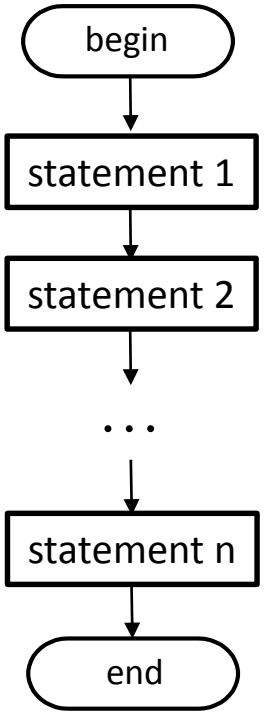
Chapter 5: Control Structures

In this chapter you will learn about:

- Sequential structure
- Selection structure
  - if
  - if … else
  - switch
- Repetition Structure
  - while
  - do… while
  - for
- Continue and break statements

Sequential Structure

- Statements are executed one by one until the end of the program is reached.




```
int main(void)
{
   int count = 0;
   printf("count = %d\n", count);
   count++;
   printf("count = %d\n", count);
   count++;
   printf("count = %d\n", count);
   return(0);
}
```

```
count = 1
count = 2
count = 3
Press any key to continue
```

Sequential Structure

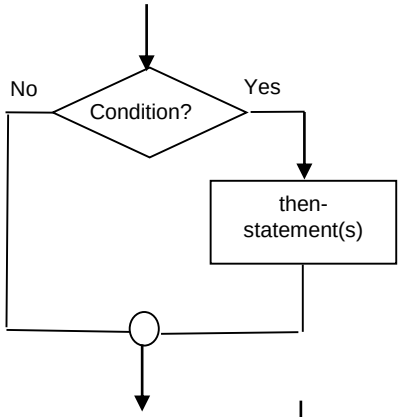
- Write a C program that asks the user to enter 3 integer numbers, and then prints the entered numbers in reverse order.

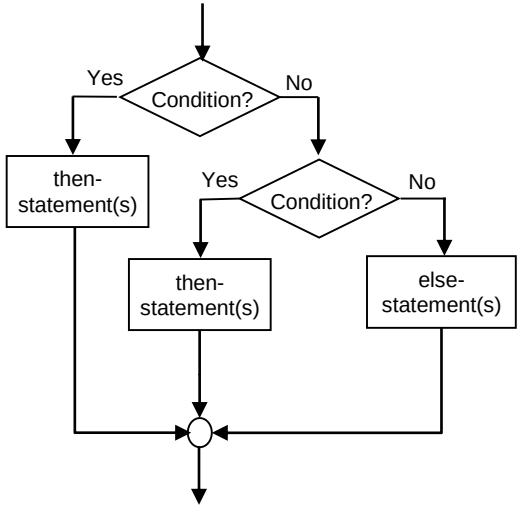
```
#include <stdio.h>

int main(void)
{
    int num1, num2, num3;

    // reading the input
    printf("Enter the first number: ");
    scanf("%d",&num1);

    printf("Enter the second number: ");
    scanf("%d",&num2);

    printf("Enter the third number: ");
    scanf("%d",&num3);

    /* printing the result to the screen */
    printf("The numbers in reverse order: %d %d %d \n",num3,num2,num1);
    return (0);
}
```

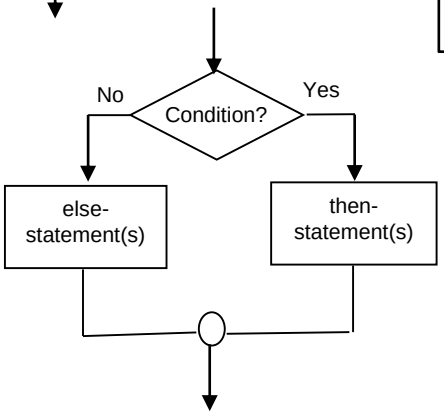
```
Enter the first number: 5
Enter the second number: 9
Enter the third number: 11
The numbers in reverse order: 11 9 5
Press any key to continue
```

Selection Structure

- In selection structure, the program is executed based upon the given *condition*.
- Only instructions that satisfy the given condition are executed.
- There are 3 types of selection structure:
  - if
    - A single alternative
  - if…else
    - Two alternatives
  - nested if..else
    - Multiple alternatives
  - switch
    - Multiple alternatives







Selection structure: *if*

- Syntax :

```
if (condition)
    Statement;
```

A condition is an expression that can return *true* or *false* (usually involving the use of an operator).

