COMP1511 - Programming Fundamentals

Week 3 - Lecture 5

mv 文件.c lecture04 同一层
../lecture05 上一层

Specific Issues

- Header comment doesn't show the program's intentions
- No blank lines separating different components
- Multiple expressions on the same line
- Inconsistent indenting 不一致的缩进
- Inconsistent spacing 不一致的间隔
- Variable names don't make any sense
- Comments don't mean anything
- Inconsistent bracketing of if statements
- Bracketing is not indented {}不缩进
- Inconsistent structure of identical code blocks
- The easter egg - there's actually incorrect code also!

Keeping your house (code) clean

Regular care is always less work than a big cleanout

- Write comments before code
- Name your variables before you use them
- **{** everything inside gets indented 4 spaces
- **}** line up your closing brackets vertically with the line that opened them
- One expression per line
- Maintain consistency in spacing

Comments before code

Comments before code. It’s like planning ahead

- Making plans with comments
- You can fill them out with correct code later
- Some of these comments can stay even after you've written the code

```
// Checking against the target value
if () {
    // success
} else if () {
    // tie
} else {
    // failure (all other possibilities)
}
```

Indentation

A common convention is to use 4 spaces for indentation

```
int main (void) {
    // everything in here is indented 4 spaces
    int total = 5;
    if (total > 10) {
        // everything in here is indented 4 more
        total = 10;
    }
    // this closing curly bracket lines up
    // vertically with the if statement
    // that opened it
}
// this curly bracket lines up vertically
// with the main function that opened it
```

One expression per line

Any single expression that runs should have its own line

```
int main (void) {
    // NOT LIKE THIS!
    int numOne; int numTwo;
    numOne = 25; numTwo = numOne + 10;
    if (numOne < numTwo) { numOne = numTwo; }
}
```

```
int main (void) {
    // Like this :)
    int numOne;
    int numTwo;
    numOne = 25;
    numTwo = numOne + 10;
    if (numOne < numTwo) {
        numOne = numTwo;
    }
}
```

Spacing

Operators need space to be easily read

```
int main (void) {
    // NOT LIKE THIS!
    int a;
    int b;
    int total=0;
    if(a<b&&b>=15){
        total=a+b;
    }
}
```

空格

```
int main (void) {
    // Like this :)
    int a;
    int b;
    int total = 0;
    if (a < b && b >= 15) {
        total = a + b;
    }

}
```

Weekly Tests

one hour only

Self Invigilated Weekly Tests start this week

- A mini exam you run yourself
- The detailed rules are in the test itself
- Releases on Thursday and you will have one week to complete it

- Use it as a way to test your progress so far
- Great practice for coding with time pressure and limited resources (exams or job interviews)

Functions

Let's introduce at functions

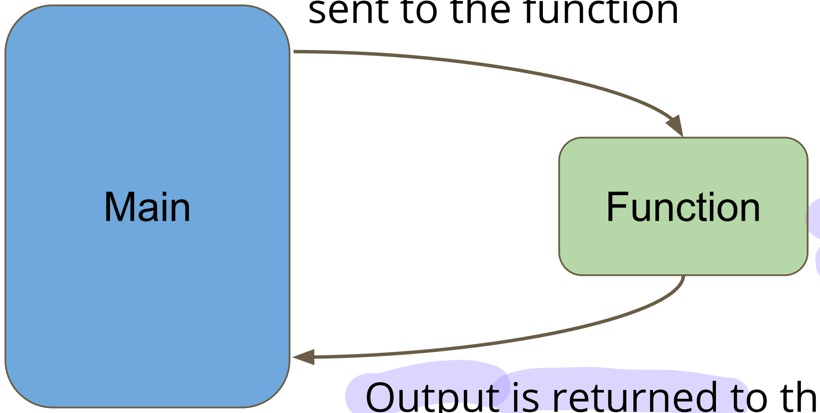
- We've already been using some functions!
- **main** is a function
- **printf** and **scanf** are also functions

What is a function?

- A separate piece of code identified by a name
- It has inputs and an output
- If we "call" a function it will run the code in the function

Functions

How do they work?



Main calls the Function, asking it to run. Input is sent to the function

Function runs its code using the input

Output is returned to the Main. Main continues from where it was

Function Syntax

We write a function with (in order left to right):

- An output (known as the function's type)
- A name
- Zero or more input(s) (also known as function parameters)
- A body of code in curly brackets

```
// a function that adds two numbers together
int add (int a, int b) {
    return a + b;
}
```

Return

An important keyword in a function

- `return` will deliver the output of a function
- `return` will also stop the function running and return to where it was called from

How is a function used?

If a function already exists (like printf)

- We can use a function by calling it by name
- And providing it with input(s) of the correct type(s)

```
// using the add function
int main (void) {
    int firstNumber = 4;
    int secondNumber = 6;
    int total;

    total = add(firstNumber, secondNumber);
    return 0;
}
```

Compilers and Functions

How does our main know what our function is?

- A compiler will process our code, line by line, from top to bottom
- If it has seen something before, it will know its name

```
// An example using variables
int main (void) {
     // declaring a variable means it’s usable later
     int number = 1;

     // this next section won’t work because the compiler
     // doesn’t know about otherNumber before it’s used
     int total = number + otherNumber;
     int otherNumber = 5;
}
```

Functions and Declaration

We need to declare a function before it can be used

```c
// a function can be declared without being fully
// written (defined) until later
int add (int a, int b); exist

int main (void) {
    int firstNumber = 4;
    int secondNumber = 6;
    int total = add(firstNumber, secondNumber);
    return 0;
}

// the function is defined here
int add (int a, int b) {
    return a + b;
}
```

function 主体

Void Functions

We can also run functions that return no output

- We can use a void function if we don't need anything back from it
- The return keyword will be used without a value in a void function

```
// a function of type "void"
// It will not give anything back to whatever function
// called it, but it might still be of use to us
void add (int a, int b) {
     int total = a + b;
     printf("The total is %d", total);
}
```

no output