CS106B Chris Piech
Winter 2016 March 12th, 2016

Pointers

In C++ you can "dynamically allocate" memory. That means that at any point in your program execution you can specially request space to store new variables. Unlike variables declared on the stack, a variable stay allocated until the programmer explicitly releases or "frees" it.

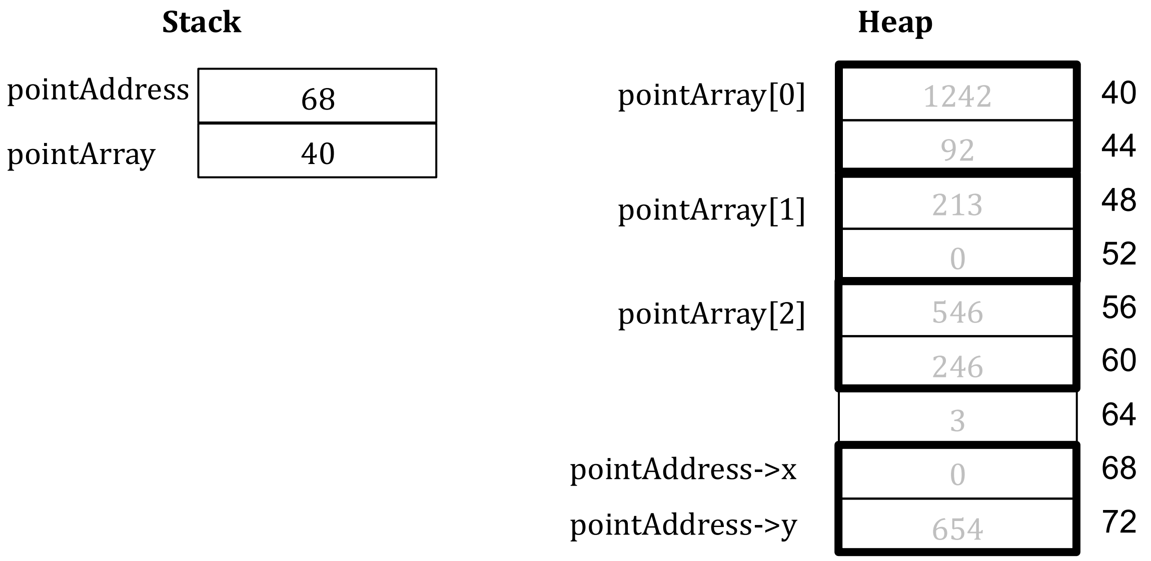
The mechanism for accessing such memory is through pointers: special variables that store memory addresses. When one requests dynamically allocated memory, a pointer is returned.

Dynamic Allocation

There are two ways to request memory: you can ask for a single variable or you could ask for an array of variables:

```
Point * pointAddress = new Point;     // allocates a single "point"
Point * pointArray = new Point[3];    // allocates 3 points.
```

And here is a picture of what happens in memory. pointAddress stores the address of its pointee:



In this simple memory picture, each bucket of memory on the heap has an address (valued 40 through 72). Each allocated point gets two buckets (for the x and y components). The pointers pointAddress and pointArray are variables that live on the stack and hold addresses of memory on the heap.

Pointer Types

We have just introduced a new variable type. The “pointer”. It is a stack variable that stores an address. You can tell a variable is a pointer if its type ends with a *.

Type	**Meaning**
int *	Address of an int
Point *	Address of a point
Set<int> *	Address of a Set<int>

Accessing Pointees

Pointees is the name of the variables that pointers store the address of. We would like to be able to get and set their values.

Single variable dynamic allocation:
If a class or struct was dyammically allocated, we can apply the -> operator to its pointer to access the pointee’s members values or to call methods on the pointee.

```
pointAddres->x = 5;          // makes the pointee x = 5
cout << pointAddress->y;     // gets the pointeee y
```

Array dynamic allocation:
If an array of pointees were created, you can get the ith value using bracket notation.

```
pointArray[0].x = 5          // sets the x value of the first element
cout << pointArray[1].y;     // gets the y value of the second element
```

Assignment

You can use the = operator to copy a pointers address. Then two pointers point to the same pointee. This is called “sharing”.

```
Point * a = new Point;                 // allocates a single “point”
Point * list = new Point[3];           // allocates 3 points.
Point * b = a;
```

Delete

When you use the new keyword to allocate memory, that memory persists until you tell the computer it can re-use it (or your program exits). To free the memory, use the keyword delete:

```
delete pointAddress;         // how to delete a single variable
delete[] pointArray;         // how to delete an array.
```

Other Operators

There are a few other special operators that you can perform related to pointers. We don't emphasize them in CS106B and you **won't need to know them for the final**. I included them here for full measure.

Pointer Operator	Meaning
&	Get the address of a variable
*	Get the pointee on the other side of the pointer.

Important: The * operator is not to be confused with the **much more common** use of * as part of a variable type name.

```
int stackInt = 5;
Point * a = &stackInt;        // a points to the address of stackInt
cout << *a                    // prints 5
```