Functions

Song Liu (song.liu@bristol.ac.uk)
GA 18, Fry Building,
Microsoft Teams (search "song liu").

Entry point and Order of Execution

- Where does the program begin?
- In what order is my program executed?

Entry Point

```c
#include <stdio.h>

int main()
{
    // Program always starts HERE!!
    printf("My name is Song Liu. \n");
    // ... other stuff
    return 0;
}
```

- `main` function is **the entry point of your program**.
  - CPU will always find the `main` function in your code, and start executing your program.
  - No matter where the main function is located in the C file.
- All C (and C++) program must have a `main` function.

Case Study: Hey Jude

Hey Jude 1.0

```c
#include <stdio.h>
int main(){
    printf("Hey Jude, don't make it bad.\n");
    printf("Take a sad song and make it better.\n");
    printf("Remember to let her into your heart,\n");
    printf("Then you can start to make it better.\n");
    printf("\n");
    printf("Hey Jude, don't be afraid.\n");
    printf("You were made to go out and get her.\n");
    printf("The minute you let her under your skin,\n");
    printf("Then you begin to make it better.\n");
    return 0;
}
```

Writing Functions

- You can write all your code in `main`
  - it will work, but it is **very messy**!
  - It is like writing an essay in only one paragraph.
- We want to segment the program into pieces.
- Functions are basic units of your program.
  - It segments your program ino meaningful pieces.
  - Similar to "sections" in an essay.

Hey Jude 2.0

Consider the following code:

```c
#include <stdio.h>
void verse2(){
    printf("Hey Jude, don't be afraid.\n");
    printf("You were made to go out and get her.\n");
    printf("The minute you let her under your skin,\n");
    printf("Then you begin to make it better.\n");
}
void verse1(){
    printf("Hey Jude, don't make it bad.\n");
    printf("Take a sad song and make it better.\n");
    printf("Remember to let her into your heart,\n");
    printf("Then you can start to make it better.\n");
}
int main(){
    printf("Hey Jude by The Beatles \n");
    verse1(); //function call statement
    printf("\n");
    verse2(); //function call statement
    return 0;
}
```

Hey Jude 2.0

- In above, we wrote two functions: `verse1` and `verse2`.
- The program starts at `main`
- The CPU executes statements sequentially.
  - When `verse1` is **called**, it **enters** the function `verse1` and execute sequentially.
  - Once it finishes executing `verse1`, it **exits** `verse1` and continue where it left off in `main`.
- It continues to execute statements sequentially in `main`.
  - When `verse2` is **called**, it **enters** the function `verse2` and execute sequentially.
  - Once it finishes executing `verse2`, it **exits** `verse2` and continue where it left off in `main`.
- The CPU reaches the end of `main`, the program stops.

Functions

- **What is a function?**
- How to define a function?
  - Data Types in C
- Function Body
  - Variable Declarations
  - Expressions
- How to call a function?

What is a function?

- $f(x) = ax + b$.
- It receives an **input**.
- It produces an **output** following a certain rule.
- Function in programming is a generalization of this mathematical concept.

Functions in Programming

- Functions are individual building blocks of your program that accomplish specific tasks.
  - Function helps you divide your code into smaller, more **manageable and readable** pieces.
  - Statements in a function is only executed when its host function is "called".
  - In our "Hey Jude" example, `main` **calls** `verse1` and `verse2`
- Some functions take *input arguments* from the caller.
- Some functions return *an output value* to the caller after all its code are executed.
- Some functions do not have input or output.

Functions

- What is a function?
- **How to define a function?**
  - Data Types in C
- Function Body
  - Variable Declarations
  - Expressions
- How to call a function?

How to Define a Function?

1. A function definition starts by indicating the **return type** (`void` means no return value will be produced.).
2. Followed by the **function name**.
3. The **Input variable** come after that, inside `(` and `)`.
4. The **body of the function** is enclosed by `{` and `}`.

```
...
return_type function_name(input variable deceleration){
    function body
}
```

Your Own Function

You can write your own function and call it from `main()`:

```
...
void sayhello(){
    printf("Hello World!\n");
}
int main(){
    sayhello(); //calling "sayhello" function.
    return 0;
}
```

- You can choose your own name for your function, but it should be **succinct, reflects what your function does**.
  - e.g. sayhello, sort, add, delete, etc.
  - avoid long and ambiguous name.

Your Own Function 2

Below is an example calculating circumference of a circle:

```
...
double calculate_circumference(double radius){
    double circ = 2.0*3.1415926*radius;
    return circ; //returns the output to the caller
}
void main(){
    // call calculate_circumference
    double res = calculate_circumference(2.0);
    printf("%f\n", res); //print out result
}
```

- `calculate_circumference` takes one input argument `radius` and **returns** a decimal number.
- It is called by `main`, who will collect its returned value and printed it out.
- What is `double`?

