Conditional Statement and Loops

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

Previously

- Non-sequential Execution
  - Example: Hey Jude
- Function
  - Definition of Function
  - Data Type
  - Declaration of Variables
  - Input and Output of a Function
  - Function Calls
- Have a look at examples!

Issues in Previous Lab

- Step over (F10) vs. Step in (F11)
  - Step over will **not** enter function while it is being executed!
  - To enter a function, you need to **step in**.
- Do not **define** one function inside another function

```
void koo(){
    void goo(){ //WRONG!!!COMPILATION ERROR!
        //...
    }
}
```

  - However, you can **call a function** from another function.
  - **Know the diff. between function definition and call**.

Issues in Previous Lab

- Input of a function can be the output of another function
    - The input of a **function can be any expression** whose value **is compatible with the input type**.

```c
#include <stdio.h>
double square(double a){
    return a*a;
}
void main(){
    double a = 2;
    printf("%f", square(square(a))); // output 16
}
```

Conditional Statement

- We have seen how we can use functions to write non-sequential code.
- Now, we will introduce two other ways to write non-sequential code:
    - Conditional Statement
    - Loops

Conditional Statement

- In many cases, we want to write a program responding to different conditions.
  - If a score >= 40, print "pass". Otherwise, print "fail".
  - If user's input equals to password, proceed to log in. Otherwise, print out an error message.

If-Else

- This simple conditional statement is called "if-else" statement and exists in many programming languages.
- If-Else statement in C is written as follows:

```
    if (condition){
        statements to be executed,
        if condition is true
        ...
    } else{
        statements to be executed,
        if condition is false
    }
```

    - If condition is true, the program bypasses all statements following `else`.
- `condition` can be logical and relational expressions.

If-Else

```
int score = 41;
if(score >= 40){
    printf("pass!\n");
}else{
    print("fail!\n");
}
//prints "pass"
```

- `else` clause is optional.

```
int password = 4321;
if(password == 1234){
    printf("pass!\n");
}
printf("1...");
printf("2...\n");
// Prints out 1... 2...
```

Relational and Logical Expressions

- We often use relational and logical expressions as conditions in if-else statements.
- Relational operator compares expressions on both sides of the operator.
  - `score>3`, compares variable `score` with constant `3`.
- Logical operator performs logical operations given expressions on both sides of the operator
  - `a> 0||b >0`. `a` greater than 0 OR `b` greater than 0.
- The values of these expressions can either **be 0 (FALSE) or 1(TRUE)**.

Relational and Logical Operators

- Relational Operators
  - `>` strictly greater. `2>1` is TRUE.
  - `>=` greater or equal. `2>=3` is FALSE.
  - `==` equals to. `1 == 1` is TRUE.
    - Note, single `=` is assignment operator. It assigns the value of RHS to the variable on the LHS. Do not get confused!
  - `!=` not equal. `2 != 1` is TRUE.
- Logical Operators
  - `&&` logic AND. `1>0 && 1>-1` is TRUE.
  - `||` logic OR. `1>0 || -1>0` is TRUE.

if-else-if Ladder

- What if we have more than two branches?
- For example, we classify students into 5 categories:
  - $\text{score} >= 70$, first class.
  - $60 <= \text{score} < 70$, two-one (above average)
  - $50 <= \text{score} < 60$, two-two (average)
  - $40 <= \text{score} < 50$, pass
  - $\text{score} < 40$, fail.
- Can we use conditional statement to do that?

if-else-if Ladder

```
if (condition1){
    statment1;
} else if(condition2){
    statment2;
} else if(condition3){
    statment3;
}
...
else{ //optional
    statement0;
}
```

- The program will check conditions **sequentially**.
- Once a true condition is found
    - It executes the associated statements.
    - then **bypasses the rest of the ladder**.
- If none of the conditions are true, it executes the `else` statements (if there is one).

if-else-if Ladder

The ladder below prints out the classification given a score.

```
int score = 55; // score is 55.
if (score >= 70){
    printf("First Class.\n");
} else if(60<=score && score < 70){
    printf("Two-One.\n");
} else if(50<=score && score < 60){
    printf("Two-Two.\n");
} else if(40<=score && score < 50){
    printf("Pass.\n");
}
else{
    printf("Fail\n");
}
// prints out Two-Two
```

- Note that if an earlier condition check is true, it bypasses the entire ladder without checking the latter conditions!

if-else-if Ladder

If I made a mistake on the second condition

```
int score = 55;
if (score >= 70){
    printf("First Class.\n");
} else if(50<=score && score < 70){ // typo, 60->50
    printf("Two-One.\n");
} else if(50<=score && score < 60){
    printf("Two-Two.\n");
} else if(40<=score && score < 50){
    printf("Pass.\n");
}
...
```

- What will happen?
  - Will it still prints out "two-two"?
- Careful! Mistakes like this is hard to detect!
  - The program runs, but the output is wrong!

Loops

- In programming language, we sometimes want to repeat a certain operation for many times.
  - Adding up a sequence of numbers
  - Read a text file until it reaches the last line.
- This mechanism is called **loop**.
- When encounter loops, the CPU will continue to execute a code block, until certain exit conditions are met.
- Loop is another case where code do not run sequentially.

