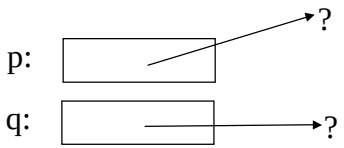
Pointers

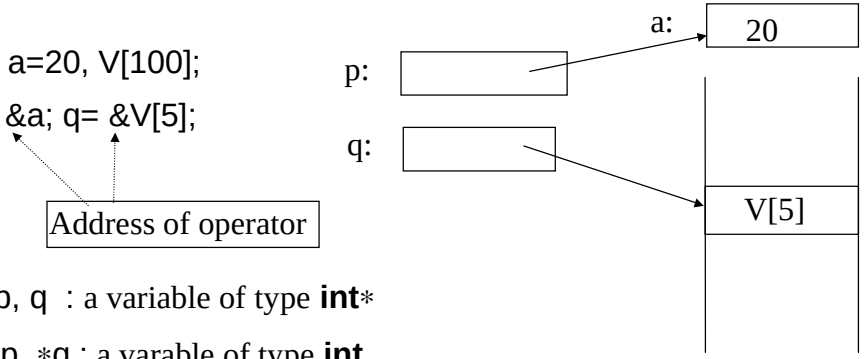
Pointers are addresses used to reference variables and to access memory.

- A pointer definition merely creates the pointer variable;
- It neither initializes the pointer variable nor allocates memory space for the variable to point to.

int ∗p, ∗q;



int a=20, V[100];
p= &a; q= &V[5];

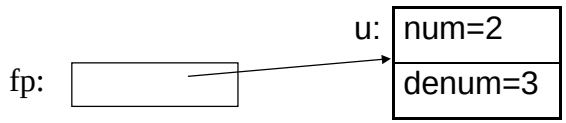


- p, q : a variable of type **int**∗
- ∗p, ∗q : a varable of type **int**

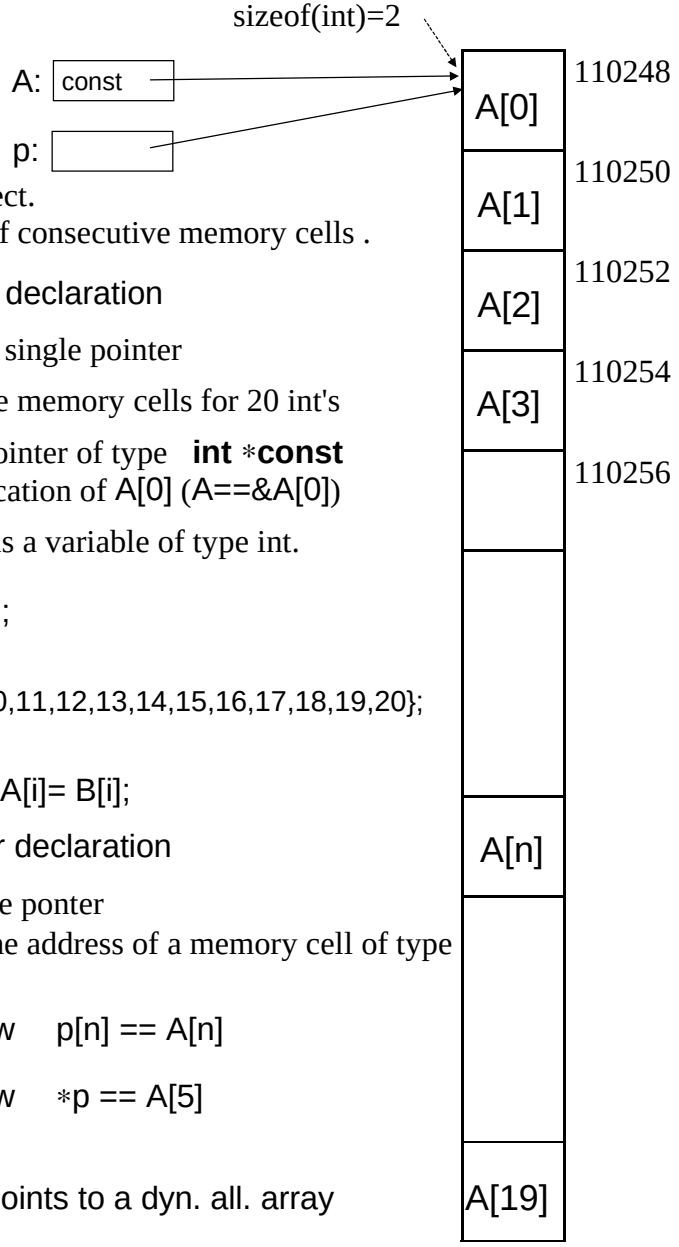
a= V[5] + 100; ~ ∗p= ∗q +100; // ∗ ~ value of operator

```
class Fraction
{ public:
    int num;
    unsigned int denom;
};
```

