COMP26020 Programming Languages and Paradigms -- Part 1

Functions

Functions

- They have a **name**, zero or more **parameters** with **types**, and a **return type**
- Like variables functions must be declared before being used

```c
int add_two_integers(int a, int b) {
    int result = a + b;
    return result;
}

int main() {
    int x = 1;
    int y = 2;

    int sum = add_two_integers(x, y);

    printf("result: %d", sum);

    if(add_two_integers(x, y))
        printf(" (non zero)\n");
    else
        printf(" (zero)\n");

    return 0;
}
```

Functions

- The absence of return value can be indicated with the `void` type

```c
void print_hello(void) {
    printf("hello!\n");
    return; // we can even omit this as the function does not return anything
}

int main() {

    print_hello();
    return 0;
}
```
07-functions/function-return-void.c

Function Calls

Function parameters and return value are passed as copies

```c
void my_function(int parameter) {
    parameter = 12; // does not update x in main
}

int main() {
    int x = 10;

    my_function(x);
    printf("x is %d\n", x); // prints 10

    return 0;
}
```

07-functions/function-parameters.c

Forward Declarations

```c
#include <stdio.h>

/* Forward declaration, also called function _prototype_ */
int add(int a, int b);

int main(int argc, char **argv) {
    int a = 1;
    int b = 2;

    /* Here we need the function to be at least declared -- not necessarily defined */
    printf("%d + %d = %d\n", a, b, add(a, b));

    return 0;
}

/* The actual function definition */
int add(int a, int b) {
    return a + b;
}
```

07-functions/forward-declaration.c

- The compiler parses source files from top to bottom
  - One can declare a function before 1) using it and 2) defining it

Variables Scope and Lifetime

- **Global** variables declared outside functions
  - visible at the level of the entire source file

```c
int global_var;

void add_to_global_var(int value) {
    global_var = global_var + value;
}

int main() {
    global_var = 100;
    add_to_global_var(50);
    printf("global_var is %d\n", global_var);
    return 0;
}
```

- Try to limit the use of globals as much as possible
  - A lot of global state makes it harder to reason about the program
  - More info why globals are dangerous: https://bit.ly/3iE0M2Y

Variables Scope and Lifetime

- Within a function, a declared local variable is accessible within its enclosing braces (a *block* of code)

```c
int x = 12;

if(x) {
    int y = 14;

    printf("inner block, x: %d\n", x);
    printf("inner block, y: %d\n", y);
}

printf("outer block, x: %d\n", x);

// ERROR -- y only visible in the if body:
// printf("outer block, y: %d\n", y);
```

```c
for(int i=0; i<10; i++) {
    printf("In loop body, i is %d\n", i);
}

// ERROR -- i only visible in loop body
// printf("Out of loop body, i is %d\n", i);

int j;
for(j=0; j<10; j++) {
    printf("In loop body, j is %d\n", j);
}

// WORKING -- j declared in current block
printf("Out of loop body, j is %d\n", j);
//                  07-functions/variable-scope-lifetime.c
```

- It is good practice for clarity to declare local variables at the beginning of their block

Summary

- Functions
- Global vs. Local variables

Feedback form: https://bit.ly/3jD6dRc