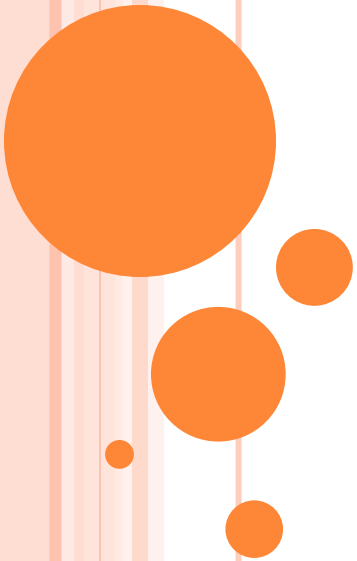
Accessing Arrays and Dynamic Memory Allocation Lecture



OOP/Computer Programming

Instructor: Dr. Danish Shehzad

Recap

- Pointers
- Referencing and Dereferencing
- Arithmetic on Pointers
- Relationship between Pointers and Arrays
- Accessing array elements through pointers
- Accessing 2 D Arrays using pointers
- Passing Arguments to Functions by Reference using Pointers
- Constant Pointers
- Sizeof
- Arrays of Pointers

Today's Lecture

- Dynamic Memory Allocation
- Dangling Pointer
- Memory Leak
- Casting Pointers
- Double Pointers

Dynamic Memory Allocation

- *Used when space requirements are unknown at compile time.*
- *Most of the time the amount of space required is unknown at compile time.*
  - *For example consider the library database example, what about 201th book information.*
  - *Using static memory allocation it is impossible.*

- *Dynamic Memory Allocation (DMA):-*
  - *Allows to set array size dynamically during run time rather than at compile time. This helps when program does not know in advance about the number of values to be stored.*
  - *With Dynamic memory allocation we can allocate/delete memory (elements of an array) at runtime or execution time.*

Diferences between Static and Dynamic Memory Allocation

- Dynamically allocated memory is kept on the memory heap (also known as the free store)
- Dynamically allocated memory can't have a "name", it must be referred to

- The ***new*** operator is used to dynamically allocate memory

Pointing to Memory Allocated at Run Time

- *int *ptr;*
- *ptr = new int;*
- **ptr = 7;*
- *int *a;*

- *a = new int[3];*
  - **a = 300;*
  - **(a+1) = 301;*
  - **(a+2) = 302;*

Name Table

Name	Type	Contents	Address
ptr	int pointer	0x3D3B38	0x22FF68
a	int array pointer	0x3D3BA0	0x22FF64

Memory Heap (Free Store)

	0x_____0	0x_____4	0x_____8	0x_____C
0 0x3D3B3_			7	
0 0x3D3B4_				
0 0x3D3B5_				
0 0x3D3B6_				
0 0x3D3B7_				
0 0x3D3B8_				
0 0x3D3B9_				
0 0x3D3BA_	300	301	302	
0 0x3D3BB_				
0 0x3D3BC_				
0 0x3D3BD_				

Returning Memory to the Heap

- How Big is the Heap?
  - Most applications request memory from the heap when they are running
  - It is possible to run out of memory (you may even have gotten a message like "Running Low On Virtual Memory")
  - So, it is important to return memory to the heap when you no longer need it

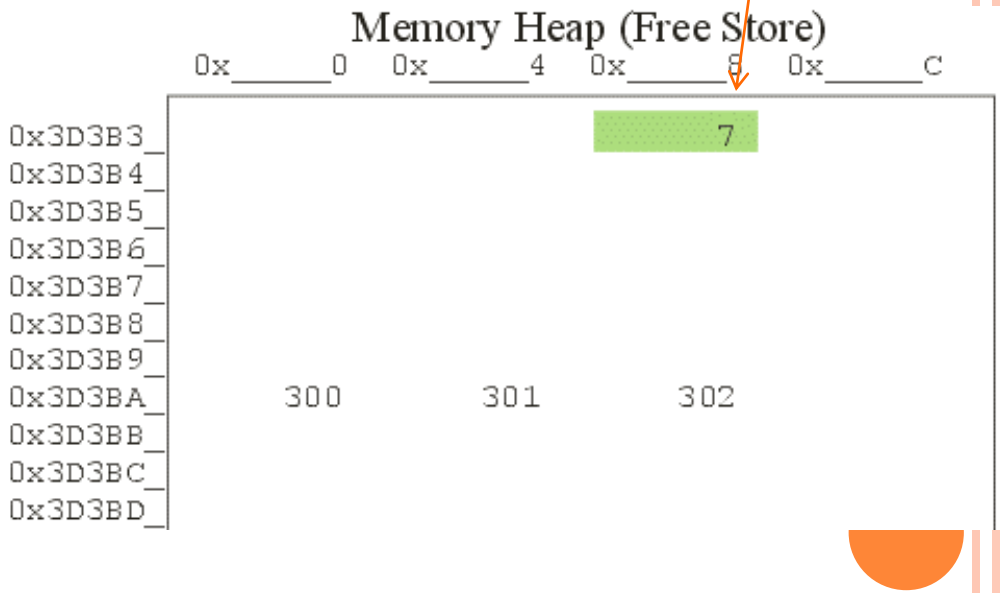
Returning Memory to the Heap

- The Opposite of **new**:
  - In C++: The **delete** operator in C++

- delete ptr;