Functions

- What is a function?
- How to define a function?
    - **Data Types in C**
- Function Body
    - Variable Declarations
    - Expressions
- How to call a function?

Data Types in C

- Some data types are:
  - `int` or `long` : integers
  - `float` or `double` : decimal numbers
  - `char` : characters
- Specifying data types tells the compiler: "reserve __ bytes of memory for this data when function is running!".
- On modern PCs (and most smartphones):
  - `int` and `float` occupies 4 bytes of memory.
  - `long` and `double` occupies 8 bytes of memory.
  - `char` occupies 1 byte of memory.

Expressiveness of Data Types

- Obviously, `long` and `double` are **more expressive** than `int` and `float`, but uses more memory.
- `int` has a range of -2147483648 to 2147483647.
- `long` has a range roughly plus or minus 9 quintillion
- `double` has about 15 decimal significant digits of precision, and has a range of about $\pm 10^{\pm 308}$
- If your memory space is precious, in applications such as computer graphics or data science, you can use `float`.

Functions

- What is a function?
- How to define a function?
  - Data Types in C
- **Function Body**
  - Variable Declarations
  - Expressions
- How to call a function?

Function Body

- The function body may contain any number of statements.
- The convention is

```
... function_name(...){
    declaration of variables
    ...

    other statements
}
```

Example

```c
#include <stdio.h>
double calc_gravity(double dist){
    //declare variable m1, m2, G and gravity.
    double gravity;
    double G = 6.674E-11;
    double m1 = 1.0, m2 = 2.0;

    //compute "gravity" using declared variables.
    gravity = G*m1*m2/dist/dist;
    return gravity;
}
int main(){
    printf("%E", calc_gravity(1.256));
    return 0;
}
```

Declarations

- Variable in C is a placeholder of some value.
- The value held by a variable can be changed later.
- In C programming language, all variables must be **declared**.
- The syntax of declaration is: `data_type variable_name`.
  - Declaration: `double gravity;`
  - Declaration **with initializations**: `double m1 = 1.0;`
  - Declaration of multiple variables of the same type: `double m1, m2;`.
  - Declaration of multiple variables of the same type with initializations: `double m1 = 1.0, m2 = 2.0;`.

Declarations

Once a variable is declared, **the variable is assigned a memory space by the compiler.** If the variable is uninitialized, **the variable can contain whatever (rubbish) value** that is already at that memory location!

- Undefined value leads to all sorts of unpredictable behaviors and is a source of error!

```c
#include <stdio.h>
void main(){
    // It will print out some garbage value.
    int a;
    printf("%d\n", a);
}
```

Declarations

- If possible, initialize the variable when it is declared.

```
...
double calc_gravity(double dist){
    //declare variable m1, m2, G and gravity.
    double m1 = 1.0, m2 = 2.0;
    double G = 6.674e-11;
    //initialize gravity as soon as it is declared.
    double gravity = G*m1*m2/dist/dist;
    return gravity;
}
...
```

- If you do not know how to initialize the variable, assign an "default value" (e.g. 0), so that when you see the value, you know this variable has not been assigned any useful value.

Expressions and Assignments

- You can use **variables** and **operators** to construct **expressions**:
  - `G*m1*m2/dist/dist`
- **Each expression has a value.** The value of `G*m1*m2/dist/dist` is its computation outcome.
- `m1 = 1.0`, `gravity = G*m1*m2/dist/dist` are all **assignment expressions**.
  - It assigns the value of expression on the RHS to the variable on the LHS.
  - The value of an assignment expression is the value that is being assigned.
  - `printf("%d", a=1); // prints 1`

Functions

- What is a function?
- How to define a function?
  - Data Types in C
- Function Body
  - Variable Declarations
  - Expressions
- **How to call a function?**

Calling a Function

- You can call a function **after its definition**.
- Calls are made writing the function name followed by all input values **in the same order they are declared**!
- Suppose your function is declared as
  - `int function_name(int var1, int var2, int var3)`
- You can call it like
  - `function_name(1, 2, 3)`
  - 1 is the input of `var1`, 2 is the input of `var2` ...
  - **Each function call is an expression** whose value is the return value of the function

Calling a Function

```c
#include <stdio.h>
double calc_gravity(double m1, double m2, double dist){
    //declare variable G and gravity.
    double G = 6.674e-11;
    return G*m1*m2/dist/dist;
}
int main(){
    // what is the input value of dist?
    double res = calc_gravity(1.0, 2.0, 1.256);
    // the function call calc_gravity(1.0, 2.0, 1.256)
    // is an expression, it is used to initialize
    // the variable res above.
    printf("%E\n", res);
    return 0;
}
```

To Sum Up

- Writing functions are great ways to split your program into smaller, and more specific tasks.
- Functions can take input variables and return output value.
- Function body includes variables declarations and other statements.
    - When you declare a variable, the compiler automatically allocates certain amount of memory space for that variable.
- Functions are called using their names, with input variables in the same order arranged in the function definition