ENGG1003 - Tuesday Week 4

Functions
Static Variables

Brenton Schulz

University of Newcastle

March 12, 2020

Writing Functions - Example

- Lets view a few common errors

```c
#include <stdio.h>
float mySqrt(float k);
int main() {
  printf("%f\n", mySqrt(26));
}
```

- Results in:

```
/tmp/ccT6mLDi.o: In function `main':
/projects/voidTest/hello.c:4: undefined
    reference to `mySqrt'
collect2: error: ld returned 1 exit status
```

Writing Functions - Example

- Likewise, forgetting the prototype:

```c
#include <stdio.h>
int main() {
  printf("%f\n", mySqrt(26));
}
```

- Results in (cut down):

```
hello.c: In function ‘main’:
hello.c:4:17: warning: implicit declaration of
    function ’mySqrt’
  printf("%f\n", mySqrt(26));
/projects/voidTest/hello.c:4: undefined
   reference to ‘mySqrt’
```

Function Compiler Errors

- “implicit declaration of...”
  - The function prototype is missing
- “undefined reference to...”
  - The function definition is missing

Function Definition Placement

- The following *works* but isn't recommended:

```
1  #include <stdio.h>
2  #include <math.h>
3
4  float mySqrt(float k) {
5    int n;
6    float xn = k/2.0;
7    for(n = 0; n < 10; n++)
8      xn = 0.5*(xn + k/xn);
9    return xn;
10 }
11
12 int main() {
13   printf("sqrt(26) = %.8f\n", mySqrt(26.0));
14   printf("Library sqrtf(26): %.8f\n", sqrtf(26.0));
15 }
```

- Only useful in very small projects but common

Function Arguments

- Function arguments automatically become variables inside the function

```
1 float mySqrt(float k) {  // k is an argument
2   int n;
3   float xn = k/2.0;  //k used here
4   for(n = 0; n < 10; n++)
5     xn = 0.5*(xn + k/xn); // and here
6   return xn;
7 }
```

- Don't declare them as variables!

Function Arguments

- By default, arguments are "passed by value"
- The function gets *copies*
- Modifying them in a function doesn't change the original variable
  - No, not even if they have the same name
- The argument variables are discarded on function return
- The return value is the *only thing* that goes back

Function Return Values

- Return values can only be one number
- How can we write a function which modifies (or returns) multiple things?

Function Return Values

- Return values can only be one number
- How can we write a function which modifies (or returns) multiple things?
- Pointers!

Function Return Values

- Return values can only be one number
- How can we write a function which modifies (or returns) multiple things?
- Pointers!
- We'll learn how to use pointers in Week 6(ish)
- For now, just learn to live with the single return value

Function Example

Write a C function, `isPrime()`, which takes an `int` as an argument and returns 1 if it is prime and zero otherwise

- Name: `isPrime`
- Argument(s): `(int x)`
- Return Value: `int`

Function Example

Write a C function, `isPrime()`, which takes an `int` as an argument and returns 1 if it is prime and zero otherwise

- Name: `isPrime`
- Argument(s): `(int x)`
- Return Value: `int`
- Function prototype:

```
int isPrime(int x);
```

Function Example

... Do it live!

Static Vs Auto Variables

- Any "normal" variable declared within the function (including arguments) is lost on function exit
  - These are called *auto* variables
- By default, any declared variable is an auto variable
  - Their value is lost outside the block where they are declared

Static Vs Auto Variables

- Any "normal" variable declared within the function (including arguments) is lost on function exit
  - These are called *auto* variables
- By default, any declared variable is an auto variable
  - Their value is lost outside the block where they are declared
- Alternatively, `static` variables can be used

Static Vs Auto Variables

- Any "normal" variable declared within the function (including arguments) is lost on function exit
  - These are called *auto* variables
- By default, any declared variable is an auto variable
  - Their value is lost outside the block where they are declared
- Alternatively, `static` variables can be used
  - Their value is retained

Static Vs Auto Variables

- Any "normal" variable declared within the function (including arguments) is lost on function exit
  - These are called *auto* variables
- By default, any declared variable is an auto variable
  - Their value is lost outside the block where they are declared
- Alternatively, `static` variables can be used
  - Their value is retained
  - Their scope is still limited

Static Variables

- Example: the `rand()` function returns different random numbers each time it is called
  - How? Shouldn't everything be lost when the function returns?
  - Not always! The `rand()` function's "state" is kept by a `static` variable.

Static Variables

- Example: the `rand()` function returns different random numbers each time it is called
  - How? Shouldn't everything be lost when the function returns?
  - Not always! The `rand()` function's "state" is kept by a `static` variable.
- Variables are static if declared with the `static` keyword
- Declaration examples:

Static Variables

- Example: the `rand()` function returns different random numbers each time it is called
  - How? Shouldn't everything be lost when the function returns?
  - Not always! The `rand()` function's "state" is kept by a `static` variable.
- Variables are static if declared with the `static` keyword
- Declaration examples:
- `static int k = 0;`
- `static float z = 0, y = 0;`
- `static long bigNum = 2345235234432;`

Static Variable Example

- Example: Write a function, `counter()` which returns an integer equal to the number of times it has been called.

Static Variable Example

- Example: Write a function, `counter()` which returns an integer equal to the number of times it has been called.
- Function prototype: `int counter(void);`

Static Variable Example

- Example: Write a function, `counter()` which returns an integer equal to the number of times it has been called.
- Function prototype: `int counter(void);`
- Function definition:

```
1 int counter() {
2   static int count = 0;
3   return count++;
4 }
```

Static Variable Example

- The variable `count` is declared `static`
- The initialisation, `count = 0`, happens *once*
- The value of `count` is retained between function calls

```
1 int counter() {
2   static int count = 0;
3   return count++;
4 }
```