Dynamic allocation of memory

Pointer: Pointer is an address of a memory location. A variable, which holds an address of a memory location, is known as a Pointer Variable (or simply pointer).

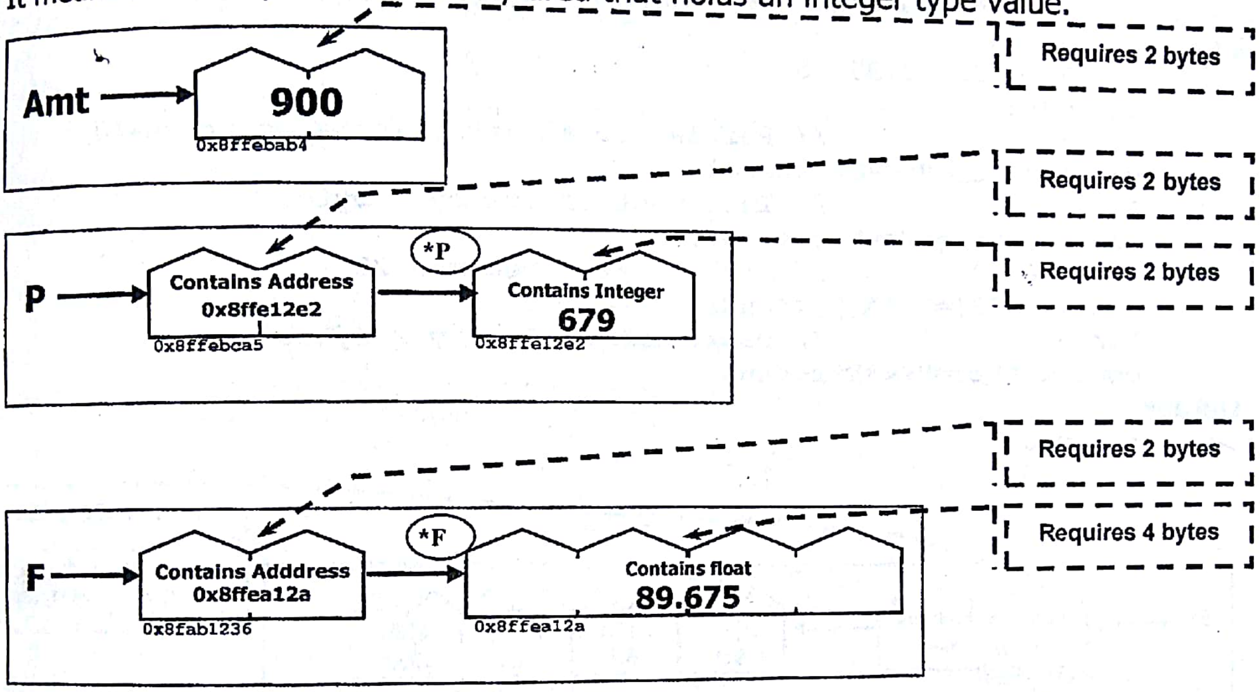
Declaration of a pointer variable

```
int   *P;    //Pointer to an integer
float *F;    //Pointer to a float
char  *Ch;   //Pointer to a character
```

When a simple variable is declared as

```
int Amt=900;
```

It means Amt is a place in memory area that holds an integer type value.



The reference operator **&** returns an address of a memory location of a variable to which it is applied.

```
int Amt=900;
int *Ptr;              //Ptr points to int
Ptr=&Amt;              //Ptr holds the address of Amt
Amt+=100;
(*Ptr)-=50;
cout<<"Amt="<<Amt<<" *Ptr="<<*Ptr<<endl;
cout<<"&Amt="<<&Amt<<" Ptr="<<Ptr<<endl;
```

Above program will display the following output as Ptr holds an address of Amt and hence any change in Amt will be same as change in *Ptr.

smalln

```
Amt=950 *Ptr=950
&Amt=0x8ffebab4 Ptr=0x8ffebab4
```

Using new operator

new operator in C++ returns the address of a block of unallocated bytes (depending on data type a pointer pointing to).

```
float *F,*G;           //F and G point to float
F=new float;           //Allocates storage for 1 float
*F=89.675;             //Assigns a float value to *F
G=F;                   //G shares the same address as F
cout<<"*F="<<*F<<"*G="<<*G<<" F="<<F<<" G="<<G<<endl;
```

Above program on execution will display the following output as F and G are sharing the same address and so the content.

```
*F=89.675 *G=89.675 F=0x8ffea12a G=0x8ffea12a
```

EAll addresses shown above are hypothetical

Using delete operator

delete operator in C++ reverses the process of new operator, by releasing the memory location from a pointer. (It de-allocates the address allocated by new)

```
float *F;          //F points to float
F=new float;       //Allocates storage for one float
*F=89.675;         //Assigns a float value to *F
:
delete F;          //De-allocates address from F
```

