

August 18, 2015

# Launchpad

Lecture - 12

Mixed Topics

Aman Bahl

---



# Some more stuff related to Functions

# Inline Functions

- I. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.
- II. To inline a function, place the keyword `inline` before the function name and define the function before any calls are made to the function
- III. The compiler can ignore the inline qualifier in case defined function is more than a line.

# Default Parameter

- I. A default parameter is a function parameter that has **a default value** provided to it.
- II. If the user does not supply a value for this parameter, the default value will be used. If the user does supply a value for the default parameter, the user-supplied value is used

# Scope and Duration of Variables

- I. When discussing variables, it's useful to separate out the concepts of **scope** and **duration**.
- II. A variable's scope determines where a variable is accessible
- III. A variable's duration determines when it is created and destroyed

# Local Variables

- I. Variables defined inside a block are called local variables.
- II. Local variables have automatic duration, which means they are created when the block they are part of is entered, and destroyed when the block they are part of is exited.
- III. Local variables have block scope (also called local scope), which means they are visible only within the block that they are defined in.

# Global Variables

- I. Variables declared outside of a block are called global variables
- II. Global variables have static duration, which means they are created when the program starts and are destroyed when it ends
- III. Global variables have global scope (also called “global namespace scope” or “file scope”), which means they are visible until the end of the file in which they are declared
- IV. By convention, global variables are declared at the top of a file, below the includes, but above any code.

# Local Static Variables

- I. Using the static keyword on local variables changes them from automatic duration to static duration.
- II. A static duration variable (also called a “static variable”) is one that retains its value even after the scope in which it has been created has been exited!
- III. Static duration variables are only created (and initialized) once, and then they are persisted throughout the life of the program.



# Function Overloading

- I. You can have multiple definitions for the same function name in the same scope
- II. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list.
- III. You can not overload function declarations that differ only by return type.

# More about Pointers!



Pointer to a Pointer ( type `** var` )

# Dynamic Memory Allocation!

# Allocating Memory

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

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

# Dynamic Memory Allocation

- I. We can dynamically allocate space while the program is running but we cannot create new variable names “on the fly”
- II. For this reason, dynamic allocation requires two steps
  - 1. Creating the dynamic space
  - 2. Storing its address in a pointer
- III. To dynamically allocate memory in C++, we use new operator

# De-allocation

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

# new Operator

- I. To allocate space dynamically, use the unary operator `new`, followed by the type being allocated.
  - I. `new int;` // dynamically allocates an int
  - II. `new double;` // dynamically allocates a double
- II. If creating an array dynamically, use the same form, but put brackets with a size after the type:
  - I. `new int[40];` /allocates an array of 40 ints
  - II. `new double[size];` // allocates an array of size double  
// doubles
- III. 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

- I. 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:**

- I. 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 some input/output problems.

# References in C++

# Reference Variable

- I. A reference variable is an alias, that is, another name for an already existing variable/memory instance.
- II. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.
- III. The reference variable once defined to refer to a variable can't be changed to point to other variable.

# Defining Reference Variable

- I. Reference variables are defined by using & symbol in the definition.
- II. Since they do not have any storage of their own and are just another name for the existing storage, they need to be initialized before using them.
- III. For e.g. `int x; int & y = x`. The existing memory X will now also have another name Y.

# Reference Variable VS Pointer Variable

- I. You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.
- II. Once a reference is initialized to an object, it cannot be changed to refer to another object. Pointers can be pointed to another object at any time.
- III. A reference must be initialized when it is created. Pointers can be initialized at any time.

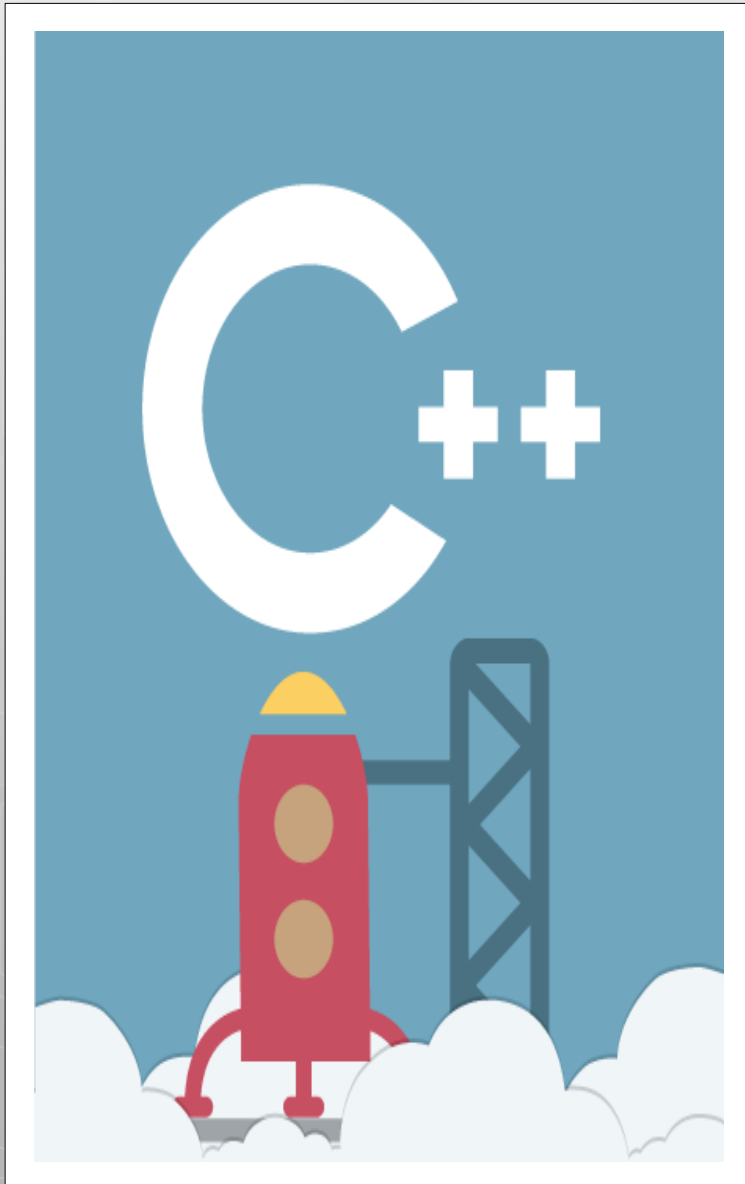
# References as Function Parameters!



# References as Return Value from Functions.

# What is next class about?

## I. Structures.



# Thank You!

Aman Bahl

[aman.or.b@gmail.com](mailto:aman.or.b@gmail.com)

+91-9908124628

---