Pointers

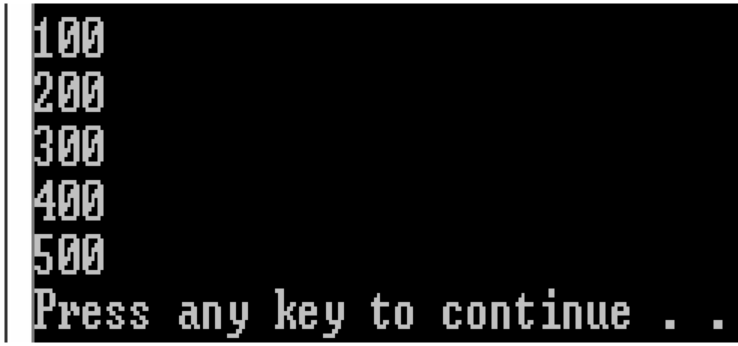
contd

Pointer and arrays

- array elements can be accessed using pointers as well.
- There are two ways to do that.

- **By incrementing pointer variables**

```
int num[5]={100,200,300,400,500};
int* ptr=num;

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

*ptr = num
```



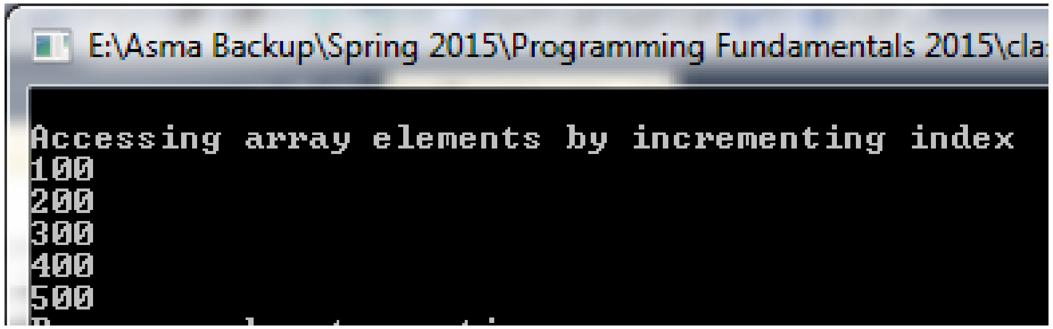
```
100
200
300
400
500
Press any key to continue . .
```

-Declares a pointer ptr and initializes it to the first element of the array num

- The post increment operator moves the pointer reference forward in the memory
- The change of memory addresses referenced by the pointer depends on the data type of pointer
- If the pointer is pointing to an int datatype or it is a pointer to int, the reference will be changed by 2 by 4 bytes in it.
- `

- in case of char ptr, it will change the reference by 1 byte.
- similarly to reference back we can use the decrement operator
- the increment/decrement also shows that why the compiler needs to know that whether a pointer is a pointer to int or double or any datatype so that it can perform the correct arithmetic to access the elements of the array.
- **By incrementing index**

```
for(int i=0; i <5 ; i++)
              cout<< *(num+i)<<endl;
```


E:\Asma Backup\Spring 2015\Programming Fundamentals 2015\cla

```
Accessing array elements by incrementing index
100
200
300
400
500
```

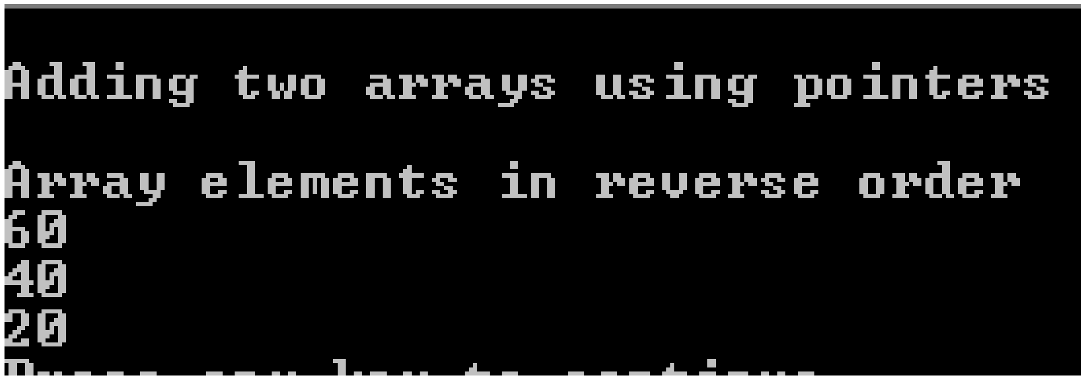
- the expression *(num+i) ==num[i]
- name of the array is an address hence when we add i to it, the address changes

(2000+0x4) =(2000)=100
(2000+1x4) =(2004)=200
(2000+2x4) =(2008)=300

Example

- Adding two arrays using pointers notation and displaying in the reverse order using pointers notation

```
void main()
{

cout<<endl<<"Adding two arrays using pointers\n";

int num1[3]={10,20,30};
int num2[3]={10,20,30};
int num3[3];

int *num1ptr=num1;
int *num2ptr=num2;
int *num3ptr=num3;

for(int i=0;i<3;i++)
*num3ptr++=*num1ptr++ + (*num2ptr++);

cout<<endl<<"Array elements in reverse order\n";
cout<<*(--num3ptr)<<endl;
cout<<*(--num3ptr)<<endl;
cout<<*(--num3ptr)<<endl;
}
```



```
Adding two arrays using pointers

Array elements in reverse order
60
40
20
```

