

## Tutorial 1 - Basics of C & Control Structures

1. Being able to trace through the execution of a program without running it is known as program tracing. Program tracing involves building up a mental model of program execution in your mind (or on paper). By tracing a program, you will gain a stronger understanding of the program execution flow and improve your programming skill. Consider the following program:

```
In [ ]: #include <stdio.h>

int main(void)
{
    int cur, prev1=1, prev2=1;
    cur = prev1 + prev2;
    prev2 = prev1;
    prev1 = cur;

    cur = prev1 + prev2;
    prev2 = prev1;
    prev1 = cur;

    cur = prev1 + prev2;
    prev2 = prev1;
    prev1 = cur;

    printf("cur is %d; prev1 is %d; prev2 is %d\n", cur, prev1, prev2);
    return 0;
}
```

Trace the execution of the program, paying particular attention to variable declarations and changes to the values of the variables. Verify your program trace by including your own debugging printf statements at strategic points in the program to track the value changes of variables.

0) cur=2, prev1=1, prev2=1  
1) cur=2, prev1=1, prev2=1  
2) cur=2, prev1=2, prev2=1  
3) cur=3, prev1=2, prev2=1  
4) cur=3, prev1=2, prev2=2  
5) cur=3, prev1=3, prev2=2  
6) cur=5, prev1=3, prev2=2  
7) cur=5, prev1=3, prev2=3  
8) cur=5, prev1=5, prev2=3

cur is 5; prev1 is 5; prev2 is 3

2. Assume there are three integers a, b and c already defined in our program, and we want to assign the value of the largest integer to variable max. For each of the following code fragments below, fill in the missing parts so that the code will fulfil the given task.

```
In [ ]: int part_a(int a, int b, int c) {
        int max;
        if (a > b || a > c) {
            max = a;
        } else {
            if (b > c) {
                max = b;
            } else {
                max = c;
            }
        }
        return max;
    }

    int part_b(int a, int b, int c) {
        int max = a;
        if (b > a) {
            max = b;
        }
        if (c > a) {
            max = c;
        }
        return max;
    }

    int part_c(int a, int b, int c) {
        int max;
        if (a > b) {
            if (a > c) {
                max = a;
            } else {
                max = c;
            }
        } else {
            if (b > c) {
                max = b;
            } else {
                max = c;
            }
        }
        return max;
    }
}
```

3. Study the following program fragment:

```
In [ ]: int foo(int a)
        {
            if (a > 0)
                if (a >= 1000)
                    a = 0;
                else
                    if (a < 500)
                        a = a * 2;
                    else
                        a = a * 10;

            else
                a = a + 3;
        }
}
```

Deduce the functionality of the program fragment and re-compose the nested if..else statements into a form that is easier to understand.

```
In [ ]: int foo(int a)
{
    if (a > 0 && a < 500)
    {
        a = a * 2;
    }
    else if (a >= 500 && a < 1000)
    {
        a = a * 10;
    }
    else if (a >= 1000)
    {
        a = 0;
    }
    else
    {
        a = a + 3;
    }
    return a;
}
```

4. Given the following program fragment:

```
In [ ]: int i=1;
while (i > 0)
{
    i = i + 1;
}
printf("%d\n", i);
```

- What do you think will happen?
- Run the program, observe what happens and make your own deductions.

a. i will continue to increase by 1, as i continuously satisfies the while loop condition ( $i > 0$ ). i will stop and print the results when it reaches the maximum int value: 2147483647.

b. The result is -2147483648 instead. At the binary level, 01111111111111111111111111111111 -> 10000000000000000000000000000000, with two's complement, the second number is negative, hence -2147483648.

5. Given the following program fragment:

```
In [ ]: int n=321, count2=0, count3=0, count5=0;

for (int i = 0; i <= n; i=i+1)
{
    if (i%5 == 0)
    {
        count5 = count5 + 1;
        if (i%3 == 0)
        {
            count3 = count3 + 1;
        }
    }
    else
    {
        if (i%2 == 0)
        {
            count2 = count2 + 1;
        }
    }
}
printf("%d %d %d\n", count5, count3, count2);
```

- Perform a timeline trace of the variables count2, count3 and count5 from  $i = 0$  to  $i = 30$ .
- Using the trace above, predict the output of the program.
- Verify your prediction by running the program.

count5: increment of 5  $\{321/5 = 64\}$  &  $0 = 65$

count3: increment of 5 && increment of 3  $\{321/15=21\}$  &  $0 = 22$

count2: increment of 2  $\{321/2=160\}$  - increment of 10  $\{321/10=32\} = 128$

6. Write a program that takes in two positive integers and returns the greatest common divisor (*gcd*) of the two integers. For example, the gcd of 539 and 84 is 7. Two algorithms for determining the gcd are given below:

(a) Set a variable *gcd* to be the smaller of the two values. If this value of *gcd* completely divides the two numbers, then return this value as the *gcd*. Otherwise, reduce the value of *gcd* by one and repeat the test.

(b) We apply the Euclidean algorithm. Let the two values be *a* and *b*. Replace *b* with the remainder of  $a/b$ , and *a* with the original value of *b* (before the replacement). Keep doing this until *b* becomes zero; the value of *a* is the *gcd*.

```
In [ ]: int gcd(int a, int b)
{
    // Loop until b is 0//
    if (b != 0)
    {
        /*Recursion*/
        //Replace b with remainder of a%b//
        //Replace a with original value of b//
        return gcd(b,a%b);
    }
    else
    {
        //If b=0, a is the gcd results//
        return a;
    }
}
```

-END-