Fraction u={2,3};
Fraction ∗fp = &u
(∗fp).num= (∗fp).denum +1;
fp->num= fp->denum+1;
u.num= u.denum+1;



C-Arrays/ Low-level array's



a C-array is not an object.
a C-array is a section of consecutive memory cells .

int A[20]; // an array declaration

- Allocates space for a single pointer
- Allocates consecutive memory cells for 20 int's
- The *value of A* is a pointer of type **int** ∗**const**
  It's value is the location of A[0] (A==&A[0])
- For 0 ≤ n ≤ 19, A[n] is a variable of type int.

A[5]= 10; A[n]+= A[5];

int B[ ] =
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};

A= B // ILLEGAL
for(**int** i=0; i<20;i++) A[i]= B[i];

int ∗p; // a pointer declaration

- allocates space for the ponter
- p must be assigned the address of a memory cell of type int before used

p= A ; // now p[n] == A[n]

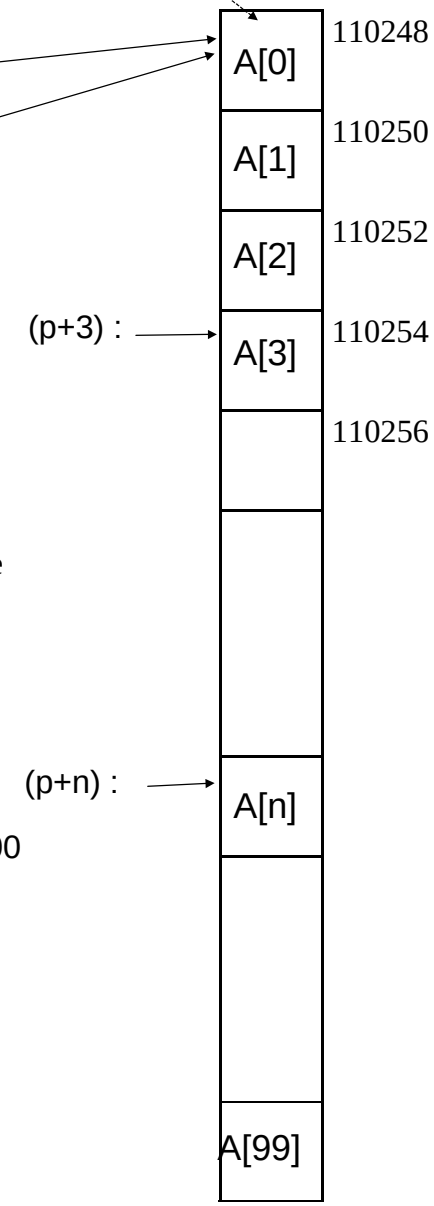
p= & A[5] // now ∗p == A[5]

