C++

LAUNCHPAD

CODING BLOCKS

Lecture-10

Miscellaneous Topics

- Dynamic Allocation
- Space Time Complexity Analysis

Deepak Aggarwal

Any doubts?

Pointers Recap

Recap

1. How to define pointers?
2. Address of Operator?
3. Dereference operator?
4. Arithmetic Operators on pointers?
5. Arrays & Pointers
6. Reference Variables
7. Pass by reference
8. Returning pointers or References from functions

Address typecasting

Dynamic Memory Allocation!

Allocating Memory

There are two ways that memory gets allocated for data storage:

- Compile Time (or static) Allocation
  - Memory for named variables is allocated by the compiler
  - Exact size and type of storage must be known at compile time
  - For standard array declarations, this is why the size has to be constant
- Dynamic Memory Allocation
  - Memory allocated "on the fly" during run time
  - dynamically allocated space usually placed in a program segment known as the heap or the free store
  - Exact amount of space or number of items does not have to be known by the compiler in advance.
  - For dynamic memory allocation, pointers are crucial

Dynamic Memory Allocation

- We can dynamically allocate space while the program is running but we cannot create new variable names "on the fly"
- For this reason, dynamic allocation requires two steps
  1. Creating the dynamic space
  2. Storing its address in a pointer
- To dynamically allocate memory in C++, we use new operator
- De-allocation:
  - De-allocation is the "clean-up" of space being used by variable

De-allocation

- De-allocation is the “clean up” of space being used by variables or other data storage
- Compile time variable are automatically deallocated based on their know scope
- It is the programmer’s job to deallocate dynamically created memory
- To de-allocate dynamic memory we use delete operator

new operator

- To allocate space dynamically, use the unary operator new, followed by the type being allocated.
  - new int; // dynamically allocates an int
  - new double; // dynamically allocates a double
- If creating an array dynamically, use the same form, but put brackets with a size after the type:
  - new int[40]; /allocates an array of 40 ints
  - new double[size]; // allocates an array of size double
    // doubles
- These statements above are not very useful by themselves, because allocation space have no names.

new operator contd..

```
int * p;        // declare a pointer p
 p = new int;    // dynamically allocate an int and load address into p

 double * d;     // declare a pointer d
 d = new double; // dynamically allocate a double and load address into d

 // we can also do these in single line statements
 int x = 40;
 int * list = new int[x];
 float * numbers = new float[x+10];
```

delete operator

- To de-allocate memory that was created with new, we use the unary operator delete. The one operand should be a pointer that stores the address of the space to be deallocated:

```
int * ptr = new int;        // dynamically created int
// ...
delete ptr;                 // deletes the space that ptr points to
```

Note that the pointer ptr still exists in this example. That's a named variable subject to scope and extent determined at compile time. It can be reused:

- To deallocate a dynamic array, use this form:

```
int * list = new int[40];   // dynamic array

delete [] list;             // deallocates the array
list = 0;                                   // reset list to null pointer
```

After deallocating space, it's always a good idea to reset the pointer to null unless you are pointing it at another valid target right away.

Lets see an example!

constant variables

#define

Inline Functions?

Default Value of Arguments?

cin.getline()

Global Variables?

Static Local Variable?

Order Complexity Analysis

Amount of time/space taken by the algorithm to run as a function of the input size

Experimental Analysis

- Selection Sort vs Merge Sort

Theoretical Analysis

- Bubble Sort
- Binary Search
- Factorial
- Polynomial Evaluation

Your turn

- Insertion sort
- Fibonacci

Complexity Analysis Examples

```
for (i=0; i<=n-1; i++){
   for (j=i+1; j<=k; j++){
      constant number of operations.
   }
}
```

Complexity Analysis Examples

```
for (i=0; i<=n-1; i++){
   for (j=i+1; j<=n; j++){
      constant number of operations.
   }
}
```

Complexity Analysis Examples

```
for (i=0; i<=n-1; ){
   for (j = 0; j<k; j++){
       constant number of operations.
   }
   i = i + j;
}
```

What is space complexity?

What in case of recursion?

HW - Go through the assignments

C++

LAUNCHPAD

CODING BLOCKS

Thank You!

Deepak Aggarwal