C Programming

Functions

Storage Classes

In most programming languages, there are multiple ways for using data within variables with functions. In C, there are 4 types of storage available.

1. Auto variables

- All variables declared/created inside any function are auto by default.
- auto is a reserved word in C.
- When a function ends, all data inside an auto variable is freed and no longer accessible.
- auto variables are also known as (aka) **LOCAL** variables. Local variables are only accessible within the function they are declared.

For example:

```
int main()
{
    .....
    .....
    my_function();

    .....

} // end main()


void my_fuction()
{
    auto int num;
    auto float average;
    auto char my_char = 'a';

    // code to follow
```

```
    // ..
    // ..

} // end my_function
```

2. Static variables

- static is a reserved word in C.
- static variables are also **LOCAL** variables, i.e., they are declared and accessible only within the function they are declared.
- Unlike auto variables, static variables do **not** have their memory freed after the function ends, i.e., the static variable retains its data, and that data can be used in subsequent calls to the function.

```
/*
Program showing the use and differences between auto and static
variables
*/

#include <stdio.h>

#define NUM 10

// Function signature
void fxn(void);

int main()
{
   int i;

   // Call the function 10 times
   for(i = 0; i < NUM; i++)
   {
       fxn();

   } // end for

} // end main
```

```c
/* This function will use both auto and static variables and
demonstrate the differences between both
*/
void fxn(void)
{
   int auto_var = 0;
   static int static_var = 0;

   //increment the static variable
   static_var++;

   //increment the auto variable
   auto_var++;

   printf("\nauto_var is %d, static_var is %d", auto_var, static_var);

} // end fxn()
```

Repl 16.1: https://replit.com/@michaelTUDublin/161-Static-variables

3. Register variables

- register is a reserved word in C.
- register variables are commonly used when memory, i.e., RAM is expensive.
- register variables have their memory placed inside the CPU (Central Processing Unit) and **not** RAM
- the OS makes a best attempt to allocate memory in the CPU for register variables

A common place where a register variable is used is as an index variable with a loop

e.g.,

```c
int main()
{
  register int i;
  register int j;
  .....
  .....
   return 0;
} // end main()
```

4. Extern variables

- extern is a reserved word in C.
- extern variables are also known as "Global variables".
- extern variables are declared outside any function and therefore are visible and can be used by every function
- If the global variable is not found inside your program, the OS will search for the variable inside the Header (.h) files. If found in any header file, this global variable will be the variable that is used. If not found, a compiler error will result.

e.g.,

```
/*
Program that uses an extern variable
*/

#include <stdio.h>

// Function signature
void fxn(void);

// Global variable - avoid when possible
int num = 0;

int main()
{
   printf("Inside main function\n");

   // Call fxn()
   fxn();

   printf("\nBack in main function\n");
   printf("\nnum is %d", num);

   return 0;

} // end main()
```

```
//Implement function fxn()
void fxn(void)
{
   //the variable num is never created here. It tells the compiler to
find it globally
   extern int num;

   printf("\nInside fxn function\n");

   //increment num
   num++;

} // end fxn()
```

Repl 16.2: https://replit.com/@michaelTUDublin/162-Extern-variables

Additional built-in functions

C provides many different built-in functions. One example are Mathematical functions as follows:

- cos(x);
- sin(x);
- tan(x);
- each of the above the parameter type and return type is a *double*

In the above, you need to include the header file: **#include <math.h>**

It also provides built-in functions including:

- log(x); // parameter is a *double* and returns a *double*
- pow(x,y); // calculates x to the power of y. Returns a *double*

- rand(); // returns a random integer number
- srand(x); // this also returns a random integer. The parameter is a seed-value, which is any positive number

both rand() and srand(x) require the header file: **#include <stdlib.h>**