

DON MARIANO MARCOS MEMORIAL STATE UNIVERSITY
LA UNION PHILLIPINES

ITCC 102 - Fundamentals of Problems Solving and Programming

1st Semester SY 2020-2021



1

Enrico Dacanay
Maricel Pre
Marylen Rodriguez
Emmalou Pimentel

Course Outline in Fundamentals of Problem Solving and Programming (ITCC 102)



COURSE DESCRIPTION

This course covers the use of general-purpose programming language to solve problems. The emphasis is to train students to design, implement, test and debug programs intended to solve computing problems using fundamental programming constructs.



COURSE OUTCOMES

By the end of this module, students should be able to:

1. design algorithms based on specified program specifications.
2. code and implement algorithms using a programming language each of the following fundamental programming components: (1) primitive data types, (2) basic computation, (3) simple I/O, (4) conditional and iterative structures, and (5) functions.
3. test and debug programs to determine the compliance of the program to specifications and desired results and improve its effectiveness and efficiency in addressing specifications.



COURSE REQUIREMENTS

1. Written assessments such as quizzes and major examination
2. Algorithm and flowcharting assignments
3. Programming assignments



GRADING SYSTEM

To pass this course, one must accumulate at least 75 percent through the course requirements discussed above. The weight for each requirement is shown below:

$$\text{Midterm Grade} = (\text{CS} \times .60) + (\text{Midterm Exam} \times .40)$$

$$\text{Final Term Grade} = (\text{CS} \times .60) + (\text{Final Exam} \times .40)$$

$$\text{Final Grade} = (\text{Midterm Grade} \times .40) + (\text{Final Term Grade} \times .60)$$

The following rubrics are likewise used in assessing your programming assignments:

Criteria	Outstanding	Very Satisfactory	Satisfactory	Fair	Poor
	(5)	(4)	(3)	(2)	(1)
Program Correctness	The application meets all the requirements specified in the project specification. The code is syntactically and logically correct for all cases. Implementation of the program follows the indicated guidelines and does not violate indicated restrictions. The implementation also exhibits the appropriate use of programming constructs.	The code works for typical input but fails for minor special cases; the major requirements are met, though some minor ones are not. Some implementation of the program violates indicated restrictions.	The code sometime fails for typical input. Many parts of the program implementation violate indicated restrictions and some parts of the solution are not implemented using appropriate programming constructs.	The code often fails, even for typical input. Most indicated restrictions were violated.	Programming does not run and/or implemented incorrectly based on specifications and restrictions).
Effective Communication / Concept Understanding	Answers to questions are correct, reasonable and reflective of the code. The justifications provided are sound.	Answers to questions are correct, but some justifications provided are weak.	Answers to questions are correct, but cannot justify solution (e.g. solution via trial and error, rather than the proper understanding and application of concepts.)	Correct understanding of the problem but was unable to explain workings of code provided.	Failure to explain and justify workings of the code submitted.

Readability	The program conforms to a coding standard that promotes code readability. Internal documentation is comprehensive.	Minor code formatting does not exhibit consistency in coding standard.	Not all functions or program features have proper internal documentation	Minimal internal documentation and code readability.	No internal documentation and code is not readable.
Efficiency and Reusability	The code is extremely efficient and can be reused.	The code is satisfactorily efficient and most of the codes can be reused.	The code is fairly efficient and many of the codes can be reused.	The code is brute force and some of the codes can be reused.	The code is huge and appears to be patched together and the code is not organized for reusability.
Delivery	The program was delivered on time.	The program was delivered 1-2 days after the due date.	The program was delivered 3-4 days after the due date.	The program was delivered 5 days after the due date.	The program was more than 5 days overdue.



