Programming techniques

Week 10: Arrays with Structure Elements

4/2024

What is in today?

- Defining and using arrays of arrays
- Remember pointer arithmetic

Subscript Operator

- The subscript operator provides access to individual elements of an array.
- The subscript operator is a binary operator.
- The first operand is an expression designating the address of the first element of an array; this can be the array identifier, or as we will see later, this can also be a pointer expression.
- The second operand is an integer expression contained within the brackets designating the element of the array to be accessed.
- The first element of an array always begins at index zero and the last element of an array ends at the index that is one less than the size of the array; thus, legal indices fall within the range 0 through size-1.

Subscript Operator

```
//for some statically allocated array:
cout <<"address of array is " <<array << endl;
for(int i=0; i<size; ++i)    //loop for i=0...6
  cout <<"array[" <<i <<"] equals " <<array[i] << endl;
```

- Each of the elements of the array sequentially follow one another in memory.
- We can think of each element of an array as an unnamed variable that we identify by using an index.
- The actual address of each element is computed by the subscript operator and takes into the account the size of the elements in the array.
- The index is independent of the actual address of each element.

sizeof Operator

- The only other operator that can be directly applied to arrays is the sizeof operator.
- The sizeof operator returns the number of characters (bytes) that an array occupies.
- The number of elements in an array can be determined by dividing the size of the array by the size of an element in the array.

```
cout <<"size of int array = "
        <<sizeof(array) <<endl;
cout <<"size of int = " <<sizeof(int) <<endl;
cout <<"number of elements in the array = "
     <<sizeof(array)/sizeof(array[0]) << endl;
```

Pointers and Arrays

- We can also use pointers to point to data that is stored sequentially in memory.
- We can treat a pointer to data stored sequentially in memory as an array.
- All operations on arrays have an equivalent pointer representation.
- We can take advantage of this to improve our programs' performance when operating on arrays.

Pointers and Arrays

- It is possible to define the behavior of the subscript operator entirely in terms of operations on a pointer.
- The first thing we need to know is that the identifier of an array is a constant pointer to the first element of that array.
- It is a pointer to the same type as the elements of the array.
- This means that we can initialize or assign an array name to a pointer, where the pointer points to data of the same type as the elements of our array.

Pointers and Arrays

```
int ai[7];  //ai is of type pointer to int
int* pi;     //pi is a pointer to int
pi = ai;     //pi now points to the array ai
```

- We now have two ways to access elements of an array, one using the name of the array (ai) and the other using a pointer (pi).
- In this example, the name of the array (ai) is a constant pointer to an int.
- The pointer (pi) is a variable pointer to an int that has been assigned the same address as ai.

Pointers and Arrays

- Since the value of ai has been assigned to pi, the residual value of using either ai or pi in an expression is the same in either case:
  - it is the address of the first element of the array.
- When ai is used in an expression, that expression uses the value that the constant ai represents.
- When pi is used in an expression, that expression uses the value currently assigned to variable pi.
- We can apply the subscript operator to this residual value (an expression of type pointer to an int) in order to access the elements of the array.

Pointers and Arrays

```
int *pi;                    //pi is a pointer to an int
pi = ai;                    //same as pi = &a[0]
for(int i=0; i<7; i++)
 if(ai[i] != pi[i])
   cout <<"Oops - big trouble in River City"<<endl;
```

- The relationship between pointers and arrays is defined by the following identity, where E1 is a pointer (either an array name or a pointer expression) and E2 is an integer expression.

E1[E2] == *((E1)+(E2))

Pointers and Arrays

```
ai[3] = 42;    //this stores 42 w/array subscripting
*(ai+3)=42;    //same thing using pointer operations
*(3+ai)=42;    //addition is communitive
3[ai] = 42;    //this works!
```

- This identity means that the subscript operation is equivalent to adding the index to the pointer expression and then dereferencing the result.
- Understanding this identity allows us to decompose array subscripting operations into pointer operations.
- Array and pointer operations can be the same, even though the declarations for arrays and pointers are different.

