

COMP1917: Computing 1

5. Functions

Reading: Moffat, Chapter 5.

Programming Language Principles

Four techniques provided by almost all programming languages:

- Calculation: doing arithmetic to compute new values
- Selection: choosing between alternative execution paths
- Iteration: repeating a computation until desired conditions are met
- Abstraction: creating units which can be reused, and whose internal details are hidden from outside inspection.

Functions

Functions have the form:

```
return-type function-name ( parameters )
{
    declarations
    statements
    return(...);
}
```

- Functions allow you to separate out and “encapsulate” a piece of code that serves a single purpose – thus allowing code to be **reused** rather than just repeated.
- The use of functions serves to clarify your code – making it easier to read, modify and debug.

Function Calls

When a function is called:

1. space is allocated for its parameters and local variables.
2. the parameter expressions in the calling function are evaluated and, if necessary, converted to the declared parameter types.
3. C uses “call-by-value” parameter passing – which means that the function works only on its own local copies of the parameters, not the ones in the calling function.
4. local variables need to be assigned before they are used – otherwise they will have “garbage” values.
5. function code is executed, until the first `return` statement is reached.

The return Statement

6. when a `return` statement is executed, the function terminates:

```
return expression;
```

7. the returned expression will be evaluated and, if necessary, converted to the type expected by the calling function.
8. all local variables and parameters will be thrown away when the function terminates.
9. the calling function is free to use the returned value, or to ignore it.

Functions can be declared as returning `void`, which means that nothing is returned. The `return` statement can still be used to terminate such a function:

```
return;
```

Function Prototypes

If a function is defined after it is used, or in a separate file, a function **prototype** must be included at the top of the file.

```
int gcd( int a, int b ); // this is the function prototype
```

```
int main( void )
{
    g = gcd( x, y );      // the function is used here
}

int gcd( int a, int b )
{
    // the actual code for the function is here
}
```

Example: Euclid's Algorithm

```
int main( void ) {
    int g, a = 15, b = 42;
    g = gcd( a, b ); // compute Greatest Common Divisor
    printf( "GCD of %d and %d is: %d\n", a, b, g );
}

int gcd( int a, int b ) { // Euclid's method to find GCD
    int r; // remainder
    while( b > 0 ) {
        r = a % b;
        a = b; // only the local "copies" of a and b change,
        b = r; // not the ones in the calling function
    }
    return( a );
}
```

Scoping of Variables

Using the SAME name for DIFFERENT variables is NOT recommended, but here's what happens if you do ...

```
int x; // global variable, accessible to all functions

this_function()
{
    int x; // local variable "occludes" the global one
    ... // within this function
    {
        int x; // inner variable occludes
        ... // both the others within this block
    }
    // now we go back to the "outer" variable
}
```

Sharing Functions between Files

```
// tell compiler that loan_term() is defined elsewhere
double loan_term(
    double rate,
    double principal,
    double pay_amount,
    double num_per_year
);

int main( void )
{
    ...
    time = loan_term( rate, principal, payment, 12 );
    ...
}
```

Compiling Multiple Files

Two ways to compile multiple files:

```
> gcc mortgage.c loan.c
```

.. OR ..

```
> gcc -c mortgage.c loan.c
```

```
> gcc mortgage.o loan.o
```

The first command creates **object** files with a `.o` suffix.

The second command links these object files together to create an executable.

Only one of the files should contain a `main` function.

Generating Documentation Automatically

There are some utilities, like Doxygen, which can automatically create snazzy html documentation for your source code with appropriate formatting and clickable links, if you include comments in a particular format.

(<http://www.stack.nl/~dimitri/doxygen>)

```
> ~cs1917/bin/doxygen mortgage.c loan.c
```

Doxygen Example

```
/** \file loan.c
    Compute the time required to pay off a loan.
 */
/**
    @param rate           percentage interest rate
    @param principal      amount of money borrowed
    @param pay_amount     amount of each payment
    @param num_per_year   number of payments per year

    @return number of years to pay off the loan
 */
double loan_term( ... )
{
    ...
}
```