LOOP STRUCTURES

Aims

1. Usage for loop
2. Usage of while loop
3. do – while loop
4. Use of break and continue

We will be using Microsoft Visual Studio 2008 in this experiment; however, all codes shown in here should work with any ANSI-C complaint compiler.

for Loop

Generally, for loop is used to perform some operation while a variables value increases or decreases between two values. for loop consists of three sections, first section is the starting section and only runs at the start of the loop. Second section is the condition and is checked before the code block inside the for loop is executed. Last section is the increment. It is performed after code block is executed, before the check for the next step. This section will not run at the start. You may perform two operations in first or second section using coma (,) between operations. If you do not require one of the sections, you may leave it empty. If condition section is empty, the loop will continue forever. Sections are separated by semicolon (“;”). Just like if statement, for statement is followed by a statement or compound statement. Consider the following example and its output.

```
for(i=0;i<10;i++) {
   printf("%d ", i);
}
```

```
0 1 2 3 4 5 6 7 8 9
```

Note: similar to if statement, you should not use semicolon after for loop. Doing so will cause problems with your program. Examine the following code.

```
for(i = 0; i<=5; i++);
   printf("%d\n", i);
```

The program above will print 6 as output, because for loop will continue to increase the value of **i** until it becomes 6 where it is no longer less than or equal to 5 without performing any other

operation. Note that semicolon itself is called null statement and does not perform any operation. The printf function following the for loop will print the last value of **i** which is 6.

while Loop

Using while loop you may run a statement or a compound statement while the given condition is true. Like for loop, while should be followed by a statement or compound statement, not a semicolon.

```
X = 0;

while(x <= 5) {
   x += 1;
   y = x;
   y++;
   x = y;

   printf("%d", y);
}
```

do-while loop

When required, it is possible to move conditional statement to the end of while loop. This will guarantees that the statement inside the loop will run at least once and conditional statement will be checked after the first execution. To construct such a loop, use do keyword at the top of the loop and move while statement to the end of the loop. The while at the end is not followed by any other statement or compound statement; therefore, it should have semicolon afterward.

```
do {
   scanf("%d", &x);
   printf("x = %d\n", x);
} while(x>0); /*Since x is always read before checking this
statement it is safe to check its value in here*/
```

break and continue

It is possible to terminate a loop without satisfying its condition. You may use break keyword for this. break should be used in exceptional situations. Using it as a normal part of your loop will increase the complexity of your program. For instance, instead of using break in the following program, it is possible to change its condition. At the first glance, program seems to print until 5; however, it will terminate before printing 2 by a break keyword.

```
for(i=0; i<5; i++) {
   if(i == 2) {
```

```
        break;
   }

   printf("%d\n", i);
}
```

The following is the proper use of break keyword.

```
for(i=0; i<10; i++) {
   printf("Please enter a positive value: ");
   scanf("%d", &number);

   if(number<0) {
        printf("The number you have entered is invalid.\n");
        break;
   }

   sum += number;
}
```

continue keyword can be used to skip the rest of the loop. However, this operation can simply be performed using if. Therefore, if not really required (or requested from you), it's better not to use it. The following code segment shows its use.

```
while(1) {
   printf("Please enter a number, negative number to exit: ");
   scanf("%d", &number);

   if(number == 0) {
        printf("This operation is not defined for 0.\n");
        continue;
   }
   if(number < 0) {
        break;
   }

   printf("7 mod number = %d", 7 % number);
}
```

Experiment

1. Write a program to calculate sum of the numbers between 10 and 100 using all types of loop we have learned so far (for, while and do … while). The following is the equation for this operation.

$$\frac{\sum_{i=10}^{100} i}{91}$$

2. Write a program that calculates the following formula. Read the value of n from the keyboard.

$$\sum_{i=1}^{n} i^2 + 5$$

3. Write a program that prints requested amount of asterisk characters to the screen. This program should ask the amount again and again until user enters a value higher than 0.

Note: this program is composed of two sections; each section requires a different type of loop. Use the most appropriate loop type for these sections.

4. Write a program that prints the digits of the given number starting from the least significant digit. Print digits in separate lines. If user enters a value less than 0, print a message that the number is invalid.

Not: Use the most appropriate loop type.

5. Read a number from the user. Then ask user to enter number amount of values and calculate sum, minimum and maximum of these numbers.