Pointers

The Address-of Operator &

```
#include <iostream>
using namespace std;
int main()
{
int var1 = 11;
int var2 = 22;
int var3 = 33;
cout << &var1 << endl                  //print the addresses
       << &var2 << endl                  //of these variables
       << &var3 << endl;
return 0;
}
```

Pointer Variables

Pointer: A variable that holds an address value is called a pointer variable, or simply a pointer.

```
#include <iostream>
using namespace std;
int main()
{
int var1 = 11;                      //two integer variables
int var2 = 22;
cout << &var1 << endl                //print addresses of variables
      << &var2 << endl;
int* ptr;                             //pointer to integers
ptr = &var1;                         //pointer points to var1
cout << ptr << endl;                  //print pointer value
ptr = &var2;                        //pointer points to var2
cout << ptr << endl;                 //print pointer value
return 0;
}
```

Accessing the Variable Pointed To

```
#include <iostream>
using namespace std;
int main()
{
int var1 = 11;                          //two integer variables
int var2 = 22;
int* ptr;                                  //pointer to integers
ptr = &var1;                               //pointer points to var1
cout << *ptr << endl;        //print contents of pointer (11)
ptr = &var2;                                //pointer points to var2
cout << *ptr << endl;         //print contents of pointer (22)
return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
int var1, var2;            //two integer variables
int* ptr;                    //pointer to integers
ptr = &var1;           //set pointer to address of var1
*ptr = 37;                  //same as var1=37
var2 = *ptr;                //same as var2=var1
cout << var2 << endl;          //verify var2 is 37
return 0;
}
```

Pointer to void

```
#include <iostream>
using namespace std;
int main()
{
int intvar;                              //integer variable
float flovar;                              //float variable
int* ptrint;                             //define pointer to int
float* ptrflo;                          //define pointer to float
void* ptrvoid;                            //define pointer to void
ptrint = &intvar;                           //ok, int* to int*
// ptrint = &flovar;                           //error, float* to int*
// ptrflo = &intvar;                         //error, int* to float*
ptrflo = &flovar;                         //ok, float* to float*
ptrvoid = &intvar;                          //ok, int* to void*
ptrvoid = &flovar;                        //ok, float* to void*
return 0;
}
```

Applications of pointers

- Accessing array elements
- Passing arguments to a function when the function needs to modify the original argument
- Passing arrays and strings to functions
- Obtaining memory from the system
- Creating data structures such as linked lists

Pointer Arithmetic

- We can not add, multiply or divide two addresses (Subtraction is possible).
- We can not multiply an integer to an address and similarly we can not divide an address with an integer value.
- We can add or subtract an integer to/from an address.

  (pointer+n=pointer+sizeof(type of pointer)*n)
- We can subtract two addresses of same type.

  (pointer1-pointer2=literal subtraction/sizeof(type of pointer))

Pointers and Arrays

```
#include <iostream>
using namespace std;
int main()
{
int intarray[5] = { 31, 54, 77, 52, 93 };
for(int j=0; j<5; j++)
    cout << intarray[j] << endl;
return 0;
}
```

Pointers and Arrays

```
#include <iostream>
using namespace std;
int main()
{
int intarray[5] = { 31, 54, 77, 52, 93 };
for(int j=0; j<5; j++)
    cout << *(intarray+j) << endl;
return 0;
}
```

Pointers and Arrays

```
#include <iostream>
using namespace std;
int main()
{
int intarray[5] = { 31, 54, 77, 52, 93 };
for(int j=0; j<5; j++)
    cout << *(intarray++) << endl;      // not valid
return 0;
}
```

Pointer Constants and Pointer Variables

```
#include <iostream>
using namespace std;
int main()
{
int intarray[] = { 31, 54, 77, 52, 93 };
int *ptrint;
ptrint=intarray;
for(int j=0; j<5; j++)
    cout << *(ptrint++) << endl;
return 0;
}
```

Pointers and Functions

- Passing Simple Variables

```
#include <iostream>
using namespace std;
int main()
{
void centimize(double&);
double var = 10.0;
cout << “var = “ << var << “ inches” << endl;
centimize(var);
cout << “var = “ << var << “ centimeters” << endl;
return 0;
}
void centimize(double& v)
{
v *= 2.54;
}
```

Call by Address

```
#include <iostream>
using namespace std;
int main()
{
void centimize(double*);
double var = 10.0;
cout << “var = “ << var << “ inches” << endl;
centimize(&var);
cout << “var = “ << var << “ centimeters” << endl;
return 0;
}
void centimize(double* ptr)
{
*ptr *= 2.54;
}
```

Passing Arrays

```
const int MAX = 5;                              //number of array elements
int main()
{
void centimize(double*);
double varray[MAX] = { 10.0, 43.1, 95.9, 59.7, 87.3 };
centimize(varray);
for(int j=0; j<MAX; j++)
    cout << varray[j] << endl;
return 0;
}
void centimize(double* ptr)
{
for(int j=0; j<MAX; j++)
    *ptr++ *= 2.54;                       //ptrd points to elements of varray
}
```

- Question : *ptr++ will be interpreted as *(ptr++) or (*ptr)++ ?

