Object Oriented Concepts

Pointers in C++

Introduction

- A *pointer* is a variable that holds a memory address.
- Why pointers?
    - Provide the means by which functions can modify the arguments of the calling function.

      Note: pass by reference was not available in C.
    - Pointers support dynamic allocation of memory.
    - Can improve the efficiency of some routines.

Declaring a Pointer

Syntax:

type *name;

Examples:

```
int *m;
char *array;
double *temp;

int **matrix;
```

Memory

	Address	Contents
m	0x3267A1B0	0x32
	0x3267A1B1	67
	0x3267A1B2	A1
	0x3267A1B3	B8
	0x3267A1B4	
	0x3267A1B5	
	0x3267A1B6	
	0x3267A1B7	
	0x3267A1B8	
	0x3267A1B9	
	0x3267A1BA	
	0x3267A1BB	

Pointer Operators

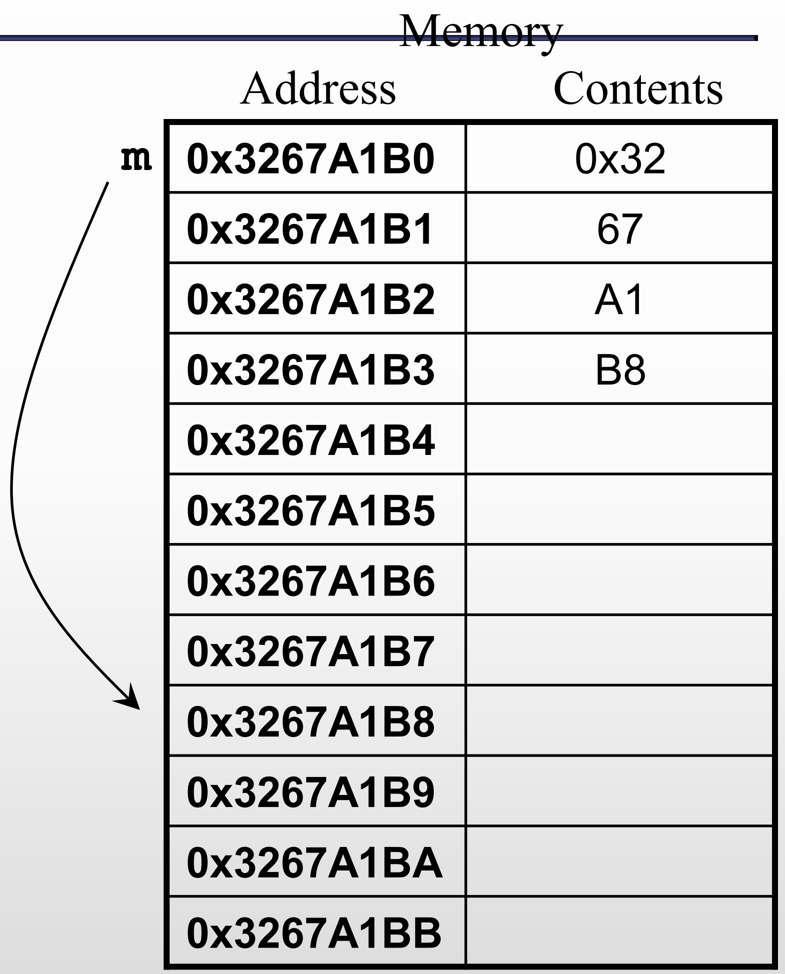
- Once a pointer is declared, the operator * can be used to obtain the value located at the address that is held by the pointer.
  - Eg : cout<< *m ; // will print the value at the location pointed by m.
- The operator & can be used to obtain the memory address of an operand.
  - Eg : cout << &a; // will print the memory address of a.
- Both the operators, * and & are unary operators. That is, it uses only one operand.
-

Pointer Operators

Declaring a pointer

- Example:

```
int *m;
```



Address	Contents
0x3267A1B0	0x32
0x3267A1B1	67
0x3267A1B2	A1
0x3267A1B3	B8
0x3267A1B4	
0x3267A1B5	
0x3267A1B6	
0x3267A1B7	
0x3267A1B8	
0x3267A1B9	
0x3267A1BA	
0x3267A1BB	

Note :

Initially the pointer will point at (will store the memory address of) any location in the memory.

