BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJ.)

Second Semester 2019-20   CS F111 Computer Programming

LABORATORY SESSION #7

(Operators and Expressions; Flow of Control)

1. Write a program that reads an integer that represents a month (1 for January… 12 for December), and then displays the season in which the month belongs to: December through February is winter, the next three months are spring, the next three qualify as summer, and September through November are autumn. Do this (i) using the if construct and (ii) without it.

2. Write a program that reads a four-digit positive integer and uses the `?:` (ternary) operator to display a message whether it can be read the same in reverse order or not (e.g., the number 7667 can be). Do not use an `if` statement.

3. Write a program that simulates a physical calculator. The program should read the symbol of an arithmetic operation (any of the five: +, –, *, / and %) and two integers and display the result of the arithmetic operation. Remember to incorporate an error check for division by zero. Complete the program whose first few lines are shown below:

```
#include <stdio.h>
int main()
{
      char sign;
      int i, j;
      printf("Enter math sign and two integers: ");
      scanf("%c%d%d", &sign, &i,&j);
      ...
      ...
```

4. The following code is meant to give you practice with short-circuit evaluation:

```
int a=0, b=0, x;

x = 0 && (a=b=777);
printf("%d %d %d\n", a, b, x);

x = 777 || (a = ++b);
printf("%d %d %d\n", a, b, x);

x = 0 && ++a || ++b;
printf("%d %d %d\n", a, b, x);
```

   What gets printed? First, write down your answers. Then, write a program to check them.

5. What output do you think the following program should give? What does it actually give? Justify.

```
#include <stdio.h>
int main()  {
      if (sizeof(int) > -1)
            printf("Yes\n");
      else
            printf("No\n");
}
```

6. Write a C program that reads four integers and displays the pair with the largest sum. For example, if the user enters 10, -8, 17 and 5, the program should display `10 + 17 = 27`.

Next, you will get introduced to writing iterative constructs, i.e., loops. One of the loops used in C is the while loop whose syntax is shown below:

```
while (expr)    /* loop condition */
{
        stmt1;  ┐
        stmt2;  │
        …       ├ /* body of the loop */
        …       ┘
}
```

English equivalent of the loop shown on the right:

As long as *expr* evaluates to TRUE, then execute all the statements given in the body of the loop.

The loop will "run" until the condition evaluates to a false. Obviously, something has to be done in the body of the loop so that a component of the condition changes during the execution of the loop; otherwise, the loop will continue to run indefinitely, i.e., it will be an "infinite loop".

Example #1: Printing the first N natural numbers, one per line, where N is user input.

```
sum = 0;                /* let sum be zero, to begin with */
numb = 1;               /* start adding up from 1 */
while (numb <= N)       /* as long as numb has not exceeded N... */
{
      printf("%d\n", numb);   /*print out the number */
      numb++;           /* increment it to the next value */
}
printf("Bye\n");
```

You should understand that the condition is checked each time, and if the condition is TRUE (i.e., the number is smaller than N), then the loop continues; otherwise, control comes out of the loop and then goes to the next statement after the loop (in this case, the `printf("Bye\n");`statement.

Example #2: Printing only the odd numbers from 1 to N, where N is user input:

```
numb = 1;
while (numb <= N)       /* as long as numb has not exceeded N... */
{
      if (numb % 2 == 1)      /* If numb is an odd number... */
            printf("%d\n", numb);       /* print out its value */
      numb++;     /* numb is the next number */
}
```

1. (a) Would it have made any difference if you had written `++numb;` instead of `numb++;` in Example 2 shown above? Explain.

   (b) What would have happened if you missed the statement `numb++;` altogether?

2. (a) There could have been an alternative logic without using the % operator to print only the odd numbers of the first N natural numbers. Write a complete C program employing this logic.
   (b) Now modify your program so that it also computes the sum of all odd numbers between 1 and N and prints it out at the end.

3. Explain the effect of the following code:

```
int i;
while (i = 2)
{
      printf("Some even numbers: %d %d %d\n", i, i+2, i+4);
      i = 0;
}
```

Contrast this code with the following:

```
int i;
```

```
if (i = 2)
      printf("Some even numbers: %d %d %d\n", i, i+2, i+4);
```

Both pieces of code are logically wrong. The run-time effect of one of them is so striking that the error is easy to spot, whereas the other piece of wrong code has a subtle effect that is much harder to spot. Record the explanation in your notebook.

4. Write a program that reads in an integer value for n and then sums the integers from n to 2*n if n is nonnegative, or from 2*n to 3*n if n is negative.

5. Write a complete C program that will take one of the following options from the user, and performs jobs accordingly:

Option-1: Convert from degree Celsius to deg. Fahrenheit

Option-2: Convert from deg. Fahrenheit to deg. Celsius

Option-3: Quit the program execution

You must also ensure the user inputs a temperature value greater than –273.15 deg. C (for option-1) and –459.67 deg. F (for option-2), by repeatedly asking the user to input a valid temperature as long as it not within the limit.

Additional Practice Exercises:

6. Try running the program /home/share/hailstones whose source is found at this location: /home/share/hailstones.c. Copy the file into your current directory, compile and run the program using these inputs: (i) 45, and (ii) –21. Observe the output, and understand why it is called hailstone sequence. (Try searching on the web "Collatz conjecture").
Next, make a copy of the source code file as `lab7_hailstones_modified.c` and use the latter file for doing the following tasks.

a. Incorporate an input validation check so that the user is repeatedly prompted to enter a positive number (instead of the program stopping when a negative integer is received as input by the user).

b. Take two integers **a** and **b** as input from user. Modify the program so that it stops at the first possible occurrence of a hailstone number which falls in the range [a,b), after printing an appropriate message. The program prints all the hailstones generated thus far, and also the number of terms in the sequence thus far. In case none of the hailstones falls within the specified range, then entire sequence that is generated gets printed. Observe the working by running `/home/share/hailstones_modify` with the same input values you had used earlier: (i) 45 and (ii) –21. For 45, try these ranges: 50 – 60, 70 – 100.

7. The standard library function rand() is used to generate pseudo random numbers.

a. Generate and print 5 random numbers separated by a tab, using the rand() function inside a for loop. Observe the output. Run the program a few times repeatedly. What do you observe about the random numbers that are generated?

b. Now modify your program like this before the for loop you have written:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main()
{
 int i, n, seed;
 seed = time(NULL);
 srand(seed);
 // the for loop goes here...
```

Run the program a few times repeatedly. Now what do you observe about the random numbers that are being generated? Given below is the explanation for this.

On most C systems, writing `time(NULL)` gives the number of elapsed seconds since 1 January 1970. Using this value as the seed, we now use another function `srand()` to seed the random-number generator. Repeated calls to `rand()` will generate all the integers in a given interval, but in a mixed-up order. The value used to seed the random-number generator determines where in the mixed-up order `rand()` will start to generate numbers. If we use the value produced by `time()` as a seed, then the seed will be different each time we call the program, causing a different set of numbers to be produced.

c. Modify the above code to limit the random numbers only in the range [20,40), i.e. numbers greater than or equal to 20 and lesser than 40. (Hint: Use the modulo operator %) Copy the entire program into your notebook.

8. Until interrupted, the following code prints TRUE FOREVER on the screen repeatedly:

```
while (1)
     printf("  TRUE FOREVER  ");
```

Write a simple program that accomplishes the same thing, but use a for statement instead of a while statement. The body of the for statement should contain just the empty statement ";".