Arrays

Comp Sci 1570 Introduction to C++

Outline

Outline

1 Array basics
Declaration
Definition
Assignment

2 Multi-dimensional arrays
Declaration
Access
Assignment and access

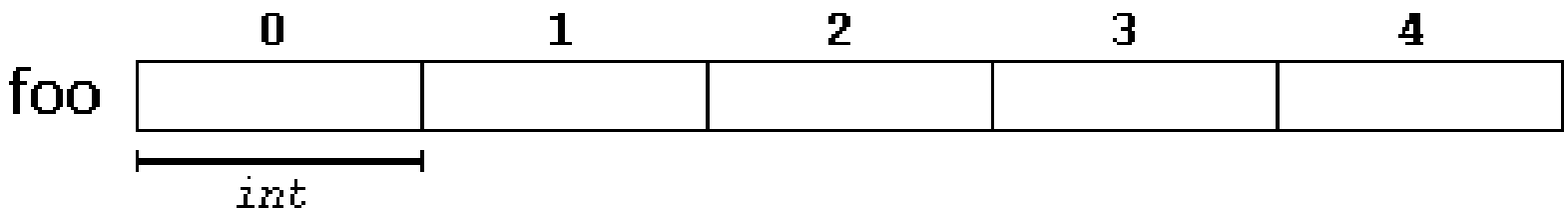
3 Arrays as parameters to functions
Multidimensional arrays to functions

4 Sorting arrays

5 Random stuff

Array declaration

- An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.
- That means that, for example, five values of type int can be declared as an array without having to declare 5 different variables (each with its own identifier).
- Instead, using an array, the five int values are stored in contiguous memory locations, and all five can be accessed using the same identifier, with the proper index.
- For example, an array containing 5 integer values of type int called foo could be represented as:




```
int foo [5];
```

Outline

1 Array basics
Declaration
Definition
Assignment

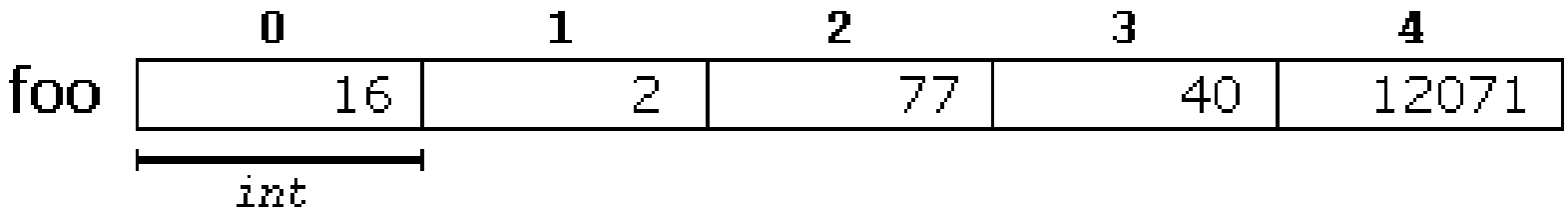
2 Multi-dimensional arrays
Declaration
Access
Assignment and access

3 Arrays as parameters to functions
Multidimensional arrays to functions

4 Sorting arrays

5 Random stuff

Definition

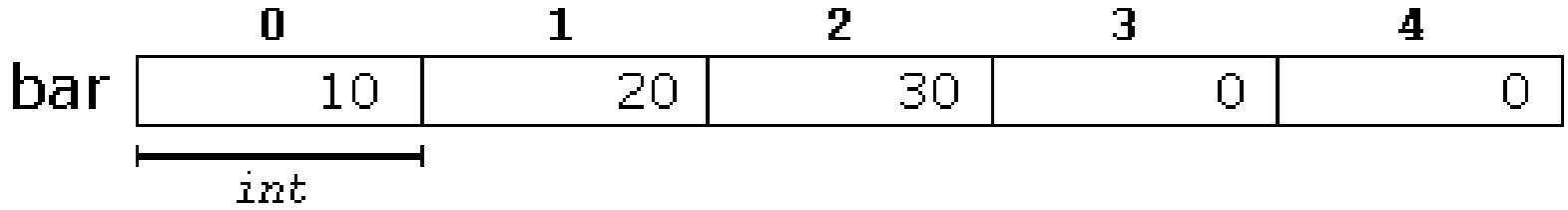



```
int foo [5] = { 16, 2, 77, 40, 12071 };
```

The number of values between braces shall not be greater than the number of elements in the array. For example, in the example above, foo was declared having 5 elements (as specified by the number enclosed in square brackets, []), and the braces contained exactly 5 values, one for each element.