Question-1

```
int main()
{
    int *ptr;
    int x;

    ptr = &x;
    *ptr = 0;

    cout<< x;
    cout<<*ptr;

    *ptr += 5;
    cout<< x;
    cout<<*ptr;

    (*ptr)++;
    cout<< x;
    cout<<*ptr;

    return 0;
}
```

Question-2:What is the output for printxy(1,1)?

```
void printxy(int x, int y)
{
   int *ptr;
   x = 0;
   ptr = &x;
   y = *ptr;
   *ptr = 1;
   cout<<x<<endl<<y;
}
```

Question-3

```
int main()
{
    float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5};
    float *ptr1 = &arr[0];
    float *ptr2 = ptr1 + 3;

    cout<<*ptr2;
    cout<<ptr2 - ptr1;

   return 0;
}
```

Question-4

```
void f(int *p, int *q)
{
  p = q;
  *p = 2;
}
int i = 0, j = 1;
int main()
{
  f(&i, &j);
  cout<< i << j;
  return 0;
}
```

Question-5

```
int main()
{
   int arr[] = {1, 2, 3, 4, 5};
   int *p = arr;
   ++*p;
   p += 2;
   cout<< *p;
   return 0;
}
```

Question-6

```
int x;
void Q(int z)
{
    z += x;
    cout<<z;
}
void P(int *y)
{
    int x = *y + 2;
    Q(x);
    *y = x - 1;
    cout<<x;
}
int main()
{
    x = 5;
    P(&x);
    cout<<x;
}
```

Question-7

```
int main()
{
 int var;

 void *ptr = &var;
 *ptr = 5;
 cout<<var<<endl<<*ptr;

 return 0;
}
```

Question-7

```
int main()
{
 int var;

 void *ptr = &var;
 *(int*)ptr = 5;
 cout<<var<<endl<<*(int*)ptr;

 return 0;
}
```

Question-8

```
void mystery(int *ptra, int *ptrb)
{
  int *temp;
  temp = ptrb;
  ptrb = ptra;
  ptra = temp;
}
int main()
{
   int a=2016, b=0, c=4, d=42;
   mystery(&a, &b);
   if (a < c)
      mystery(&c, &a);
   mystery(&a, &d);
   cout<<a;
}
```

Question-9

```
void f(int* p, int m)
{
   m = m + 5;
   *p = *p + m;
   return;
}
void main()
{
   int i=5, j=10;
   f(&i, j);
   cout<<i+j;
}
```

Question-10

```
int main()
{
   int x=30, *y, *z;
   y=&x;                        /* Assume address of x is 500 and
                                      Integer is 4 byte size */

   z=y;
   *y++=*z++;
   x++;
   cout<<x << y << z;
   return 0;
}
```

Question-11

```
int main()
{
    int a = 5;
    int* const ptr=&a;
    ptr=&b;
    ptr = &a;
    *ptr = 10;
    cout<< a;
    return 0;
}
```

Question-12

```
int* fun()
{
    int A = 10;
    return (&A);
}

int main()
{
    int* p;
    p = fun();
    cout<<p;
    cout<<*p;
    return 0;
}
```

Ordering with Pointers :

```
int main()
{       void order(int*, int*);
        int n1=99, n2=11;
        int n3=22, n4=88;
        order(&n1, &n2);
        order(&n3, &n4);
        cout << “n1=” << n1 << endl;
        cout << “n2=” << n2 << endl;
        cout << “n3=” << n3 << endl;
        cout << “n4=” << n4 << endl;
         return 0;
}
void order(int* numb1, int* numb2)        //orders two numbers
{
        if(*numb1 > *numb2)                              //if 1st larger than 2nd,
        {       int temp = *numb1;                                  //swap them
                *numb1 = *numb2;
                *numb2 = temp;
        }
}
```

