CS 240 Programming in C

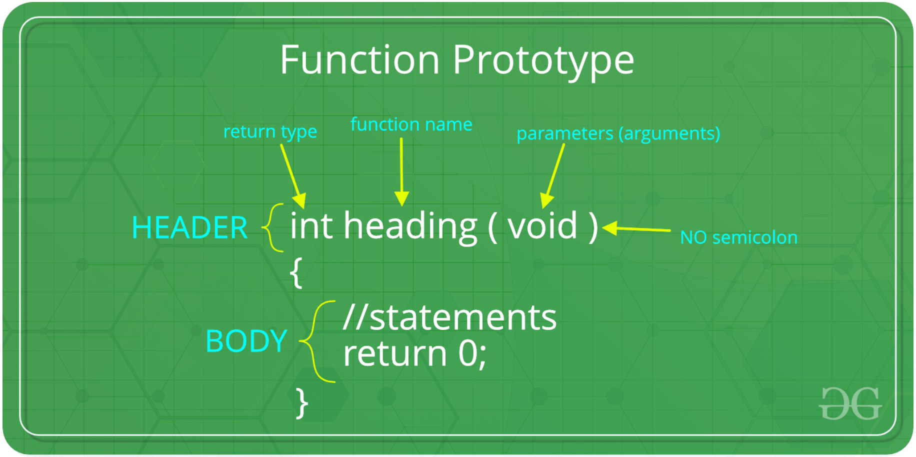
Functions

September 20, 2022

Functions

- Functions provide a convenient way to encapsulate some computation, which can then be used without worrying about its implementation
- A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body. A function prototype gives information to the compiler that the function may later be used in the program.

Functions


Function Prototype

```
int heading ( void )
{
    //statements
    return 0;
}
```

Link: https://www.geeksforgeeks.org/c-function-argument-return-values/

Function Declaration and Definition

- Function declaration
  `int power(int m, int n);`
  - Function declaration says that power is a function that expects two int arguments and returns an int
- Function definition

```
returnType funcName(parameter declarations, if any) {
  declarations

  statements
}
```

Parameters vs Arguments

- Formal parameter
    - The names given to the arguments in the function definition
    - Often referred to as parameters
- Actual parameter
    - The names supplied in a function call
    - Often referred to as arguments

Call by Value vs Call by Reference

- In C all function arguments are passed "by value"
- The called function is given the values of its arguments in temporary variables, distinct from the originals
- This means that anything you do to a variable inside a function has no effect on that variable outside of the function
- When call by reference is used, we can alter an argument outside of the function with actions inside the function

Called by Value

Write a function to swap two integers.
Try it now.

Called by Value

When an array variable being passed into a function as argument, what has been passed?
and what this means for manipulation of the array in local function to the array in the called scope?

getchar & putchar

- The C library function int getchar(void) gets a character (an unsigned char) from stdin.
- The putchar(int char) method in C is used to write a character, of unsigned char type, to stdout.

getLineNumber

Write a function getLineNumber which reads a text file from stdin and returns the number of lines.

```
int getLine()
```

maxLine

Write a function maxLine which reads a text file from stdin and returns the number of characters of the line which holds the most.

```
int maxLine()
```

getLineOrN

Write a function getLine which gets one line from stdin and returns the number of characters in this line.

```
int getLine(s[],int lim)
```

Specifically,

- The **s** being passed in will be used to store the all the characters on the line.
- The **lim** is the maximum length of **s**.
- We set a limitation on the length of **s** is because char array is not re-sizable. When we learned memory reallocation, we will have means to resize an array.

Storage Classes

- A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program. They precede the type that they modify. We have four different storage classes in a C program
  - **auto** - default storage class for all local variables.
  - **extern** - used to define a global variable or function, which will also be used in other files.
  - **static** - The static storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope.
  - **register** - used to define local variables that should be stored in a CPU register instead of memory/cache.
- Book Chapter: 4, page: 73
- More: https://www.geeksforgeeks.org/storage-classes-in-c/

External Variables

- In C, function name has to be unique, even they are compiled separately from different files, since they will be used to reference their binary code body.
- Variables usually exists inside a function body, and for each function there can not exists two variables assume the same name; inside different functions, there can be variables assume the same name.
- Well, there can also exist variables outside any function, and these variables are called external variables.

External Variables

- Like function names, external variables names must also be unique from each other.
- External variables and internal variables can share a same name, and they reference different memory address. And they have difference access scope.

Block and Scope

- Block: A section of code that is grouped together
  - In C, blocks are delimited by curly braces

    ```
    { [block statements] }
    ```
  - or the parenthesis of for loop

    ```
    for (int i=0 ; i<10; i++);
    for (int i=0 ; i<10; i++) {int j = 0; j = 1};
    ```
- Scope: the area of a program where a variable can be referenced
  - For each different entity that an identifier designates, the identifier is visible (i.e., can be used) only within a region of program text called its *scope*

Block and Internal Variable

- Variables are defined within a block are local to the block where they are defined by which it means that they are not accessible for the outside of the block; they come and go with the block of codes executing and finishing.
- Internal variables are also often called "auto" variables. Inside a function block, these two definition are equal (it is just the "auto" key word is often ignored):

```
int j = 0;
auto int j;
```

Block

- If a variable was not defined within this block, then it will resort to the outer block for the definition of this variable, until outside the function within the same file.
- Let's see demos.