- The statement is only executed if the condition is satisfied.
- Example:

```
if (score >= 60)
  printf("Pass!!\n");
```

Note that there is no semicolon (;) after the *if* statement. If there is one, that means the *if* statement and the *printf()* statement are 2 different statements and they will both get executed sequentially.

- In the example above, the word "Pass!!" will only be printed out if the value of score is larger than or equal to 60. If not, the word "Pass!!" will not be printed out and the program will continue with the next statement.

```
#include <stdio.h>

int main(void)
{
   int score;

   printf("Enter the score: ");
   scanf("%d",&score);

   if (score >= 60)
    printf("Pass\n");

   printf("Bye\n");
   return(0);
}
```

```
Enter the score: 75
Pass
Bye
Press any key to continue
```

```
Enter the score: 20
Bye
Press any key to continue
```

Selection structure: *if... else*

- Syntax :

```
if (condition)
     statement1;
else
     statement2;
```

- If the condition is satisfied, statement1 will be executed. Otherwise, statement2 will get executed.
- Example :

```
if (score >= 60)
     printf("Pass!!\n");
else
     printf("Fail!!\n");
```

- In the above example, the word "Pass!!" will be printed if the value of score is bigger than 60 or equal to 60. Otherwise the word 'Fail!!" will be printed out.

```c
#include <stdio.h>

int main(void)
{
   int score;

   printf("Enter the score: ");
   scanf("%d",&score);

   if (score >= 60)
     printf("Pass\n");
   else
     printf("Fail\n");

   printf("Bye\n");

   return(0);
}
```

```
Enter the score: 75
Pass
Bye
Press any key to continue
```

```
Enter the score: 50
Fail
Bye
Press any key to continue
```

Plurality of Statements

- In the examples that we have seen so far, there is only one statement to be executed after the *if* statement.

- If we want to execute more than one statement after the condition is satisfied, we have to put curly braces **{ }** around those statements to tell the compiler that they are a part of the *if* statement, making it a *Compound Statement*

- ***Compound Statement*** **-** A group of statements that executed sequentially which is usually grouped by { }

Plurality of Statements

```
int score;

printf("Enter the score: ");
scanf("%d",&score);

if (score >= 60)
{
     printf("You have done very well\n");
     printf("I'll give you a present\n");
}
else
{
     printf("You have failed the course\n");
     printf("Sorry no present for you\n");
     printf("Go and study more");
}
```

Compound statement

```
Enter the score: 75
You have done very well
I'll give you a present
Press any key to continue
```

```
Enter the score: 25
You have failed the course
Sorry no present for you
Go and study more
Press any key to continue
```

What happen of you omit the { } for compound statement?

```
int score;

printf("Enter the score: ");
scanf("%d",&score);

if (score >= 60)
{
     printf("You have done very well\n");
     printf("I'll give you a present\n");
}
else
     printf("You have failed the course\n");
     printf("Sorry no present for you\n");
     printf("Go and study more");
```

```
Enter the score: 75
You have done very well
I'll give you a present
Sorry no present for you
Go and study more
Press any key to continue
```

What happen of you omit the { } for compound statement?

```
int score;

printf("Enter the score: ");
scanf("%d",&score);

if (score >= 60)
     printf("You have done very well\n");
     printf("I'll give you a present\n");
else
{
     printf("You have failed the course\n");
     printf("Sorry no present for you\n");
     printf("Go and study more");
}
```

Nested *if*... *else* statements

- A nested *if…else* statement is an *if…else* statement with another *if…else* statements inside it (multiple choice statement)
- Example :

```
if (score >= 90)
     printf("A\n");
else if (score >= 80)
     printf("B\n");
else if (score >= 70)
     printf("C\n");
else if (score >= 60)
     printf("D\n");
else
     printf("F\n");
```

- The **else if** statement means that if the above condition is not satisfied, then try checking this condition.If any one of the condition is already satisfied, then ignore the rest of the available conditions

Nested *if*... *else* statements

- Can you re-write nested if..else statement using multiple single if statements?
- It depends on the type of <condition> that we are dealing with.

Can you re-write nested if..else statement using multiple single if statements?