Sorting Array elements

```
int main()
{       void bsort(int*, int);
         const int N = 10;
         int arr[N] = { 37, 84, 62, 91, 11, 65, 57, 28, 19, 49 };
         bsort(arr, N);
         for(int j=0; j<N; j++)
         cout << arr[j] << “ “;
         cout << endl;
         return 0;            }
void bsort(int* ptr, int n)
{       void order(int*, int*);
         int j, k;
         for(j=0; j<n-1; j++)
               for(k=j+1; k<n; k++)
                   order(ptr+j, ptr+k);     }
void order(int* numb1, int* numb2)
{     if(*numb1 > *numb2)
         {    int temp = *numb1;
               *numb1 = *numb2;
               *numb2 = temp;            }
}
```

Pointers and C-Type Strings

```
int main()
{
char str1[] = “Defined as an array”;
char* str2 = “Defined as a pointer”;
cout << str1 << endl;
cout << str2 << endl;
// str1++;                    // can’t do this; str1 is a constant
str2++;                       // this is OK, str2 is a pointer
cout << str2 << endl;         // now str2 starts “efined...”
return 0;
}
```

Strings as Function Arguments

```
int main()
{
void dispstr(char*);
char str[] = “Idle people have the least leisure.”;
dispstr(str);
return 0;
}
void dispstr(char* ps)
{
while( *ps )                    //until null character,
    cout << *ps++;                 //print characters
cout << endl;
}
```

Copying a String Using Pointers

```
int main()
{
void copystr(char*, const char*);
char* str1 = “Self-conquest is the greatest victory.”;
char str2[80];
copystr(str2, str1);         //copy str1 to str2
cout << str2 << endl;
return 0;
}
void copystr(char* dest, const char* src)
{
while( *src )                              //until null character,
     *dest++ = *src++;                     //copy chars from src to dest
*dest = ‘\0’;                              //terminate dest
}
```

Question

```
void swap(char *str1, char *str2)
{
  char *temp = str1;
  str1 = str2;
  str2 = temp;
}

int main()
{
  char *str1 = “hello";
  char *str2 = “hi";
  swap(str1, str2);
  cout<<str1<<str2;
  return 0;
}
```

Solution-1

```
void swap1(char **str1_ptr, char **str2_ptr)
{
  char *temp = *str1_ptr;
  *str1_ptr = *str2_ptr;
  *str2_ptr = temp;
}

int main()
{
  char *str1 = “hello";
  char *str2 = “hi";
  swap1(&str1, &str2);
  cout<<str1<<str2;
  return 0;
}
```

Solution-2

```
void swap2(char *str1, char *str2)
{
  char *temp = new char[strlen(str1)+1];
  strcpy(temp, str1);
  strcpy(str1, str2);
  strcpy(str2, temp);
  delete temp;
}
int main()
{
  char str1[10] = “hello";
  char str2[10] = “hi";
  swap2(str1, str2);
  cout<<str1<<str2;
  return 0;
}
```

Question

```
int main()
{
   char str1[] = "hello";
   char str2[] = “bye";
   char *s1 = str1, *s2=str2;
   while(*s1++ = *s2++)
      cout<<str1;

   cout<<endl;
   return 0;
}
```

- OP: bellobyllobyelo

Question

```
void fun(int *p)
{
 int q = 10;
 p = &q;
}
int main()
{
 int r = 20;
 int *p = &r;
 fun(p);
 cout<< *p;
 return 0;
}
```

Question

```
#include<iostream>
using namespace std;
void fun(char** str_ref)
{
    str_ref++;
}
int main()
{
    char *str = new char[100*sizeof(char)];
    strcpy(str, “Hello");
    fun(&str);
    cout<<str;
    delete str;
    return 0;
}
```

Question

```
int main(void)
{
   int i;
   int *ptr=new int[5*sizeof(int)];

   for (i=0; i<5; i++)
      *(ptr + i) = i;

   cout<<*ptr++;
   cout<<(*ptr)++;
   cout<<*ptr;
   cout<<*++ptr;
   cout<<++*ptr;
}
```

Question

```
int fun(int  arr[]) {
   arr = arr+1;
   cout<<arr[0];
}
int main(void) {
   int arr[2] = {10, 20};
   fun(arr);
   cout<<arr[0];
   return 0;
}
```

Question

```
int f(int x, int *py, int **ppz)
{
 int y, z;
 **ppz += 1;
  z  = **ppz;
 *py += 2;
  y = *py;
  x += 3;
  return x + y + z;
}
 void main()
{
  int c, *b, **a;
  c = 4;
  b = &c;
  a = &b;
  cout<< f(c,b,a);
  }
```

new and delete operator

- These operators are used for dynamic memory.
- new operator is used for dynamic memory allocation and delete operator is used for dynamic memory deallocation.
- Dynamically allocated memory is allocated on **Heap** and non-static and local variables get memory allocated on **Stack.**
- The most important use is flexibility provided to programmers. We are free to allocate and deallocate memory whenever we need and whenever we don’t need anymore.