Name Table

Name	Type	Contents	Address
ptr	int pointer	0x3D3B38	0x22FF68
a	int array pointer	0x3D3BA0	0x22FF64



Dangling Pointers

- The delete operator does not delete the pointer, it takes the memory being pointed to and returns it to the heap
- It does not even change the contents of the pointer
- Since the memory being pointed to is no longer available (and may even be given to another application), such a pointer is said to be dangling
- A pointer that is pointing to deallocated memory is called a **dangling pointer**

- ptr = NULL;

Name Table

Name	Type	Contents	Address
ptr	int pointer	0x000000	0x22FF68
a	int array pointer	0x3D3BA0	0x22FF64

Returning Memory to the Heap

- Remember:
  - Return memory to the heap before undangling the pointer

- What's Wrong with the Following:
  - ptr = NULL;
  - delete ptr;

Returning Memory to the Heap

- For Arrays
  - delete[ ] a;
  - a = NULL;

Memory Leaks

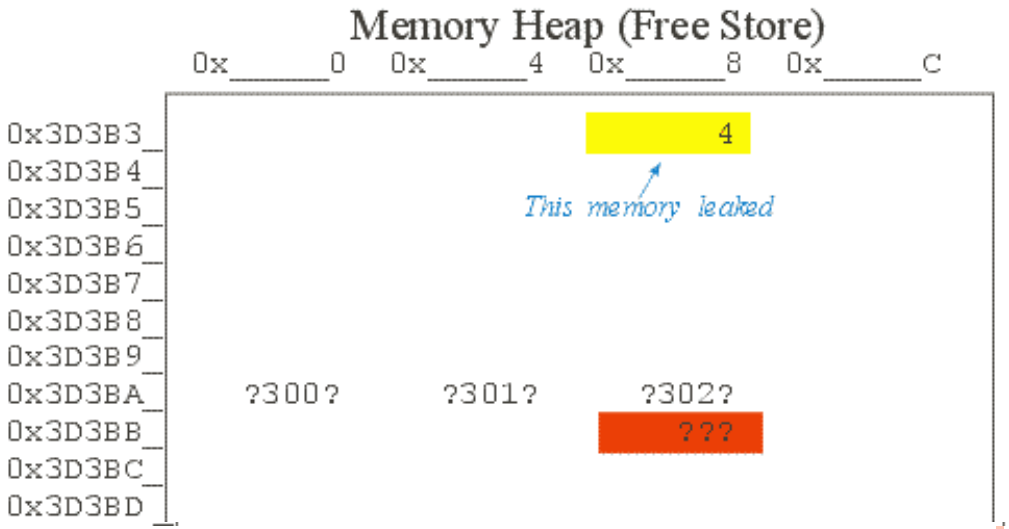
- Memory *leaks* when it is allocated from the heap using the **new** operator but not returned to the heap using the **delete** operator

- *int *otherptr;*
- *otherptr = new int;*

- **otherptr = 4;*
- *otherptr = new int;*

Name Table

Name	Type	Contents	Address
ptr	int pointer	0x000000	0x22FF68
a	int array pointer	0x000000	0x22FF64
otherptr	int ptr	0x3D3BB8	0x22FF60



Casting pointers

- Pointers have types, so you can't just do

```
int *pi;      double *pd;
pd = pi;
```

(**Error**)

- Even though they are both just integers, C won't allow

Casting pointers

- C++ will let you change the type of a pointer with an **explicit cast**

```
int *pi; double *pd;
pd = (double*) pi;
```

Void Pointers

- A void pointer is a pointer that has no associated data type with it. A void pointer can hold address of any type and can be typcasted to any type.

- int a = 10;
- char b = 'x';
- void *p ;

- P=&a; // void pointer holds address of int 'a‘

- p = &b; // void pointer holds address of char 'b'

Double Pointers

- We already know that a pointer points to a location in memory and thus used to store the address of variables. So, when we define a pointer to pointer.
- The first pointer is used to store the address of the variable. And the second pointer is used to store the address of the first pointer. That is why they are also known as double pointers.

Today's Lecture

- Dynamic Memory Allocation
- Dangling Pointer
- Memory Leak
- Casting Pointers
- Double Pointers