Pointer to an array

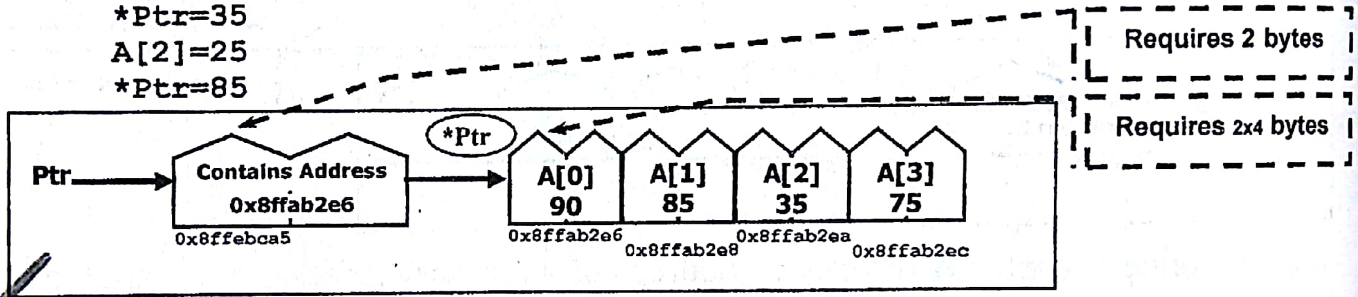
A pointer, which stores an address of an array, is known as pointer to an array.

Example

```
int A[]={90,85,35,75};
int *Ptr;
Ptr=A;             // Pointer to an Array (same as Ptr=&A[0])
cout<<"*Ptr="<<*Ptr<<endl;
Ptr+=2;            // Increment of Ptr by 4 bytes
cout<<"*Ptr="<<*Ptr<<endl;
(*Ptr)-=10;        // A[2] or *Ptr becomes 25
cout<<"A[2]="<<A[2]<<endl;
Ptr--;             // Decrement of Ptr by 2 bytes
cout<<"*Ptr="<<*Ptr<<endl;
```

Output

```
*Ptr=90
*Ptr=35
A[2]=25
*Ptr=85
```



Array of pointers

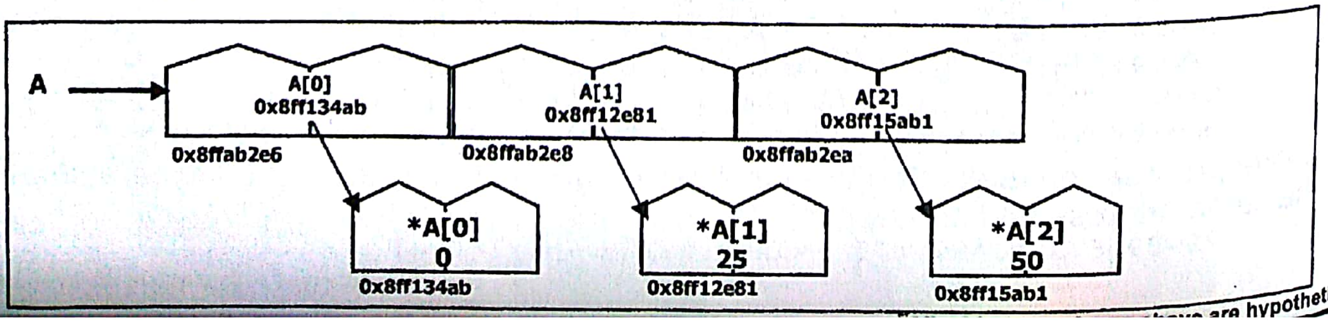
An array, whose each element is pointer type, is known as Array of pointers.

Example

```
int *A[3];
for (int I=0;I<3;I++)
{
  A[I]=new int;
  *A[I]=I*25;
}
for (I=2;I>=0;I--) cout<<"*A["<<I<<"]="<<*A[I]<<endl;
:
for (I=0;I<3;I++) delete A[I];
```

Output

```
*A[2]=50
*A[1]=25
*A[0]=0
```



above are hypothetical

Pointer to character

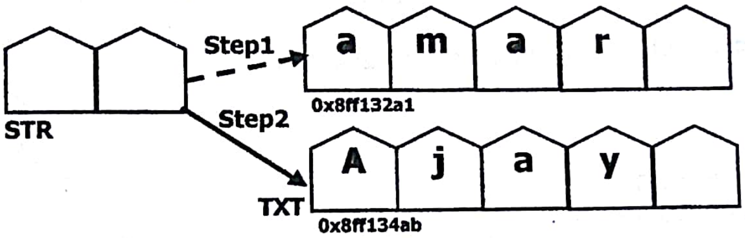
Pointer to character is a special pointer.