- Example 1:

```
if (score >= 90)
     printf("A\n");
else if (score >= 80)
     printf("B\n");
else if (score >= 70)
     printf("C\n");
else if (score >= 60)
     printf("D\n");
else
     printf("F\n");
```

```
Enter the score: 85
B
Press any key to continue
```

- Re-write using multiple single if

```
if (score >= 90)
     printf("A\n");
if (score >= 80)
     printf("B\n");
if (score >= 70)
     printf("C\n");
if (score >= 60)
     printf("D\n");
if (score < 60)
     printf("F\n");
```

Why?

```
Enter the score: 85
B
C
D
Press any key to continue
```

Can you re-write nested if..else statement using multiple single if statements?

- Example 2:

```
if (number == 1)
      printf("One\n");
else if (number == 2)
      printf("Two\n");
else if (number == 3)
      printf("Three\n");
else
      printf("Others\n");
```

```
Enter the score: 2
Two
Press any key to continue
```

- Re-write using multiple single if

```
if (number == 1)
      printf("One\n");
if (number == 2)
      printf("Two\n");
if (number == 3)
      printf("Three\n");
if (number < 1 && number > 3)
      printf("Others\n");
```

Why?

```
Enter the score: 2
Two
Press any key to continue
```

Selection structure: *switch*

- A *switch* statement is used to choose one choice from multiple cases and one default case.
- Syntax:

```
switch (variable)
{
    case case1:
       statement1;
      break;
    case case2:
       statement2;
       break;
    …
    default;
       statement;
       break;
}
```

The *break* statement is needed so that once a case has been executed, it will skip all the other cases and go outside the *switch* statement.

If the *break* statement is omitted, the execution will be carried out to the next alternatives until the next *break* statement is found.

switch - example

```
int number;

printf("Enter a positive integer number: ");
scanf("%d",&number);

switch (number)
{
    case 5:
          printf("Five!!\n");
          break;
   case 9:
          printf("Nine!!\n");
          break;
   case 13:
          printf("Thirteen!!\n");
          break;
   default:
           printf("Others\n");
}
```

This program reads a number from the user and print out the string equivalent for 5, 9, 13.

If the value being keyed in is other than 5, 9 or 13, the default statement will be executed where the statement "Others" will be printed out.

switch cont…

- The **value** for ‘case’ must be either in **integer** or **character** datatype.
  - Eg.1
```
switch (number) {
        case 5 :
             statement;
             break; ….
```
  - Eg.2
```
switch (color) {
        case 'R' :
             statement;
             break;
```

The value for case must be either in integer or character datatype

- The order of the ‘case’ statement is unimportant

switch example

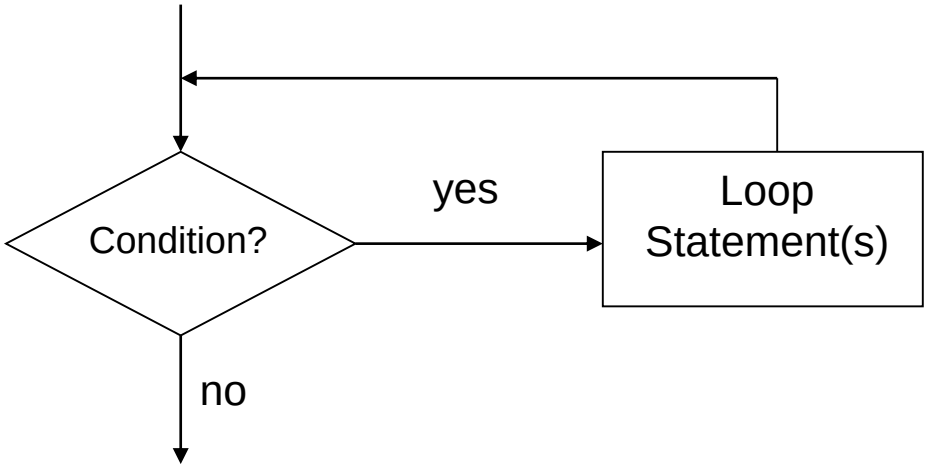
```
char grade;

printf("Enter the grade you scored for this subject: ");
scanf("%c",&grade);

switch (grade)
{
   case 'a':
   case 'A':
        printf("Excellent!!\n");
        printf("You brilliant..\n");
         break;
   case 'b':
   case 'B':
        printf("Job well done!!\n");
        printf("You deserve it..\n");
         break;
   case 'c':
   case 'C':
        printf("Oh no.. Just an average!!\n");
        printf("Try harder next time..\n");
         break;
   default:
         printf("undefined grade\n");
}
```