Pointer constant and variable

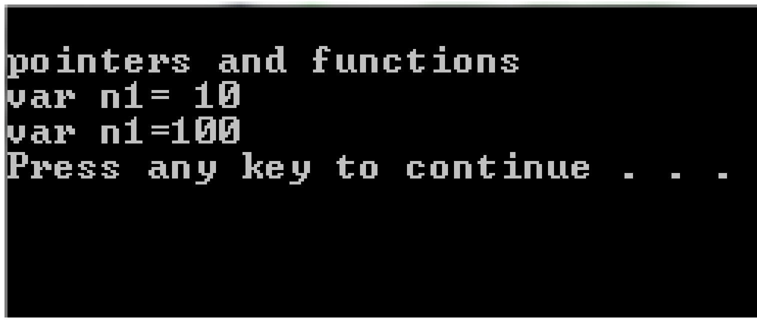
- if we wanted to do *(num1++),then its not possible because
  - num1 is the address where the system has chosen to place your array and it will stay at this address until the program terminates
  - we can say that num1 i.e. the name of the array is a pointer constant and constants can't change.

The solution is if we can't increment an address , we can increment a pointer that holds an address as in previous example

Pointer and Functions

- if we want to modify the variables in the calling program, these can’t be passed by value, but a reference argument or pointer can be used in this situation

```
void square(int *);//function take an argument that is pointer to int.
void main
{
            cout<<endl<<"pointers and functions\n";
            int n1;
            n1=10;
            cout<<"var n1= "<<n1<<endl;
            square(&n1);
            cout<<"var n1="<<n1<<endl;

}
```



```
pointers and functions
var n1= 10
var n1=100
Press any key to continue . . .
```

- void square(int *num)
- { *num *= *num; }

- function takes an argument that is a pointer to int

  void square (int *);

- when main calls the function, it supplies the address of the variable as the argument square(&n1);

- It’s not the variable itself but it’s address

- As the function is passed the address to access the contents of this address we need to use the dereference operator.

- One thing common to send by reference & send by pointer is that both permit the variable in the calling program to be modified by the function.

Pointers and passing arrays using pointers

- to pass array to a function we used

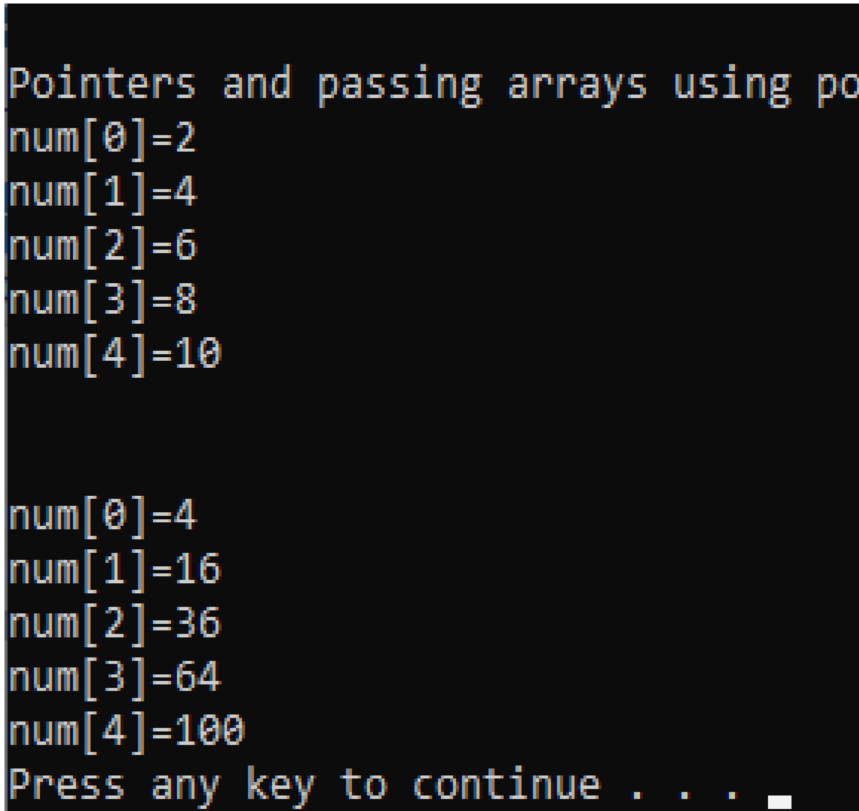
  void square(int [])

- In pointers this changes to

  void square(int *)

- because the name of the array is the arrays address, there is no need for the address operator & when function is called square(num);
- In function this address is placed in pointer num & each element is accessed using *num & to proceed to next element num++
- *num++ is evaluated as *(ptrnum++)i.e. increment the address not the contents.

```
const int size=5;
void square(int*);
void main()
{
cout<<endl<<"Pointers and passing arrays using pointers\n";
int num[]={2,4,6,8,10};
for (int j=0; j<size; j++)
cout<<"num["<<j<<"]="<<num[j]<<endl;

square(num);

for (int j=0; j<size; j++)
cout<<"num["<<j<<"]="<<num[j]<<endl;

system("pause");
}
void square(int *ptrnum)
{
 for (int i=0;i<size;i++)
 {
int a=(*ptrnum)    *    (*ptrnum);
*ptrnum=a;
++ptrnum;
}
}
```



```
Pointers and passing arrays using po
num[0]=2
num[1]=4
num[2]=6
num[3]=8
num[4]=10


num[0]=4
num[1]=16
num[2]=36
num[3]=64
num[4]=100
Press any key to continue . . .
```

HOME ASSIGNMENT

- To swap elements using pointers

- search Maxvalue in an array of 10 elements using ptr notation only i.e. all reference to array elements should be made via pointer notation.

- search all occurrences of an element in an array of 10 elements using ptr notation

- initialize an array of 10 elements using pointers by making functions and display it using functions and pointer

-search a particular element in the array