Pointers and Arrays

- This identity means that the subscript operation is equivalent to adding the index to the pointer expression and then dereferencing the result.
- Understanding this identity allows us to decompose array subscripting operations into pointer operations.
- Array and pointer operations can be the same, even though the declarations for arrays and pointers are different.

```
ai[3] = 42;    //this stores 42 w/array subscripting
*(ai+3)=42;    //same thing using pointer operations
*(3+ai)=42;    //addition is communitive
3[ai] = 42;    //this works!
```

All of the above works as well if pi were used instead!

Pointers and Arrays

- We have seen that the name of an array can be replaced with a pointer to the first element of the array.
- The only difference is that the name of an array is a constant and cannot be modified, whereas a pointer can be defined as a variable and therefore can be modified.
- The process of modifying a pointer variable is called **pointer arithmetic**.

Pointer Arithmetic

- Walk through the following in class:

```
int a[10];
int* p=a;       //initialize p to &a[0]
int* q=&a[2];   //initialize q to &a[2]
p = q;          //assign q to p

p = &a[5];      //p points to the 6th element &a[5]
p+=3;           //p now points to the 9th element &a[8]
p-=8;           //p now points to the 1st element &a[0]

p = &a[5];      //p points to the 6th element &a[5]
++p;            //p now points to the 7th element &a[6]
p++;            //p now points to the 8th element &a[7]
p = p + 2;      //p now points to the 10th element
   &a[9]
--p;            //p now points to the 9th element &a[8]
p--;            //p now points to the 8th element &a[7]
p = p - 2;      //p now points to the 6th element &a[5]
```

Pointer Arithmetic

- There are two key points in understanding pointer arithmetic.
- The first is that pointer variables can be modified whereas array names are constants and cannot be modified.
- The second is that pointer operations automatically take into account the size of the data pointed to, just like array subscripts do.
- This means that operations such as addition and subtraction are independent of the size of the data.
- When we add one to a pointer of some type, we point to the next element of that type.

Pointer Arithmetic

- Walk through the following in class:

```
int a[10];
int* p=a;       //define and initialize p to &a[0]
p = p + 1;      //add 1 to p; p==&a[1]
*p = *p + 1;    //(*p)=(*p)+1; add 1 to a[1]
*p = *(p + 1); //copy a[2] to a[1]
p+=1;           //add 1 to p; p==&a[2]
*p+=1;          //(*p)+=1; add 1 to a[2]
*(p+=1);        //add 1 to p; p==&a[3]
++p;            //add 1 to p; p==&a[4]
++*p;           //derefer p; add 1 to a[4]
*++p;           //add 1 to p; p==&a[5]
p++;            //add 1 to p; p==&a[6]
*p++;        //*(p++); rvalue==6; add 1 to p; p==&a[7]
(*p)++;      //dereference p; add 1 to a[7]
```

Arrays of Arrays

- Arrays can be formed from any type of data, even other arrays!
- When each element is an array, we define an array of arrays.
- With an array of arrays, each element is an array of some type.
- Arrays of arrays are sometimes called multidimensional arrays in C++.
- This is not strictly correct because each dimension represents a different type, rather than each dimension representing the same type.

Arrays of Arrays

- An array of arrays is defined just like an array of a fundamental type, except that the identifier is immediately followed by an additional pair of brackets ([]).
- The size of each element's array, called a subarray, is supplied within the second set of brackets as a literal, constant, or constant expression.

```
int array[3][2];
```

Arrays of Arrays

- To access elements of an array of arrays, we can use the subscript operator. To access the appropriate subarray, we follow the name of the array by an index in brackets. For example, array[0] accesses the first subarray. The value of this element is the first subarray of two integers. Its type is an array of integers (a pointer to an int).
- To access elements within a subarray, we follow the name of the array by the index of the subarray in brackets and then follow that by the index of the element within the subarray that we wish to access in brackets. For example, array[0][0] accesses the first integer in the first subarray.