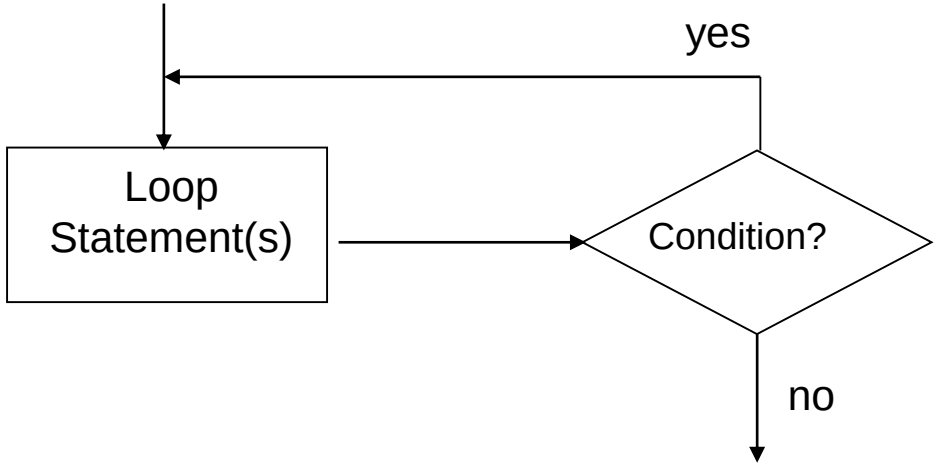
```
if (grade == 'a' || grade == 'A')
{
   printf("Excellent!!\n");
   printf("You brilliant..\n");
}
else if (grade == 'b' || grade == 'B')
{
   printf("Job well done!!\n");
   printf("You deserve it..\n");
}
else if (grade == 'c' || grade == 'C')
{
  printf("Oh no.. Just an average!!\n");
  printf("Try harder next time..\n");
}
else
 printf("undefined grade\n");
```

Repetition Structure (Loop)

- Used to execute a number of statements from the program more than one time without having to write the statements multiple times.
- Two designs of loop :
  - To execute a number of instructions from the program for a finite, **pre-determined number of time** (Counter-controlled loop) – recall the exercise from Topic 2.
  - To execute a number of instructions from the program **indifinitely until the user tells it to stop or a special condition is met** (Sentinel-controlled loop)

- There are 3 types of loops in C:
  - while
  - do…while
  - for





Repetition : *while* loop

- Syntax :

```
while (condition)
  statement;
```

Similar as in the *if* statement, the condition is an expression that can return *true* or *false*.

- As long as the condition is met (the condition expression returns true), the statement inside the *while* loop will always get executed.
- When the condition is no longer met (the condition expression returns false), the program will continue on with the next instruction (the one after the *while* loop).
- Example:

```
int total = 0;
while (total < 5)
{
      printf("Total = %d\n", total);
      total++;
}
```

Repetition : *while* loop cont…

- In this example :
  - **(total < 5)** is known as loop *repetition condition* (counter-controlled)
  - **total** is the loop *counter variable*
  - In this case, this loop will keep on looping until the counter variable is = 4. Once total = 5, the loop will terminate

```
int total = 0;
while (total < 5)
{
      printf("Total = %d\n", total);
      total++;
}
```

Repetition : *while* loop cont…

- The *printf()* statement will get executed as long as the variable **total** is less than 5. Since the variable total is incremented each time the loop is executed, the loop will stop after the 5th output.

```
int total = 0;
while (total < 5)
{
      printf("Total = %d\n", total);
      total++;
}
```

```
Total = 0
Total = 1
Total = 2
Total = 3
Total = 4
Press any key to continue
```

Example

- Write a program that will read 5 values from the user and prints the sum of the values.

```
#include <stdio.h>
int main(void)
{
   int number, count = 0, sum = 0;
   while (number < 5)
   {
    printf("Enter a number: ");
    scanf("%d",&number);

    sum = sum + number;
    number++;
   }
   printf("Sum = %d\n", sum);
   return(0);
}
```

```
Enter a number: 6
Enter a number: 4
Enter a number: -9
Enter a number: 3
Enter a number: 22
Sum = 26
Press any key to continue
```

Infinite loop

- If somehow the program never goes out of the loop, the program is said to be stuck in an **infinite loop**.
- The infinite loop error happens because the condition expression of the while loop always return a *true*.
- If an infinite loop occurs, the program would never terminate and the user would have to terminate the program by force.