Fill remainder with 0's

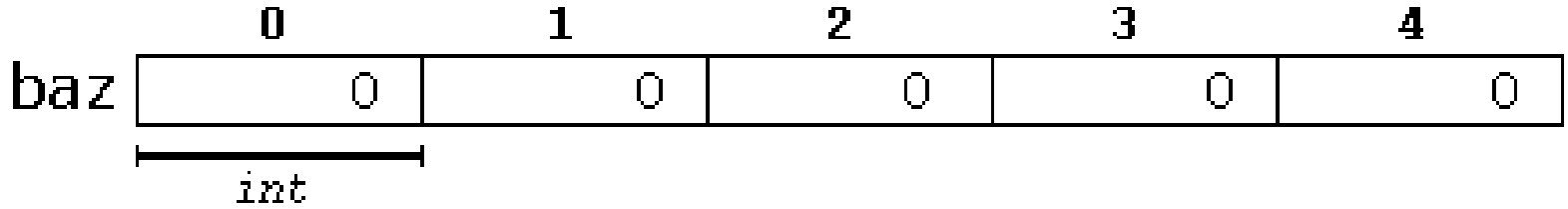
If declared with less, the remaining elements are set to their default values (which for fundamental types, means they are filled with zeroes). For example:




```
int bar [5] = { 10, 20, 30 };
```

Fill all with 0's

The initializer can even have no values, just the braces:




```
int baz [5] = { };
```

Outline

1 Array basics
Declaration
Definition
Assignment
2 Multi-dimensional arrays
Declaration
Access
Assignment and access
3 Arrays as parameters to functions
Multidimensional arrays to functions
4 Sorting arrays
5 Random stuff

Assignment to and from arrays

For example, the following statement stores the value 75 in the third element of foo:

```
foo [2] = 75;
```

```
x = foo[2];
```

and, for example, the above copies the value of the third element of foo to a variable called x:

Initialize to data size

It is possible to declare an array and initialize it with a sequence of elements, but no size specified:

```
double collection[] = { 1.0, 3.5, 0.5, 7.2 };
```

Outline

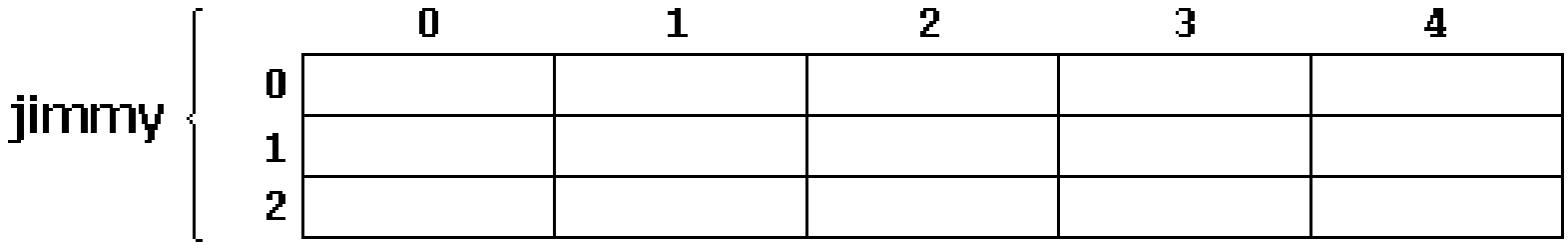
Indexing in multidimensional arrays

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Outline

Array basics
Declaration
Definition
Assignment
Multi-dimensional arrays
Declaration
Access
Assignment and access
Arrays as parameters to functions
Multidimensional arrays to functions
Sorting arrays
Random stuff

