1 Loops

Loops are C constructs that are used to repeat the execution of a statement or a block of code. There are three kinds of loops in C, we will see them all now.

1.1 The `for` Loop

The `for` loop is perhaps the most widely used and the most versatile of the three kinds of loops in C. Its general form is the following:

```
for (<initializer>; <condition>; <increment>) {
  <loop-body>;
}
```

You do not have to have the braces if all you want to repeat is a single statement. However, I do reccomend using the braces in all cases, even in the case of a single statement.

The initializer part of the `for` loop is executed *only once* when the execution of the loop starts. After that, *before* the execution of the loop body, the condition is evaluated, and the body is executed if it is true. Execution of the loop ends if the condition is false. After every time the loop body is executed, the statement within the increment portion of the `for` loop is executed before the condition is checked again. This obviously continues as long as the condition evaluates to true.

As usual, here is a simple example demonstrating the use of the `for` loop:

```
#include <stdio.h>

int main(void) {
  int i;

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

  return 0;
}
```

If you type this in correctly and run, it will display the numbers from 1 to 10. This is quite a simple example. Make sure you understand exactly what goes on in this code before looking at the next example.

```
#include <stdio.h>

int main(void) {
  double e=0.0;
  double i;
  double factorial=1.0;

  for (i=1.0; i<=100.0; i=i+1.0) {
    e+=1.0/factorial;
    factorial = i*factorial;
```

```
  }

  printf("%1.15f\n", e);

  return 0;

}
```

Now, examine the code carefully, and try to figure out what it calculates. Then, type it in, and run it, and examine the result. If you still can not figure out what it does, then you might need serious help with mathematics.

1.2 The `while` Loop

The second type of loop we are going to examine is the `while` loop. Its for is as below:

```
while (<condition>) {
  <loop-body>;
}
```

Once again, the braces are not mandatory for loop bodies consisting of a single statement, but they are strongly reccomended. The `while` loop is quite simple; the condition is evaluated, and if it is true the loop body is executed. If the condition evaluates to false, the loop ends. Note that just like a `for` loop, it is possible for the body of a `while` loop not to execute at all. In fact, a `while` loop is exactly equiavlent to a `for` loop where the initializer and increment parts are left blank.

1.3 The `do` Loop

The `do` loop has the following general form:

```
do {
  <loop-body>;
} while(<condition>);
```

It is very similar to the `while` loop except for the fact that the condition is evaluated at the *end* of the loop rather than the beginning. This also means that a `do` loop is executed at least once.

1.4 The `break` and `continue` Statements

The `break` and `continue` statements are important to control the execution of loops. When a `break` statement is executed within a loop, execution of the innermost loop (there can be multiple nested loops, obviously) ends, and execution continues with the next statement following the loop. In other words, `break` means “get out of this loop”.

The `continue` statement is different. It does not end the execution of the loop, but it causes execution of the body of the loop to stop, and start at the next iteration. In other words, `continue` means “jump to the end of this loop”.

Here is an example demonstrating both statements:

```
#include <stdio,h>

int main(void) {
  int i=0;

  while (i < 100) {
    i=i+1;

    if (i % 3) {
      continue;
    }

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

    if (i > 50) {
      break;
    }

  }

  break;
}
```

2 The `switch` Statement

You can think of the `switch` statement as sort of a "multi-if" statement. Here is the general form:

```
switch (<integer-variable>) {
  case <value-1>:
    <statements>;
  break;
  case <value-2>:
    <statements>;
  break;
  default:
    <statements>;
  break;
}
```

The `switch` statement takes an integer value to switch on. The body of the `switch` statement contains any number of `case` labels with certain values. Once execution reaches the `switch` statement, execution will jump to the `case` label which matches the value of the `switch` variable. If there is no match, execution will jump to the optional `default` statement. If there is no default statement, execution will jump right over the `switch` statement as a whole.

As a matter of fact, you do not need to use the `break` statements. If you omit them, execution will fall right through and execute code after the next `case` label. This is, in fact a design error in C, so be careful not to lose any `break` statements.

Here is a function showing the use of the `switch` statement:

```
void print_month_name(int month) {
  switch (month) {
    case 1:
      printf("January");
    break;
    case 2:
      printf("February");
    break;
    case 3:
      printf("March");
    break;
    case 4:
      printf("April");
    break;
    case 5:
      printf("May");
    break;
    case 6:
      printf("June");
    break;
    case 7:
      printf("July");
    break;
    case 8:
      printf("August");
    break;
    case 9:
      printf("September");
    break;
    case 10:
      printf("October");
    break;
    case 11:
      printf("November");
    break;
    case 12:
      printf("December");
    break;
    default:
      printf("What month??");
    break;
  }
}
```

3 Exercise

Write a C program that reads a double called `x` from the keyboard, calculate e^x, and print it on the screen. (You should do this by modifying the example program in this handout.)