Arrays of Arrays

- The name of an array of arrays also represents a pointer expression. By itself, it has a value equal to the address of the first element of the array. The type is a pointer to the first element of the array. For example, the type of array is int (*)[2] (a pointer to an array of two integers).

```
int array[3][2];
int (*p1)[2]; //define pointer of same type as array
p1 = array;   //assign pointer to point to array
```

Array of Arrays

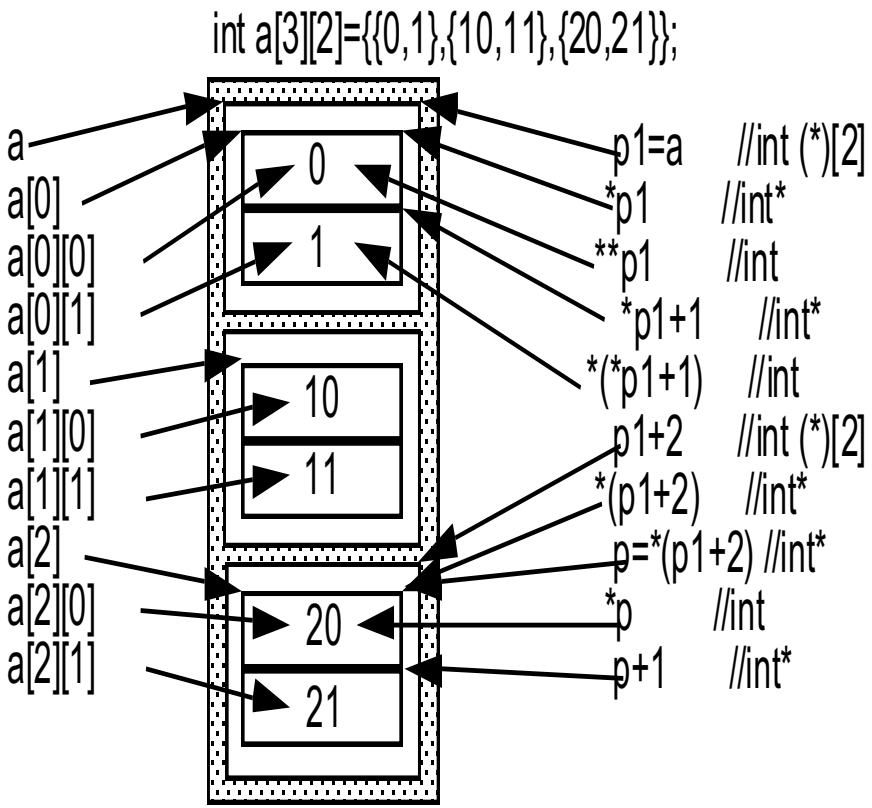
- Walk through the following in class:

```
int array[6][2];
int (*p1)[2];    //define pointer of same type as array
p1 = array;      //assign pointer to point to array

int *p;          //define ptr of same type as subarray
p = *p1;         //assign ptr to point to 1st subarray
p = array[0];    //this also points to 1st subarray and
p = *(array+0); //so does this because of our identity
p = *array;      //and so does this
```

- We can define and initialize a pointer to a subarray. The type of a subarray is a pointer to an element of the subarray. By defining a pointer of that type, we can use pointer arithmetic to access the subarray.

Array of Arrays



Arrays of Arrays

- If we were to print the value of the pointers p1 and p, their values would be the same even though they are different types.
- This is because the address of the first element of array is at the same address as the first element in the first subarray.
- However, when we add to or subtract from these two pointers, the results are significantly different.
- By adding one to pointer p1, we point to the next subarray of 2 integers.
- By adding one to pointer p, we point to the next int within the first subarray.