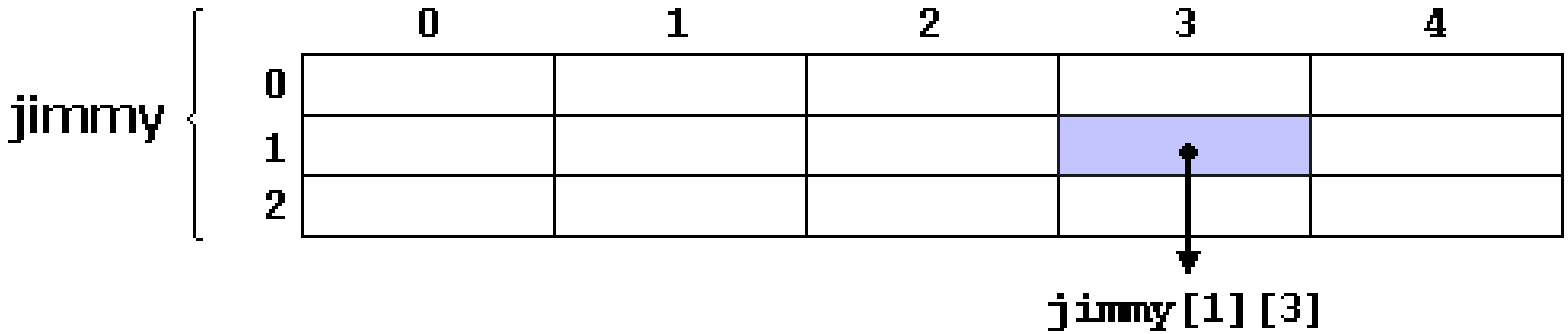
Declaration




```
int jimmy [3][5];
```

Outline

Access




```
cout << jimmy[1][3];
```

Outline

Array basics
Declaration
Definition
Assignment
Multi-dimensional arrays
Declaration
Access
Assignment and access
Arrays as parameters to functions
Multidimensional arrays to functions
Sorting arrays
Random stuff

Assignment and access

jimmy

	0	1	2	3	4
0	1	2	3	4	5
1	2	4	6	8	10
2	3	6	9	12	15

```
const int WIDTH 5
const int HEIGHT 3
int jimmy [HEIGHT][WIDTH];
int n,m;

int main ()
{
   for (n=0; n < HEIGHT; n++)
      for (m=0; m < WIDTH; m++)
      {
         jimmy[n][m]=(n+1)*(m+1);
      }
}
```

Multidimensional array options

Two-dimensional arrays with initializer lists can omit (**only**) the leftmost length specification:

```
int array[][5] =
{
{ 1, 2, 3, 4, 5 },
{ 6, 7, 8, 9, 10 },
{ 11, 12, 13, 14, 15 }
};
```

Multidimensional array options

NOT OK. The compiler can do the math to figure out what the array length is. However, the following is not allowed:

```
int array[][] =
{
{ 1, 2, 3, 4 },
{ 5, 6, 7, 8 }
};
```

Outline

1 Array basics
Declaration
Definition
Assignment

2 Multi-dimensional arrays
Declaration
Access
Assignment and access

3 Arrays as parameters to functions
Multidimensional arrays to functions

4 Sorting arrays

5 Random stuff

Arrays as parameters to functions

- In C++, we don't want to pass the entire block of memory represented by an array to a function directly as an argument.
- But what can be passed instead is its address.
- In practice, this has almost the same effect, and it is a much faster and more efficient operation.

```
void procedure(int arg[])

int main()
{
  int myarray [40];
  procedure(myarray);

...
```

Outline

1 Array basics
Declaration
Definition
Assignment

2 Multi-dimensional arrays
Declaration
Access
Assignment and access

3 Arrays as parameters to functions
Multidimensional arrays to functions

4 Sorting arrays

5 Random stuff

Multidimensional arrays to functions

Notice that the first brackets [] are left empty, while the following ones specify sizes for their respective dimensions. Including the other dimensions is necessary in order for the compiler to be able to determine the depth of each additional dimension.

```
void procedure (int myarray[][3][4]);
```

Outline

See code demos

Bubble, selection, insertion

Outline

More tips on PRNG

Please see code from day on random number generators, since it has been updated with some new tricks.