External Variables

- For accessing an external variable that is not defined within this source code, we have to use the "extern" keyword.
- Let's see a demo.

A Summary

```
        Question:

Are external variables are the variables defined
by the "extern" keyword ?
```

External Variables and the "extern" key word

- No.
- An external variable is just a variable being defined outside any functions.
- The "extern" keyword is used for searching the external/global variable reference somewhere else. It means there is no variable definition here within this function.

Declaration vs Definition

- A variable is declared by "extern" is a declaration, which does not cause memory allocation.
- A variable definition means at this line of code, this variable will be allocated and reside in memory.

```
extern int i;  Declaration
extern int i=0; This is Definition, not Declaration

int i; int i=0; These are all variable definitions
```

Declaration vs Definition

- A variable definition is also a declaration, but declaration is not necessary to be a definition.
- A variable is to be used, has to at least have a declaration first.

External Variables

Advantages:

- If a large number of variables must be shared among functions, external variables are more convenient and efficient than long argument lists.
- External variables also retain their values after the exit of a function call, since no function owns it solely.
- External (global) variables are favored in high performance computing. They allow additional optimization by compilers.

External Variables

Disadvantages:

- It is problematic for decoupling a program structure, which makes a big software into less dependent parts such that it is easy for maintaining and testing etc.
- If their value gets corrupted, hard to trace the reason. They make functions dependent on their external environment
- In fact, software architecture/design standards often prohibit use of external variables

External Variables (External Static)

- External variables can be accessed by any function in the program.
- what if we want to limits its scope ?
- The static declaration, applied to an external variable or function, limits the scope of that object to the source file being compiled.

Static Local Variable

- Static local variable is a local variable that retains and stores its value between function calls or block and remains visible only to the function or block in which it is defined.

Example: Static External

```
#include <stdlib.h>
double drand48(void);
void srand48(long int seedval);
```

- The pseudorandom number generator `drand48()` is a family of functions
- They keep an external static variable X as the seed of the generators
- We must call `srand48()` to initialize the seed to generate a different sequence of numbers.

Example: Static Internal

```
#include <stdio.h>
int counter(){
     static int num;
     return num++;
}

int main(void){
     for (int i=0;i<5;i++) counter();
     printf("%d\n", counter());
}
```

The `register` Variables

- A register declaration advises the compiler that this variable will be heavily used
- We want it placed in a machine register, but the compiler is free to ignore this suggestion if it needs registers
- Can only be applied to automatic variables

The `register` Variables

- Register variables can be defined to local variables within functions or blocks, they are stored in CPU registers instead of RAM to have quick access to these variables.

  `Example: register int age;`
- A register variable may actually not be placed into registers in many situations.
- And it is not possible to parse the address of a register variable regardless of whether the variable is actually placed in a register.
- The specific restrictions on number and types of register variables vary from machine to machine.

Example: Register Variables

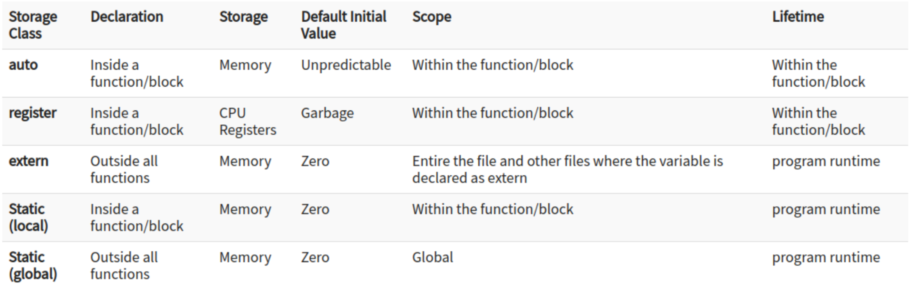
The variables declared using register has no default value. These variables are often declared at the beginning of a program.

```
#include <stdio.h>

int main(void) {
{
    register int  i;
    int *p=&i ;
    /*it produces an error when the compilation occurs,
    we cannot get a memory location when dealing
    with CPU register*/

    return 0;
}
```

Summary



Storage Class	Declaration	Storage	Default Initial Value	Scope	Lifetime
auto	Inside a function/block	Memory	Unpredictable	Within the function/block	Within the function/block
register	Inside a function/block	CPU Registers	Garbage	Within the function/block	Within the function/block
extern	Outside all functions	Memory	Zero	Entire the file and other files where the variable is declared as extern	program runtime
Static (local)	Inside a function/block	Memory	Zero	Within the function/block	program runtime
Static (global)	Outside all functions	Memory	Zero	Global	program runtime