CIS 22A – Lecture 16

Manish Goel

Local Variables

- Variables defined inside a function are *local* to that function. They are hidden from the statements in other functions, which normally cannot access them.
- Because the variables defined in a function are hidden, other functions may have separate, distinct variables with the same name.
- A function's local variables exist only while the function is executing. This is known as the *lifetime* of a local variable.
- Local variables are not automatically initialized. They must be initialized by programmer.
- When the function begins, its local variables and its parameter variables are created in memory, and when the function ends, the local variables and parameter variables are destroyed.
- This means that any value stored in a local variable is lost between calls to the function in which the variable is declared.

Global Variables and Constants

- A global variable is any variable defined outside all the functions in a program.
- The scope of a global variable is the portion of the program from the variable definition to the end.
- This means that a global variable can be accessed by *all* functions that are defined after the global variable is defined.
- You should avoid using global variables because they make programs difficult to debug.
- Any global that you create should be *global constants*.
- Global variables (not constants) are automatically initialized to `0` (numeric) or `NULL` (character) when the variable is defined.

Static Local Variables

- Local variables only exist while the function is executing. When the function terminates, the contents of local variables are lost.

- `static` local variables retain their contents between function calls or invocations.

- `static` local variables are defined and initialized only the first time the function is executed. `0` is the default initialization value.

- `static` local variables can also be used for running totals or counters

Default Arguments

- A Default argument is an argument that is passed automatically to a parameter if the argument is missing on the function call.
- Must be a constant declared in prototype:

```
void evenOrOdd(int = 0);
```

- Can be declared in header if no prototype
- Multi-parameter functions may have default arguments for some or all of them:

```
int getSum(int, int=0, int=0);
```

- If not all parameters to a function have default values, the defaultless ones are declared first in the parameter list:

```
int getSum(int, int=0, int=0);// OK
int getSum(int, int=0, int);  // NO
```

- When an argument is omitted from a function call, all arguments after it must also be omitted:

```
sum = getSum(num1, num2);     // OK
sum = getSum(num1, , num3);   // NO
```

Pass-by-Reference

- Allows a function to work with the original argument from the function call, not a copy of the argument
- Allows the function to modify values stored in the calling environment
- Provides a way for the function to 'return' more than one value
- A reference variable is an alias for another variable
- Defined with an ampersand (`&`)

```
void getDimensions(int&, int&);
```

- Space between type and `&` is unimportant
- Must use `&` in both prototype and header
- Changes to a reference variable are made to the variable it refers to
- Use reference variables to implement passing parameters *by reference*

Overloading Functions

- Overloaded functions have the same name but different parameter lists

- Can be used to create functions that perform the same task but take different parameter types or different number of parameters

- Compiler will determine which version of function to call by argument and parameter lists

Using these overloaded functions,

```
void getDimensions(int);             // 1
void getDimensions(int, int);        // 2
void getDimensions(int, double);     // 3
void getDimensions(double, double);// 4
```

the compiler will use them as follows:

```
int length, width;
double base, height;
getDimensions(length);             // 1
getDimensions(length, width);      // 2
getDimensions(length, height);     // 3
getDimensions(height, base);       // 4
```

The `exit()` Function

- Terminates the execution of a program
- Can be called from any function
- Passes an `int` value to OS to indicate status of program termination
- Usually used for abnormal termination of program
- Requires `cstdlib` header file
- Example:

```
exit(0);
```

- The `cstdlib` header defines two constants that are commonly passed, to indicate success or failure:

```
exit(EXIT_SUCCESS);
exit(EXIT_FAILURE);
```