While Loop

The simplest loop is while-loop and its syntax is:

```
while(condition){
    statements
}
```

The statements inside of the brackets will be run repeatedly as long as the `condition` is true.

```
// print out every positive integer smaller or equal than 10
int i = 1;
while(i<=10){
    printf("%d\n", i);
    i = i + 1;
}
```

Note that `i` changes every iteration.

Iteration and Loop Counter

```c
int i = 1;// define loop counter
while(i<=10){
    printf("%d\n", i);
    i = i + 1; // increment of i
}
```

- Each repetition of the loop is called **iteration**.
  - The loop above **iterates** 10 times.
- In loop, we commonly have an integer variable keeping the count of repetitions. Such a variable is called **loop counter**.
  - `i` is the loop counter in the above loop.
  - The counter is initialized before the loop and is increased by one before the end of each iteration.
  - We stop the loop by checking if `i <= 10`.

For Loop

- For loop is another type of loop mechanism in C.

```
for(init statement; condition; update statement){
    statements to be repeated
}
```

i. It initializes a counter.

ii. Check condition,

- If it is satisfied, run statements
- If not, exit the loop.

iii. Go back to ii.

For Loop

Prints out all positive integer smaller than 10

```
int i;
for(i=1; i<=10; i = i + 1){
    printf("%d\n", i);
}
```

You can put the declaration of `i` inside of the loop too.

```
for(int i=1; i<=10; i = i + 1){
    printf("%d\n", i);
}
```

It is more succinct than the `while` loop. Iteration counter `i` initialized, checked and updated all in one line.

`for` vs `while` loop

- Both loop mechanisms are widely used in algorithms.
- Use `while` loop when **you do not know how many times the loop will be run**.
    - When asking user's input, you do not know when the user will finish.
    - When playing a game, you do not know when user will hit the "exit" button.
- Use `for` loop when **you know exactly how many times the loop will repeat**.
    - Print out number from 1-10.
    - Sum up students' scores in a class.

To sum up

- In this lecture, we learned how to write non-sequential code using:
  - Conditional Statements
    - `if-else`
    - `if-else if-else` ladder
  - Relational and Logical operators.
    - `>, >=, ==, !=, &&, ||`.
  - Loops
    - `while` loop
    - `for` loop

Lab

- Download the lab file from github, unzip and place it in your labpack folder.
    - Copy the folder `2_conditional_and_loops` and the file `lab_2.bat` into your labpack folder.
    - The same way we did in the last lab.
- Double click `lab_2.bat` to start the Visual Studio code.
- The next week's lab will be the last time we help with lab pack setting issues. After that, we assume everyone can use the lab environment without any issue.

Homework 1

- Open `score.c` file, and run the program step by step using debugger as I did in the lecture.
    - Use F10 to step over.
    - See the workflow of the if-else ladder yourself.
- Change the `score` variable declared in line 4 to `10`, `40`, `50`, `90` and guess the output without running the code.
    - Did you guess right?
    - If not, trace the execution of the code using a debugger and see where you got wrong.

Homework 2

- Now, open `odd_even.c`
- Write an `if-else` in the specified place, so that the program prints the following messages.
  - `odd`, if variable `num` is odd.
  - `even`, if variable `num` is even.
- Hint: modulo operation in c is `%`.
  - `4%4` is 0.
  - `4%3` is 1.
- Hint, you can copy the `if-else` statement from the lecture slides and modify it to fit your needs.

Homework 3

- Now, open `factor.c`.
- Write an if-else-if ladder in the specified location, so that the program output
  - `divided by 3`, if `num` can be divided by 3.
  - `divided by 4`, if `num` can be divided by 4.
  - `divided by 9`, if `num` can be divided by 9.
  - If `num` can not be divided by `3,4,9`, output `Oops!`.
- One special rule:
  - If `num` can be divided by both `a` and `b` and `a>b`, it should only output `divided by a`.
- Hint, think about the workflow of `if-else-if` ladder.

Homework 4

- Open `whileloop.c`.
- Run the debugger and trace the program step by step.
- Open `forloop.c`.
- Run the debugger and trace the program step by step.
- Make sure you understand the workflow of a for loop and a while loop.
    - Ask questions if you are confused.

Homework 5 (submit)

- Open `factor2.c`.
- Write a function `is_divisible` that accepts one integer input `num` and returns no output.
    - Depending on the input `num`, `is_divisible` should printout messages according to the same rules described in homework 3.
    - You can copy and paste your code from homework 3.
- Now call `is_divisible` inside a loop, so the program check the divisibility for all integers ranging from 939 to 945, inclusive.
    - You can use either `for` or `while` loop.
- Submit this homework on blackboard.

Submission (important!)

To help us automate the marking process, please make sure your submission is named as:

- `STU_ID.c`, where `STU_ID` should be a string starting with two letters followed by numbers, e.g. `ab1234`.
- The same ID is also used in your email address. For example, if you send an email to `STU_ID@bristol.ac.uk`, you should receive that email.
- DO NOT use `.cpp` as the extension name! In the assessed CWs, we will only mark submissions with the correct name.