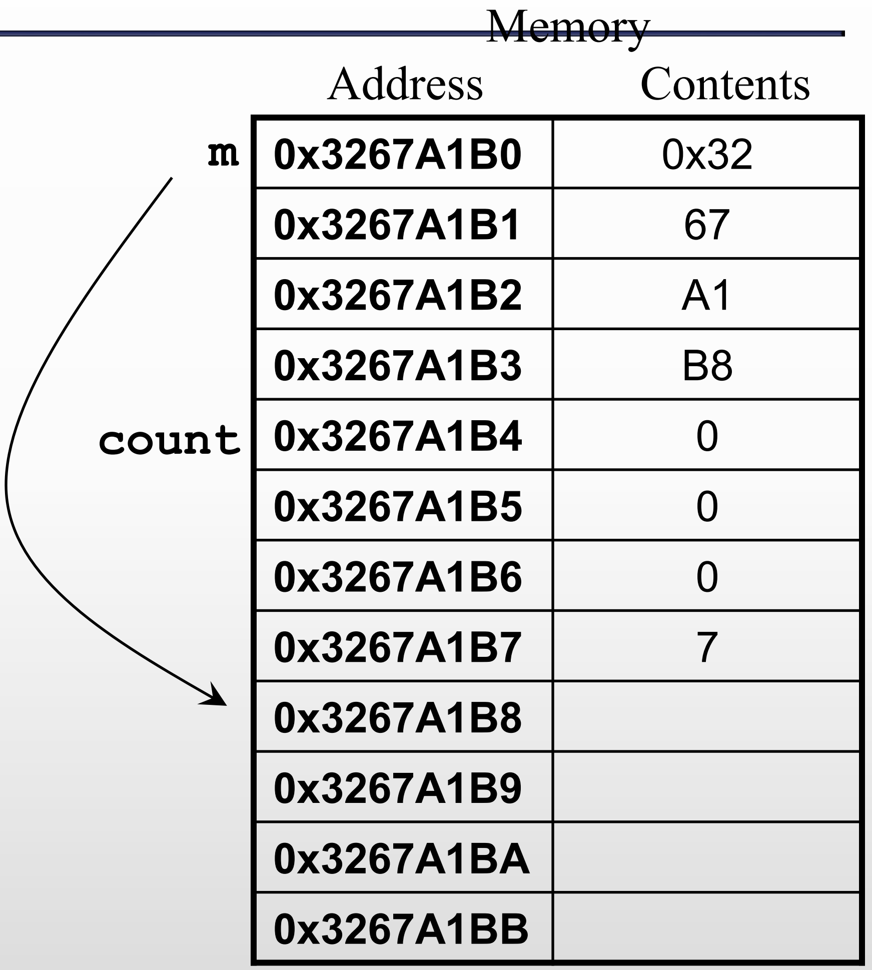
Pointer Operators

- Example:

```
int *m;

int count = 7;
```

// count is a variable that stores an integer.



	Memory Address	Contents
m	0x3267A1B0	0x32
	0x3267A1B1	67
	0x3267A1B2	A1
	0x3267A1B3	B8
count	0x3267A1B4	0
	0x3267A1B5	0
	0x3267A1B6	0
	0x3267A1B7	7
	0x3267A1B8	
	0x3267A1B9	
	0x3267A1BA	
	0x3267A1BB	

Pointer Operators

- Example:

```
int *m;
int count = 7;

m = &count;
```

// Assigns the memory address of `count` to the pointer `m`

- The value of `m` is **0x3267A1B4**
- The value of `*m` is **7**

Memory

	Address	Contents
m	0x3267A1B0	0x32
	0x3267A1B1	67
	0x3267A1B2	A1
	0x3267A1B3	B4
count	0x3267A1B4	0
	0x3267A1B5	0
	0x3267A1B6	0
	0x3267A1B7	7
	0x3267A1B8	
	0x3267A1B9	
	0x3267A1BA	
	0x3267A1BB	

Dynamic Memory Allocation

- Allows a program to obtain memory at runtime.
- Dynamic Allocation Functions in C
  - **`malloc()`** is used to allocate memory.
  - **`free()`** is used to release memory.
- Dynamic Allocation Functions in C++
  - **`new`** is used to allocate memory.
  - **`delete`** is used to release memory.

C++

- Allocate Memory for an integer variable

```
int * p  ;
p = new int;
```

```
// int *p = new int;
```

- Deallocate memory

```
delete p;
```

Variable Declarations

```
//Declaring an integer Variable and initializing to 87
int *p;
p = new int (87); //Initializes to 87. That is *p is 87
if (!p){   // check whether memory was allocated
   cout << "Cannot Allocate Memory" << endl;
   exit(1);
} //if
delete p;

//Declaring an integer array of 10 elements
int *k;
k = new int [10]; //an array of 10 integers
if (!k){
   cout << "Cannot Allocate Memory" << endl;
   exit(1);
} //if
delete
```