new operator

- This versatile operator obtains memory from the operating system and returns a pointer to its starting point.
- Syntax:

  pointer-variable = **new** data-type;

  Eg. int *p=new int;

  pointer-variable = **new** data-type[size];

  Eg: int *p=new int[10];

delete operator

- delete operator is used to release memory pointed by pointer-variable.
- Syntax:

  delete pointer-variable;

  Eg: delete p;

  delete[] pointer-variable;

  Eg: delete[] p;

Example

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
char* str = “Idle hands are the devil’s workshop.”;
int len = strlen(str);
char* ptr;
ptr = new char[len+1];
strcpy(ptr, str);
cout << “ptr=” << ptr << endl;
delete[] ptr;
return 0;
}
```

Example

```
class String
{
private:
char* str;
public:
String(char* s)
{
int length = strlen(s);
str = new char[length+1];
strcpy(str, s);
}
~String()
{
cout << “Deleting str.\n”;
delete[] str;
}
void display()
{
cout << str << endl;
}    };
int main()
{ String s1 = “Who knows nothing doubts nothing.”;
  cout << “s1=”;
  s1.display();           }
```

Overloading of new and delete operator

```
#include<iostream>
using namespace std;
void * operator new(size_t size)
{
   cout << "New operator overloading " << endl;
   void * p = malloc(size);
   return p;
}
void operator delete(void * p)
{
   cout << "Delete operator overloading " << endl;
   free(p);
}
int main()
{
   int n = 5, i;
   int * p = new int[5];
   for (i = 0; i<n; i++)
      p[i]= i;
   cout << "Array: ";
   for(i = 0; i<n; i++)
      cout << p[i] << " ";
   cout << endl;
   delete p;
}
```

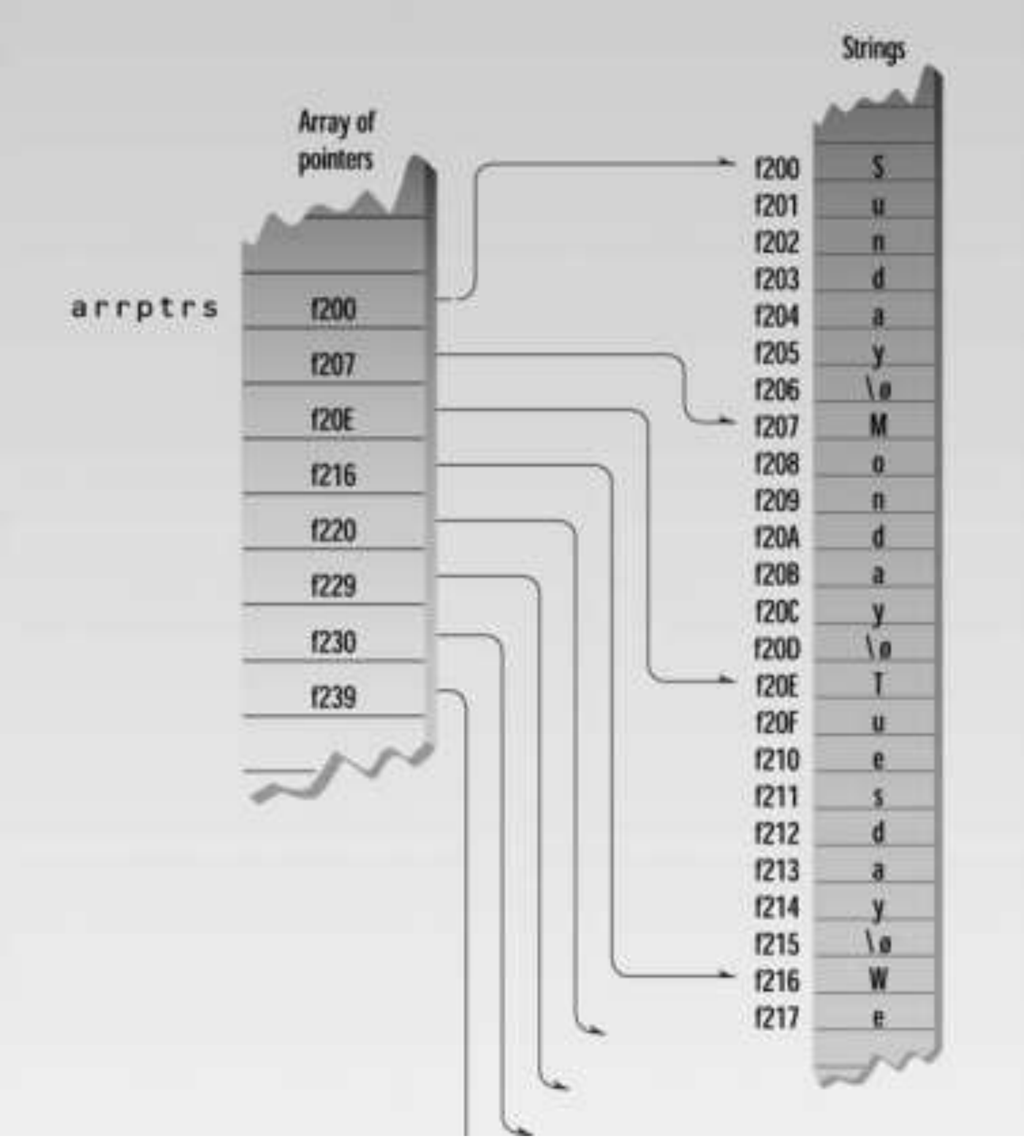
Example