What will be the output of the following programs?

```
int total = 0;

while (total < 5)
{
   printf("Total = %d\n", total);
}
```

```
int total = 0;

while (total < 5)
{
   printf("Total = %d\n", total + 1);
}
```

```
int total = 0;

while (total < 5)
{
   printf("Total = %d\n", total);
   total--;
}
```

Repetition : *do… while* loop

- Syntax

```
do {
   statement;
} while(condition);
```

- A *do…while* loop is pretty much the same as the *while* loop except that the condition is checked after the first execution of the statement has been made.
- When there is a *do…while* loop, the statement(s) inside it will be executed once no matter what. Only after that the condition will be checked to decide whether the loop should be executed again or just continue with the rest of the program.

do… while loop cont…

- Let us consider the following program:

```
int total = 10;

while (total < 10)
{
      printf("Total = %d\n", total);
      total++;
}
printf("Bye..");
```

- What does this program do?

  The program will only print the word "Bye..". The statements inside the while loop will never be executed since the condition is already not satisfied when it is time for the *while* loop to get executed.

do… while loop cont…

- Now consider the following program:

```
int total = 10;
do {
     printf("Total = %d\n, total);
     total++;
} while (total < 10)
printf("Bye..");
```

- Compared to the previous one, what will the output be?

  The program will get an output:

  Total = 10

  Bye..

  because the condition is not checked at the beginning of the loop. Therefore the statements inside the loop get executed once.

do… while loop cont…

- do.. while loop is very useful when it comes to data validation.
- Say for example we want to write a program that will read the score marks from the user, and print its equivalent grade (recall example from the selection section)
- Say that the score marks should be between 0 – 100. If the user keys in value other than 0 – 100, our program will need to do something.
  - Option 1: We could tell the users that they have entered a wrong input and terminate the program.
  - Option 2: We could tell the users that they have entered a wrong input and ask them to reenter the input.

Option 1: default case for if..else

```
int score;

printf("Enter the score: ");
scanf("%d",&score);

if (score >= 90 && score <= 100)
      printf("A\n");
 else if (score >= 80 && score < 90)
      printf("B\n");
 else if (score >= 70 && score < 80)
      printf("C\n");
 else if (score >= 60 && score < 70)
      printf("D\n");
 else if (score >= 0 && score < 60
      printf("F\n");
 else
       printf("Sorry, input should only be between 0 - 100 \n");
```

Option 2: do.. while (input validation)

```
int score;

do{
     printf("Enter the score: ");
     scanf("%d",&score);
     if (score < 0 || score > 100)
       printf("Sorry, input must be between 0 - 100\n");
}while (score < 0 || score > 100);

if (score >= 90 && score <= 100)
      printf("A\n");
 else if (score >= 80 && score < 90)
      printf("B\n");
 else if (score >= 70 && score < 80)
      printf("C\n");
 else if (score >= 60 && score < 70)
      printf("D\n");
 else
      printf("F\n");
```

```
Enter the score: -9
Sorry, input must be between 0 - 100
Enter the score: 150
Sorry, input must be between 0 - 100
Enter the score: 89
B
Press any key to continue
```

while vs do..while for input validation

do..while loop

```
do{
   printf("Enter the score: ");
   scanf("%d", &score);
   if (score < 0 || score > 100)
     printf("Sorry, input must be between 0 – 100\n");
}while (score < 0 || score > 100);
```

while loop

```
printf("Enter the score: ");
scanf("%d", &score);

while (score < 0 || score > 100)
{
   printf("Sorry, input must be between 0 – 100\n");
   printf("Enter the score: ");
   scanf("%d",&score);
}
```

Repetition : *for* loop

- Syntax :

```
for (expression1; expression2; expression3)
     statement;
```

  - Expression1: initialize the controlling variable
  - Expression2: the loop condition
  - Expression3: changes that would be done to the controlling variable at the end of each loop.