int n= 20;
p= new int[n] // p points to a dyn. all. array

Pointer Arithmetic

Pointer + Integer

Assume sizeof(int)=2



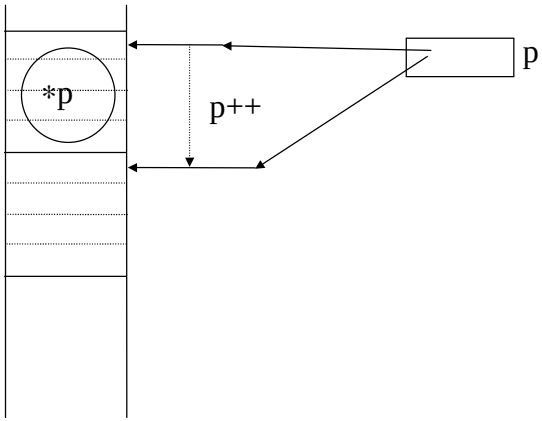

```
int A[100];
int *p;
p= A ;          //now p[n] == A[n]
int n;
```

- int(p+n) - int(p) == n * sizeof(int)
- (p+n) == & A[n]
- (p+n): address of an integer varable
- *(p+n) == A[n]
- *(p+n) : a variable of type int

```
*(p+5)= 100;
cout << A[5] << endl;   // result: 100

for (int i=0; i<100; i++)
    {cout << A[i] << endl;}
p= A;
for (int i=0; i<100; i++)
    {cout << *p << endl; p++ ;}
```


Pointer + Integer

```
#include <iostream>

enum{XSIZE=100};
long int X [XSIZE];

long int sum(long int L[ ], int max)
  {long int s=0;
  for (int i=0; i<max; i++) s= s + L[i];
  return(s);
  }

long int sum1(long int L[ ], int max)
  {long int *p=L, s=0;
   for (int i=0; i<max; i++) s= s + *p++;
   return(s);
  }

int main()
  {for (int i=0;i<XSIZE; i++)X[i]= i;
   cout << sum(X, XSIZE) << endl;
   cout << sum1(X, XSIZE) << endl;
   return(0);
  }
```