```
class student
{
   char name[20];
   int age;
public:
   student()
   {    cout<< "Constructor is called\n" ;     }
   student(char n1[], int a1)
   {    strcpy(name,name1);;
          age = a1 ;          }
   void display()
   {   cout<< "Name:" << name << endl;
      cout<< "Age:" << age << endl;      }
   void * operator new(size_t size)
   {
      cout<< "Overloading new operator with size: " << size << endl;
      void * p = malloc(size);
      return p;
   }
  void operator delete(void * p)
   {
      cout<< "Overloading delete operator " << endl;
      free(p);
   }
};
int main()
{
   student * p = new student("Yash", 24);
   p->display();
   delete p;
}
```

Array of pointers to strings

```
#include <iostream>
using namespace std;
const int DAYS = 7;      //number of pointers in array
int main()
{
char* arrptrs[DAYS] = { “Sunday”, “Monday”, “Tuesday”,
                        “Wednesday”, “Thursday”,
                              “Friday”, “Saturday” };

for(int j=0; j<DAYS; j++)     //display every string
   cout << arrptrs[j] << endl;
return 0;
}
```


Array of pointers
arrptrs
f200
f207
f20E
f216
f220
f229
f230
f239
Strings
f200 S
f201 u
f202 n
f203 d
f204 a
f205 y
f206 \0
f207 M
f208 o
f209 n
f20A d
f20B a
f20C y
f20D \0
f20E T
f20F u
f210 e
f211 s
f212 d
f213 a
f214 y
f215 \0
f216 W
f217 e

Array of pointers to integers

```
const int SIZE = 3;

void main()
{
   // creating an array
   int arr[] = { 1, 2, 3 };

   int i, *ptr[SIZE];

   for (i = 0; i < SIZE; i++) {

      // assigning the address of integer.
      ptr[i] = &arr[i];
   }

   // printing values using pointer
   for (i = 0; i < SIZE; i++) {

      cout<<*ptr[i];
   }
}
```

Pointers to Objects

```
class Distance
{
  private:
    int feet;
    float inches;
  public:
    void getdist()
    {
      cout << “\nEnter feet: “;   cin >> feet;
      cout << “Enter inches: “;   cin >> inches;
    }
    void showdist()
    {   cout << feet << “\’-” << inches << ‘\”’;   }
};
int main()
{
  Distance dist;
  dist.getdist();
  dist.showdist();
  Distance* distptr;
  distptr = new Distance;
  distptr->getdist();
  distptr->showdist();
  return 0;
}
```

Another Approach to new