- Note that each expression is separated by a semicolon (;)

for loop - example

- Example:

```
int total;
for (total = 0; total < 5; total++)
{
    printf("Total = %d\n", total);
}
```

```
Total = 0
Total = 1
Total = 2
Total = 3
Total = 4
Press any key to continue
```

for loop vs *while* loop

```
int total = 0;

while (total < 5)
{
     printf("Total = %d\n", total);
     total++;
}
```

```
int total;

for (total = 0; total < 5; total++)
{
     printf("Total = %d\n", total);
}
```

for loop cont…

- Notice that the output is the same as the one for the *while* loop example. In fact, the two examples are exactly equivalent. Using a *for* loop is just another way of writing a *while* loop that uses a controlling variable.
- It is also possible to omit one or more of the *for* loop expressions. In such a case, we just put the semicolon without the expression.

```
int total= 0;
for (; total < 5; total++)
 {
      printf("Total = %d\n", total);
 }
```

Repetition Structure (Loop)

- Used to execute a number of statements from the program more than one time without having to write the statements multiple times.
- Two designs of loop :
  - To execute a number of instructions from the program for a finite, **pre-determined number of time** (Counter-controlled loop) – recall the exercise from Topic 2.
  - To execute a number of instructions from the program **indifinitely until the user tells it to stop or a special condition is met** (Sentinel-controlled loop)

Sentinel-controlled loop

- All this while you have seen what we call ‘counter controlled loop’ method.
- Counter control loop is used when we know beforehand how many iteration that the loop should execute.
- There will be cases where we (as the programmer) do not know how many times the loop should be executed, because the decision is up to the users.
- In this case, to terminate the loop, we need to use ‘sentinel controlled loop’ method

Sentinel-controlled loop

- In order to exit from the loop, the user must enter a unique data value, called a **sentinel value**.
- The sentinel value must be a value that could not normally occur as data.
- The algorithm for sentinel-controlled loop:

```
read a value
while the value is not sentinel value
    process the value
    read the next value
end_while
```

Sentinel-controlled loop

- Write a program that will read *n* values from the user and prints the sum of the values. The program stops reading values from the users when they enter ZERO.
  - In this example, the program task is to read values from the users and sum up then values.
  - The sentinel value for this example is ZERO, since ZERO is not part of the data value (anything + zero will not cause any changes to the value)

Sentinel-controlled loop

```
#include <stdio.h>
int main(void)
{
   int number, count = 0, sum = 0;

   printf("Enter a number [zero to end]: ");
   scanf("%d",&number);

   while (number != 0)
   {
        sum = sum + number;
        printf("Enter a number [zero to end]: ");
        scanf("%d",&number);
   }
   printf("Sum = %d\n", sum);
   return(0);
}
```

```
Enter a number [zero to end]: 3
Enter a number [zero to end]: -6
Enter a number [zero to end]: 10
Enter a number [zero to end]: 4
Enter a number [zero to end]: -7
Enter a number [zero to end]: 13
Enter a number [zero to end]: 24
Enter a number [zero to end]: 0
Sum = 41
Press any key to continue
```

continue and *break* statement

- Both of these statements are used to modify the program flow when a selection structure or a repetition structure is used.
- The *break* statement is used to break out of selection or repetition structure. For example:

```
for (a = 0; a < 5; a++)
{
      if (a == 2)
         break;
      printf("a = %d\n", a);
}
```

- The output of this example would be:

```
a = 0
a = 1
```

- When a = 2, the program will break out of the *for* loop due to the *break* statement. This will effectively terminate the loop. It will not wait till the value of **a** reaches 5 before terminating the loop.

continue and *break* statement

- The *continue* statement is used to ignore the rest of the statements in the loop and continue with the next iteration.
- Example:

```
for (a = 0; a < 5; a++)
{
      if (a == 2)
       continue;
      printf("a = %d\n", a);
}
```

- Output:

```
a = 0
a = 1
a = 3
a = 4
```

- a = 2 is not printed out because the loop skips the *printf()* function when the *continue* statement is encountered.

continue and *break* statement

- In a *while* and *do…while* structure, the loop condition will be checked as soon as the *continue* statement is encountered to determine whether the loop will be continued .
- In a *for* loop, any modification to the controlling variable will be done before the condition is checked.

SUMMARY

- In this chapter, you've learnt about 3 control structures in C programming :
  - Sequential
  - Selection
    - if..else
    - nested if..else
    - switch
  - Repetition
    - while
    - do…while
    - for
- Two designs of repetition :
  - Counter-controlled
  - Sentinel-controlled
- The use of continue and break statement

Exercise

- Write a program that reads positive integer numbers using repetition control-structure until the user terminates the program by entering zero. Your program should determine and print the **smallest** and **largest** of the supplied numbers. Please include appropriate input validation in your program.