Variable Scope

Variable Scope

- Two variables should have the same name only if they are declared in separate scope.

```
#include<stdio.h>
void max(int a, int b) {
...
}
void main() {
        int a,b;
        ...
}
```

Correct

```
#include<stdio.h>
void main() {
        int a,b;
        int a,b;
        ...
}
```

Wrong

Scopes of C program

- block/function scope (local scope)
- global/external scope

Function Scope

- Variable is valid within the block/function it is defined.
- Function parameters and variables defined in the function are valid only in the function.

```
int max(int a, int b) {
        int c;
        ...
}
```

Block Scope

- Similarly for block, variable scope is from declaration to termination of block.

```
void main() {
        int a = 5;
        {
                int b = 6;
                ...
        }
}
```

Scope: Shadow (local)

- In case of nested scopes, the inner scope takes precedence.

```
void main() {
        int m = 5;
        {
                float m = 6.5; //shadow
                printf(“%f”, m); //prints 6.5
        }
        printf(“%d”, m); //prints 5
}
```

Global/External Variable

- Variable is valid within the .c file it is defined.
- It is declared outside every function definition (even outside main).
- Can be accessed by all functions in the program that follow the declaration.
- Also called external variables.
- Global variables are useful for defining constants that are used by different functions in the program.

Example

```
#include<stdio.h>
int a=10;
void fun() {
            a=20;
            printf(“a=%d”, a);
}
void main() {
            fun();
            a=30;
             printf(“a=%d”, a);
}
```

Example (Why to use?)

```
#include<stdio.h>
const double PI = 3.14159;
double circumferenceCircle(double r)
{
	return 2*PI*r;
}
double areaCircle(double r) {
	return PI*r*r;
}
```

```
void main() {
	double r = 1.5;
	printf("Circumference: %lf\n", circumferenceCircle(r));
	printf("Area: %lf\n", areaCircle(r));
}
```

Output

```
Circumference: 9.424770
Area: 7.068577
```

Scope: Shadow (Global)

- What if a variable is declared inside a function that has the same name as global variable?
- The global variable is “shadowed” inside that particular function only.

Example

```
#include<stdio.h>
int g=10, h=20;  //global variables
int add() {
        return g+h;
}
void fun1() {
         //local variable shadow
        int g=200;
        printf(“%d\n”, g);
}
```

```
void main() {
        fun1();
        printf(“%d %d %d”, g, h, 
add());
}
```

Output

```
200
10 20 30
```

Constants via #define

```
#include<stdio.h>
#define PI 3.14159;
double circumferenceCircle(double
r) {
	return 2*PI*r;
}
double areaCircle(double r) {
	return PI*r*r;
}
```

```
void main() {
	double r = 1.5;
	printf("Circumference: %lf\n",
circumferenceCircle(r));
	printf("Area: %lf\n",
areaCircle(r));
}
```

Output

```
Circumference: 9.424770
Area: 7.068577
```

#define

- During the pre-processing step, the name with #define variable is replaced with the value everywhere in the program.

```
#define PI 3.14
...
void main() {
        area = PI*r*r;
}
```

```
#define PI 3.14
...
void main() {
        area = 3.14*r*r;
}
```

Count the number of function calls

- Can this be done using local variables?

```
int f() {
        int nCalls = 0;
        nCalls = nCalls + 1;
        ...
}
```

With every call to f(), nCalls is declared and when functions ends, the variable nCalls is destroyed. Thus, you will always get value 1.

Using Global

```
int nCalls = 0;
int f() {
        nCalls = nCalls + 1;
        ...
}
```

With global variable, the scope is the entire program. Thus, the value of nCalls is not destroyed when the function ends.

Static Variables

- It is created for the first time it is executed.
- Once created, it never gets destroyed and retains its value across invocations of functions.

```
void f() {
        static int nCalls = 0;
        nCalls = nCalls + 1;
        ...
}
```