```
class Distance
{
  private:
    int feet;
    float inches;
  public:
    void getdist()
    {
      cout << “\nEnter feet: “; cin >> feet;
      cout << “Enter inches: “; cin >> inches;
    }
    void showdist()
   { cout << feet << “\’-” << inches << ‘\”’; }
};
int main()
{
  Distance& dist = *(new Distance);
  dist.getdist();
  dist.showdist();
}
```

Array of Pointers to Objects

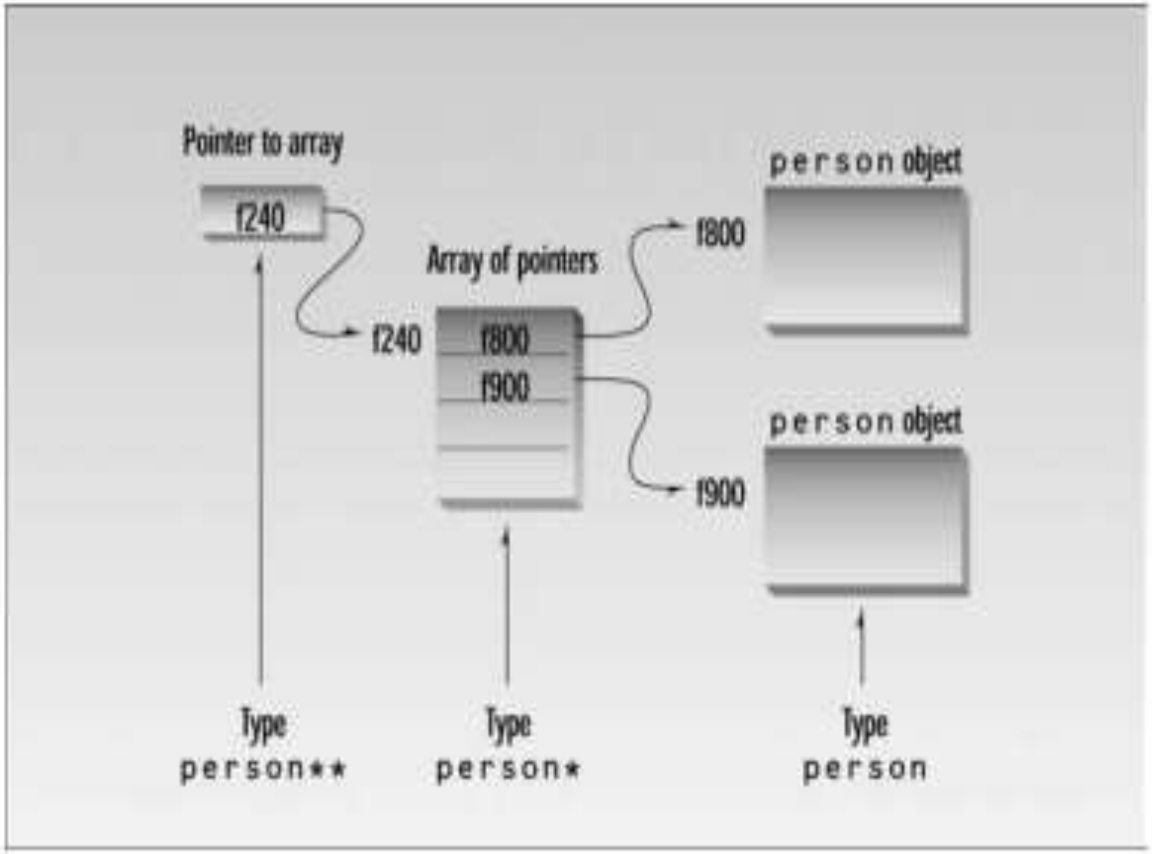
```
class person
{
  protected:
  char name[40];
  public:
  void setName()
  {
    cout << “Enter name: “;
    cin >> name;
  }
  void printName()
  {
    cout << “\n Name is: “ << name;
  }
};
```