Example

```
char *STR="amar"; //Pointer to character initialization [step1]
char TXT[]="Ajay";//Array of character
cout<<STR<<endl;  //amar
cout<<TXT<<endl;  //Ajay
STR=TXT; //STR will point to the address of TXT array   [step2]
cout<<STR<<endl;  //Ajay
cout<<*STR<<endl; //A
while (*STR!='\0')
{
  cout<<*STR<<":"<<STR<<endl;
  STR++;
}
/* Output of the code in while loop
A:Ajay
j:jay
a:ay
y:y
*/
```

*STR is the loc^n to which the ptr is pointing (only 1 char)

STR is the complete string from that loc^n onwards where the ptr is pointing



Pointer to structure

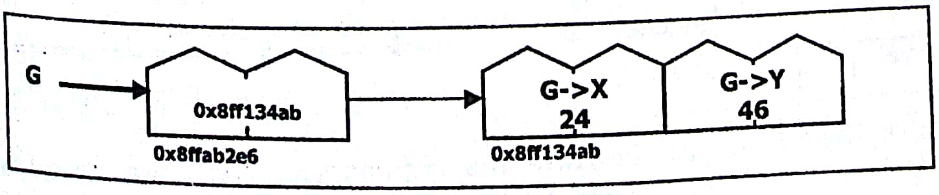
A pointer, which stores the address of struct type data, is known as Pointer to structure.

Example

```
struct Graph
{
  int X,Y;
};
void main()
{
  Graph *G;        //Pointer to structure Graph
  G=new Graph;     //allocates storage for 1 'graph' variable
  //*G.X=24;Not Allowed
  //G.*X=24;Not Allowed
  //G.X=24; Not Allowed
  G->X=24;         //-> is deference operator
  G->Y=G->X*2-2;
  cout<<"G->X="<<G->X<<" G->Y="<<G->Y<<endl;
  delete G;
}
```

Output

```
G->X=24 G->Y=46
```



Dynamic Stack

```
struct NODE
{
 int Data; NODE *Next;
};
class Stack
{
  NODE *Top;
public:
  Stack(){Top=NULL;}
  void Push();
  void Pop();
  void Disp();
  ~Stack();
};
```

Link

```
void Stack::Push()
{
  NODE *Temp;
  Temp=new NODE;
  cout<<"Data:";
  cin>>Temp->Data;
  Temp->Next=Top;
  Top=Temp;
}
```

```
NODE*T=newNODE;
cout<<"enter value"<<endl
cin>> T->Data;
T->Link=Top;
Top=T;
```

```
void Stack::Pop()
{
  if (Top!=NULL)
  {
    NODE *Temp=Top;
    cout<<Top->Data<<"Deleted.."<<endl;
    Top=Top->Next;
    delete Temp;
  }
  else
    cout<<"Stack Empty.."<<endl;
}
```

```
NODE * T=Top;
Top=Too ->Link
Cout<<T->Data<<'deleted'<<endl;
delete T;
```

```
void Stack::Disp()
{
  NODE *Temp=Top;
  while(Temp!=NULL)
  {
    cout<<Temp->Data<<endl;
    Temp=Temp->Next;
  }
}
```

```
NODE*T= TOP;
while (T!=NULL)
{ Cout<< T->Data<<endl;
 T=T->Link;
}.
```

```
Stack::~Stack() //Destructor Function
{
  while (Top!=NULL)
  {
    NODE *Temp=Top;
    Top=Top->Next;
    delete Temp;
  }
}
```

```
while(Top!=NULL)
{ NODE*T=Top;
 Top=Top->Link;
 delete T;
}
```

```
void main()
{
  Stack ST; char Ch;
  do
  {
    cout<<"P/O/D/Q";cin>>Ch;
    switch (Ch)
    {
      case 'P':ST.Push();break;
      case 'O':ST.Pop();break;
      case 'D':ST.Disp();
    }
  }
  while (Ch!='Q');
} // Destructor function will be called
  // automatically when the scope of the
  // object gets over
```

Dynamic Queue

```
struct NODE
{
  int Data;  NODE *Next;
};
class Queue
{
  NODE *Rear,*Front;
public:
  Queue(){Rear=NULL;Front=NULL;}
  void Qinsert();
  void Qdelete();
  void Qdisplay();
  ~Queue();
};
void Queue::Qinsert()
{
  NODE *Temp;
  Temp=new NODE;
  cout<<"Data:";
  cin>>Temp->Data;
  Temp->Next=NULL;
  if (Rear==NULL)
  {
    Rear=Temp;
    Front=Temp;
  }
  else
  {
    Rear->Next=Temp;
    Rear=Temp;
  }
}
void Queue::Qdelete()
{
  if (Front!=NULL)
  {
    NODE *Temp=Front;
    cout<<Front->Data<<"Deleted.."<<endl;
    Front=Front->Next;
    delete Temp;
    if (Front==NULL) Rear=NULL;
  }
  else
    cout<<"Queue Empty.."<<endl;
}
```

```
Node* t=Front;
Front = Front-> link;
cout << T -> Data<<"de
delete T;
```

```
void Queue::Qdisplay()
{
  NODE *Temp=Front;
  while(Temp!=NULL)
  {
    cout<<Temp->Data<<endl;
    Temp=Temp->Next;
  }
}
Queue::~Queue()//Destructor Function
{
  while (Front!=NULL)
  {
    NODE *Temp=Front;
    Front=Front->Next; delete Temp;
  }
}
void main()
{
  Queue QU; char Ch;
  do
  {
    ...
  }while (Ch!='Q');
}
```