202018021
Fu, Ziyu
Programming I Chapter 3 Task 1, 2, 3
2021/05/18

Task 1

1. Objective
   To learn the mechanism of `for` and `while` loops and apply it in a simple program.

2. Strategy of solving
   - Read the lecture content and understand the mechanism
   - Design the program and write the code
   - Observe the output

3. Program code

```c
#include <stdio.h>

int main(){

    int counter = 0;
    while (counter < 5){
        printf("You are a sophomore of the undergraduate.\n");
        counter++;
    }

    return 0;
}
```

```c
#include <stdio.h>

int main(){

    for(int i = 0; i < 5; ++i){
        printf("You are a sophomore of the undergraduate.\n");
    }

    return 0;
}
```

4. Results and discussions
   Console output (same for both programs):

```
You are a sophomore of the undergraduate.
You are a sophomore of the undergraduate.
You are a sophomore of the undergraduate.
You are a sophomore of the undergraduate.
You are a sophomore of the undergraduate.
```

   Both programs loop the `printf()` command for 5 times and they both output as expected.

   The `while` loop relies on a `counter` variable declared outside the loop which increments at the end of each repetition. The `while` condition is checked at the start of each repetition.

   The `for` loop initializes the counter variable in line, and increments the counter at the start of each repetition after the first one. The comparison condition is also checked at the start of each repetition.

Task 2

1. Objective
   To test and successfully run the sample code given in the textbook.

2. Strategy of solving
   - Inspect the structure of the provided sample code
   - Run the code
   - Observe the output

3. Program code

```c
#include <stdio.h>

int main(){
    int i, j, n, my_id;
    j = 1;
    n = 3;
    my_id = 21;
    while (n <= my_id){
        i = 2;
        while (i < n){
            if (n % i == 0){
                break;
            }else{
                i = i + 1;
            }
        }
        if (i == n){
            printf("%5d", n);
            if (j % 10 == 0){
                printf("\n");
            }
            j = j + 1;
        }
        n = n + 1;
    }
    printf("\n");
}
```

4. Results and discussions
   Console output:

```
    3    5    7   11   13   17   19
```

This program finds all the prime numbers up to a certain limit (`my_id`). It loops through every counting number (`n`) smaller than the limit, and for each `n`, it checks if it can be divided by any number smaller than itself (`i`). If it can't be divided by any number smaller than itself, it is identified as a prime and outputted to the console. `j` only checks the number of primes displayed so it can insert an `end-of-line` every 10 numbers. The output is as expected.

Task 3

1. Objective
   To understand the sample code provided in the textbook and modify it in order to compare the mechanism between **`while`** and **`for`** loops.

2. Strategy of solving
   - Inspect the structure of the provided sample code
   - Modify the code
   - Observe the output

3. Program code

```c
#include <stdio.h>

int main(){
    int i, j, n, my_id;
    j = 1;
    n = 3;
    my_id = 21;

    for (; n <= my_id; ++n){
        i = 2;

        for (; i < n; ++i){
            if (n % i == 0){
                break;
            }
            // else{
            //     i = i + 1;
            // }
        }
    if (i == n){
            printf("%5d", n);
            if (j % 10 == 0){
                printf("\n");
            }
            j = j + 1;
        }

    }
    printf("\n");
}
```

4. Results and discussions
   Console output:

```
    3    5    7   11   13   17   19
```

This logic of the modified program is the same as the original sample code, however it uses **`for`** loops instead of **`while`** loops. This is a rather direct (“lazy”) modification of the original program, aimed to show the similarities between **`for`** and **`while`** loop. That is why the declaration and initialization of the counter variables are left deliberately at their original place. **`for`** loops are effectively a syntax sugar for **`while`** loops. Also, from **`line 15`** to **`18`**, because the **`for`** loop would increment the counter by default, the **`else`** statement can be taken out. The program outputs the results as expected.