```
int main()
{
  person* persPtr[100];
  int n = 0;
  char choice;
  do
  {
    persPtr[n] = new person;
    persPtr[n]->setName();
    n++;
    cout << “Enter another (y/n)? “;
    cin >> choice;
  }
  while( choice==’y’ );
  for(int j=0; j<n; j++)
  {
    cout << “\nPerson number “ << j+1;
    persPtr[j]->printName();
  }
}
```

Pointers to pointers

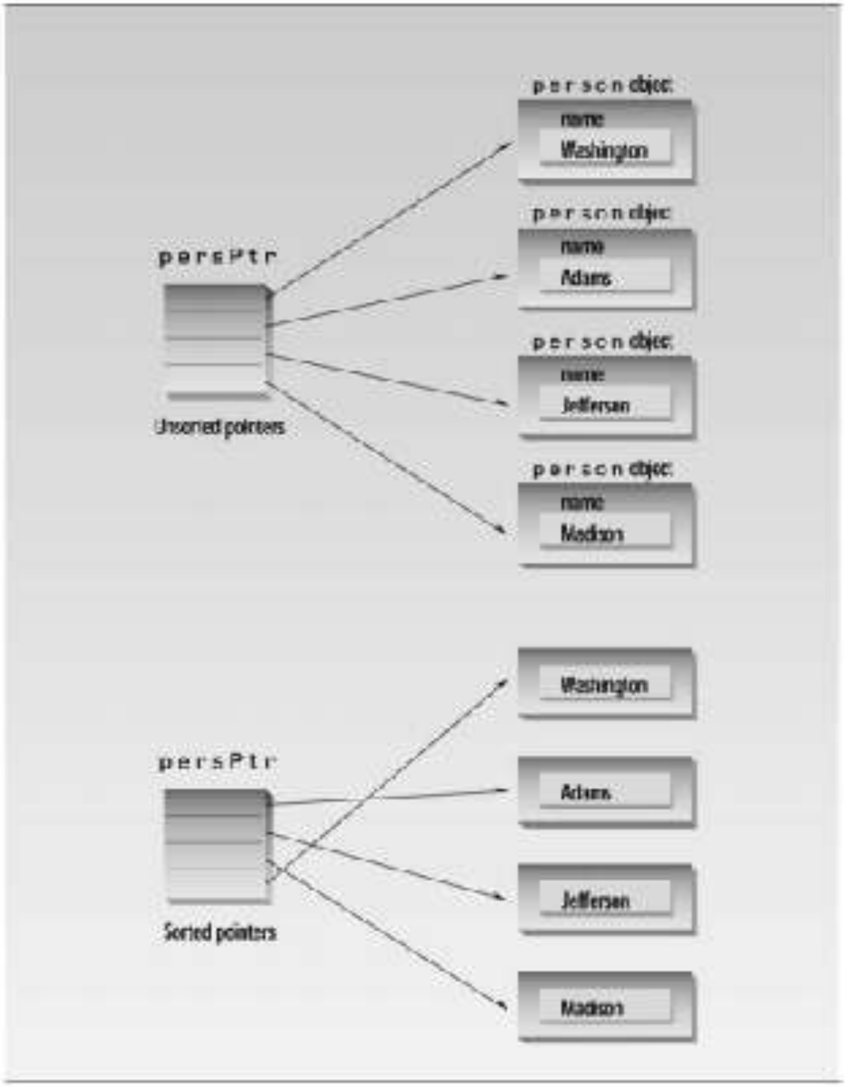
```
class person
{
  protected:
    String name;
  public:
  void setName()
  {
    cout << “Enter name: “;
    cin >> name;      }
  void printName()
  {    cout << “\n Name is: “ << name;      }
  string getName()
  { return name; }
};
int main()
{
  person* persPtr[100];
  int n = 0;
  char choice;
  do
  {
    persPtr[n] = new person;
    persPtr[n]->setName();
    n++;
    cout << “Enter another (y/n)? “;
    cin >> choice;
  }     while( choice==’y’ );
  bsort(persPtr, n);
```

```
 for(int j=0; j<n; j++)
  {
    cout << “\nPerson number “ << j+1;
    persPtr[j]->printName();
  }
}
void bsort(person** pp, int n)
{
  void order(person**, person**);
  int j, k;
  for(j=0; j<n-1; j++)
    for(k=j+1; k<n; k++)
      order(pp+j, pp+k);
}
void order(person** pp1, person** pp2)
{
  if( (*pp1)->getName() > (*pp2)->getName() )
  {
    person* tempptr = *pp1;    //swap the pointers
    *pp1 = *pp2;
    *pp2 = tempptr;
  }
}
```



Pointer to array
f240
f240
Array of pointers
f800
f900
f800
person object
f900
person object
Type
person**
Type
person*
Type
person

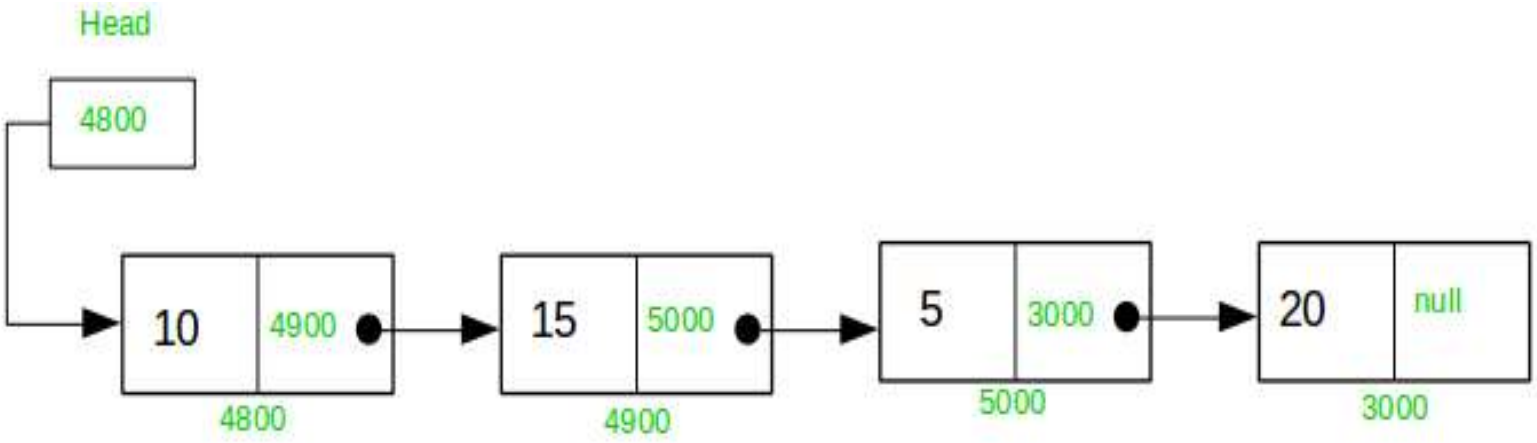
Contd..


person objec
name
Washington
person objec
name
Adams
person objec
name
Jefferson
person objec
name
Madison
persPtr
Unsorted pointers
Washington
Adams
Jefferson
Madison
persPtr
Sorted pointers

Linked List

- Linked List is a linear data structure.
- Representation:
- A linked list is represented by a pointer to the first node of the linked list.
- The first node is called the head. If the linked list is empty, then the value of the head is NULL.
- Each node in a list consists of at least two parts:
  1) Data
  2) Pointer (Or Reference) to the next node

Example of linked list



Array vs. Linked List

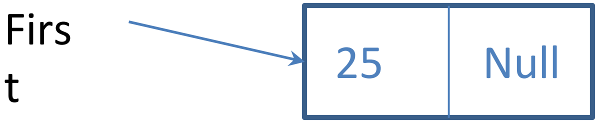
- Limitation of Array:
- The size of the arrays is fixed.
- Insertion and deletion in array is expensive.
- Advantages of linked list:
- Dynamic size
- Ease of insertion/deletion
- Drawbacks of linked list:
- Random access is not allowed.
- Extra memory space for a pointer is required with each element of the list.

A Linked list Example

