(size);
for(int i=0;i<size;i++)
   cin>>arr[i];
for(int i=0;i<size;i++)
   cout<<"Element "<<i+1<<"is "<<arr[i]<<endl;
   return 0;    }//automatic free memory
```

 Shared Ownership
 1-multiple pointers share ownership of the same object
 2-object is destroyed from heap when the last shared pointer goes out of scope
 3-usecount function used to maintains internal reference counter of object

```
 shared_ptr<int> p1=make_shared<int>(50);
 auto p2=p1;
 auto p3=p2;

 cout<<"Value from p1:"<<*p1<<endl;
 *p2=8;
 cout<<"Value from p1:"<<*p1<<endl;
 cout<<"Use count:  "<<p1.use_count()<<endl;
 cout<<"Value from p3:"<<*p3<<endl;
    return 0;

   Student *pstd=new Student;

   cout<<"Raw pointer :"<<pstd->name<<endl;    //(*pstd).name
```

Dealing pointers with array of struct dynamic

```
int n;
cin>>n;

Student *stdArr=new Student[n];
//Student *stdcurr=stdArr;
for(int i=0;i<n;i++)
{
   //cin>>stdcurr->name;
   //cin>>stdcurr->age;//(*stdcurr).age

   cin>>(stdArr+i)->name;
   cin>>(stdArr+i)->age;
   //stdcurr++;
}
//stdcurr=stdArr;
for(int i=0;i<n;i++)
{
   cout<<stdArr[i].name;//cout<<(stdArr+i)->name<<endl;//stdcurr->name<<endl;
   cout<<(stdArr+i)->age<<endl;//stdcurr->age<<endl;
   //stdcurr++;
}
```

array ->class static allocation

vector ->class Dynamic allocation

```
#include<vector>
int main()
{

cout<<"----------"<<endl;

   vector<int> numbers={10,20,30};
   int num;
   do {
          cin>>num;
          numbers.push_back(num);
     }while(num>0);
   numbers.pop_back();//remove -value

   for(int num:numbers)
       //if(num>=0)
          cout<<num<<endl;
```