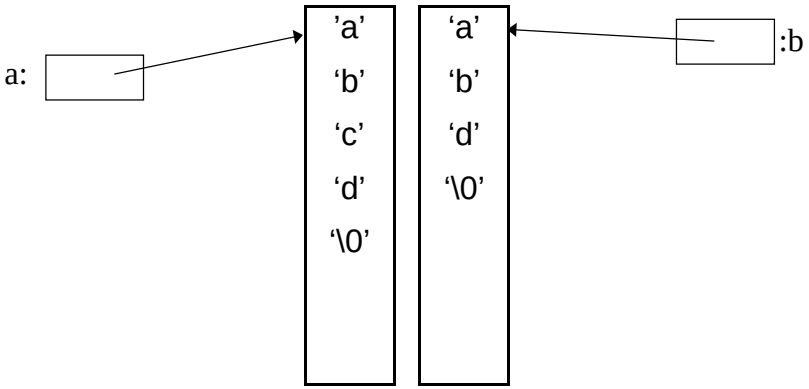
Pointer + Integer

strcmp, a Standard Library Function

//////// comparing strings ///////

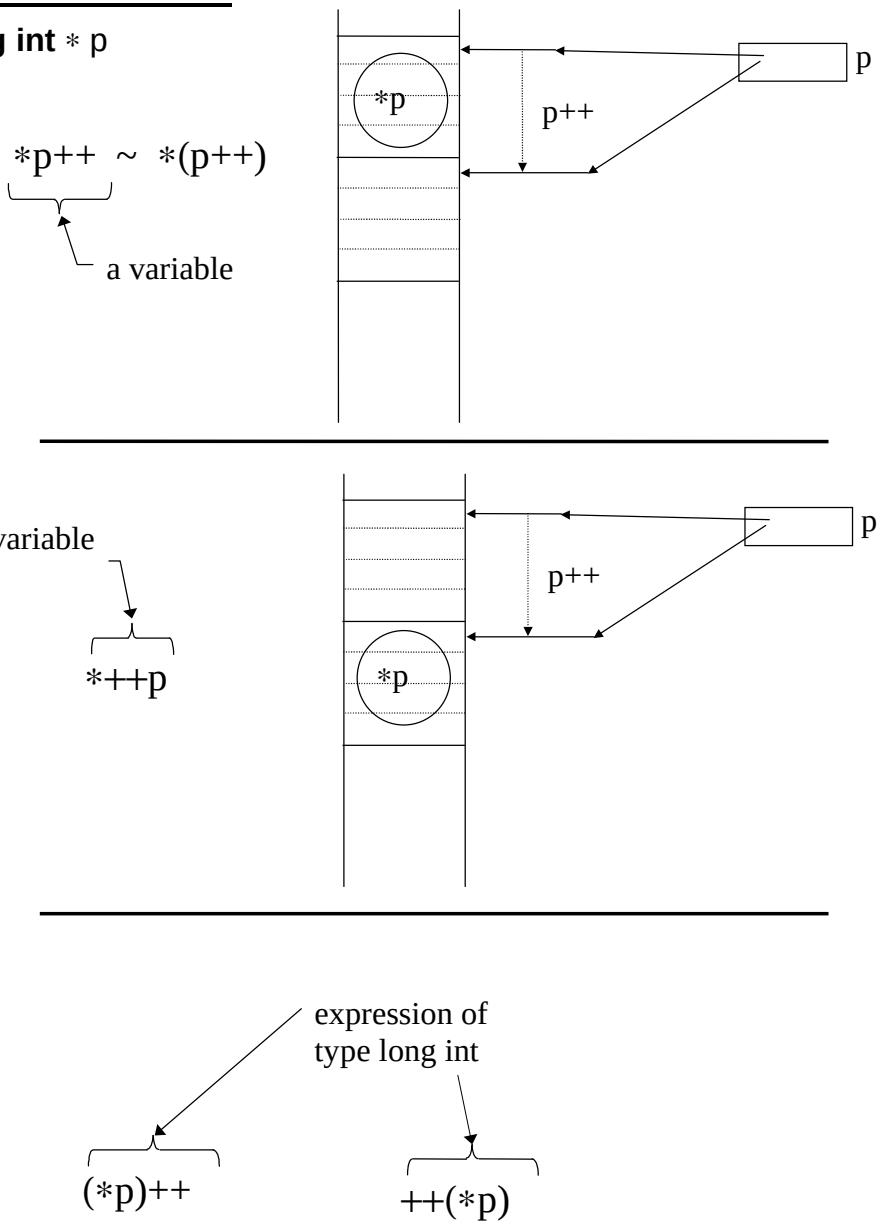
```
#include <iostream>
#include <cstring>

int strcmp(const char *r, const char *s)
  { while (*r == *s)
      { if (*r == '\0') return (0);
        r++; s++;
      }
    return(*r - *s);
  }
main()
  { char * a="abcd";
    char * b="abd";
    if (strcmp(a,b)  < 0) cout << "OK\n";
  }
```

~ **const char** s []



Pointer Arithmetic

long int * p



Dynamic Storage Allocation

```
////file: vector2.h
class Vector2
{ public:
      Vector2(float a, float b);
      float inner(Vector2 a);
  private:
      float x, y;
};
```

```
#include "vector2.h"

void F(int n)
{ Vector2 L(2.2f,6.7f);  //local
  Vector2 * p= new Vector2(1.2f,3.4f);  //dynamic
  …
  delete p;
…
}
```

- p: automatic storage
- the Vector2-object pointed to by p: dynamic storage

Delete object pointed to by p !!

Values that are *dynamically allocated resides in the heap*.

A dynamically allocated value belongs to the dynamic storage class

C++	Java
It is the programmers responsibility to delete dynamically allocated memory!!	All objects are dynamically allocated. Objects no longer referenced are deleted automatically by the garbage collector

Dynamic allocation of array's:

```
int size= …;
int * A= new int[size]
…
delete [ ] A;
```


Storage Allocation

```
int A [8];

int * p, *q;

int f(int n)
  {  int a= 2*n;
     if (n>0) return a+ f(n-1);
     else
        {p= new int(4); *q= 5;
        return 0;
        }
  }

int main()
  {  q= new int;
     cout << "f(4)= " << f(4) << endl;
     delete p;
     delete q;
     return(0);
  }
```