```
struct link
{
  int data;
  link* next;
};
class linklist
{
  private:
    link* first;
  public:
    linklist()
    { first = NULL; }
    void additem(int d);
    void display();
};
```

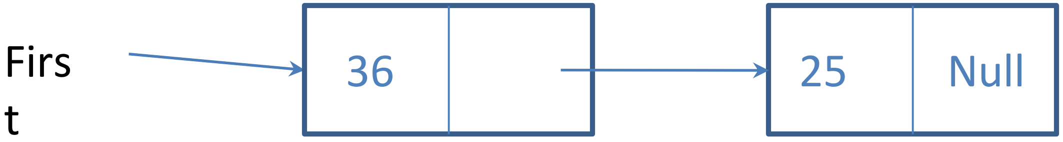
Insertion

```
void linklist::additem(int d)
{
  link* newlink = new link;
  newlink->data = d;
  newlink->next = first;
  first = newlink;
}
```



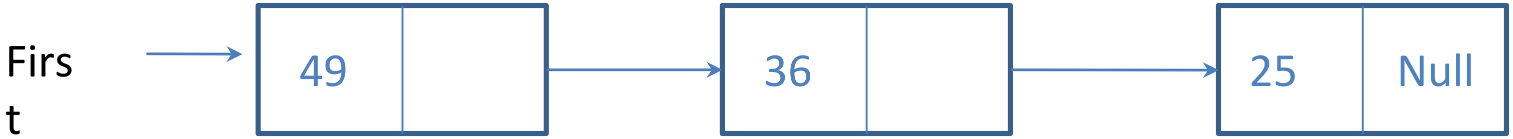
Insertion

```
void linklist::additem(int d)
{
  link* newlink = new link;
  newlink->data = d;
  newlink->next = first;
  first = newlink;
}
```



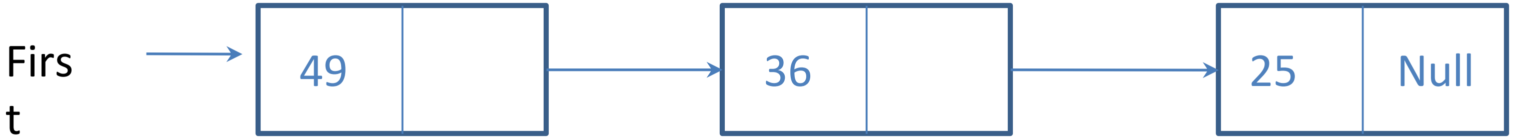
Insertion

```
void linklist::additem(int d)
{
   link* newlink = new link;
   newlink->data = d;
   newlink->next = first;
   first = newlink;
}
```



Display

```
void linklist::display()
{
   link* current = first;
   while( current != NULL )
   {
     cout << current->data << endl;
     current = current->next;
   }
}
```


Firs
t
49
36
25
Null