COURSE CONTENT

Module I Introduction

- Lesson 1 Computer Fundamentals
- Lesson 2 Programming Languages
- Lesson 3 The C Programming Language
- Lesson 4 C Program Development Environment

Module II Program Development in C

- Lesson 1 Program Development Life Cycle
- Lesson 2 CodeBlocks
- Lesson 3 Creating C Program in CodeBlocks
- Lesson 4 Structure of a Basic C Program

Module III Input/Output Statements

- Lesson 1 Data Types, Keywords, Variables and Constant
- Lesson 2 Operators
- Lesson 3 Input Statements
- Lesson 4 Output Statements

Module IV Conditional Statements

- Lesson 1 If Statement
- Lesson 2 If...Else Statement
- Lesson 3 Nested If...Else Statement
- Lesson 4 Switch Statement

Module V Iteration Statements

- Lesson 1 For Statement
- Lesson 2 While Statement
- Lesson 3 Do...While Statement



REFERENCES

- C Tutorial*. (n.d). Retrieved June 3, 2019, from w3schools: <https://www.w3schools.in/c-tutorial/>
- Chavan, S. (2017). *C Recipes: A Problem-Solution Approach*. Maharashtra, India: Apress.
- Dacanay, E. (2020). *Fundamentals of Programming Using C Language*. College of Computer Science: Don Mariano Marcos Memorial State University, Agoo, La Union.
- Deitel, P., & Deitel, H. (2016). *C How To Program* (8th ed.). Hoboken, New Jersey, USA: Pearson.
- Learn-C: Free Interactive C Tutorial*. (n.d.). Retrieved June 3, 2019, from Learn-C Org: <https://www.learn-c.org/>

Nyhoff, J., & Nyhoff, L. (2017). *Processing: An Introduction to Programming*.

Tselikis, G., & Tselikas, N. (2014). *C From Theory to Practice*. Florida, USA: CRC Press.

Vector illustrations used from <https://tinyurl.com/y627rdk5>

Vector illustrations used from <https://www.vectorfair.com/>

Wiki for CodeBlocks. (n.d.). Retrieved June 3, 2019, from
codeblock.org:<http://codeblocks.org/>

MODULE 4 – CONDITIONAL STATEMENTS



In this chapter, you will be able to:

- Use the `if` selection statement and the `if...else` selection statement to select actions of the program based from the conditional expressions
- Use the `switch` statement to choose among many different actions of the program

Conditional Statements

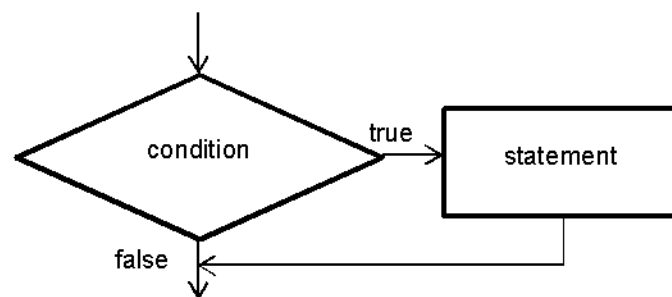
Normally, statements in a program are executed one after the other in the order in which they are written. This is called **sequential execution**. Various C statements will enable you to specify the next statement other than the next one in sequence. This is called **transfer of control** or **selection structure**. In order to do this, conditional statements are used.

Conditional statements are statements that check an expression then may or may not execute a statement or block of statements depending on the result of the condition. The **condition** uses relational, logical and equality operations that determines whether the expression is true or false.

C provides three types of selection structures or conditional statements. The **if selection statement** either selects (performs) an action if a condition is true or skips the action if the condition is false. The **if...else selection statement** performs an action if a condition is true and performs a different action if the condition is false. The **switch selection statement** performs one of many different actions, depending on the value of an expression. The `if` statement is called a **single-selection statement** because it selects or ignores a single action. The `if...else` statement is called a **double-selection statement** because it selects between two different actions. The `switch` statement is called a **multiple-selection statement** because it selects among many different actions.

The if Statement

Selection statements are used to choose among alternative courses of action. The flowchart below illustrates the single-selection if statement. This flowchart contains what is perhaps the most important flowcharting symbol—the diamond symbol, also called the decision symbol, which indicates that a decision is to be made. The decision symbol contains an expression, such as a condition, that can be either true or false. The decision symbol has two flow lines emerging from it. One indicates the direction to take when the expression in the symbol is true and the other the direction to take when the expression is false. Decisions can be based on conditions containing relational or equality operators.



The general form of the `if` statement is:

```
if (expression)
    statement;
```

where:

- expression is any relational, logical and equality expression that evaluates to a true (1) or false (0) value.
- statement may either be a single C statement or a block of statements.

A block starts with left brace and ends with right brace

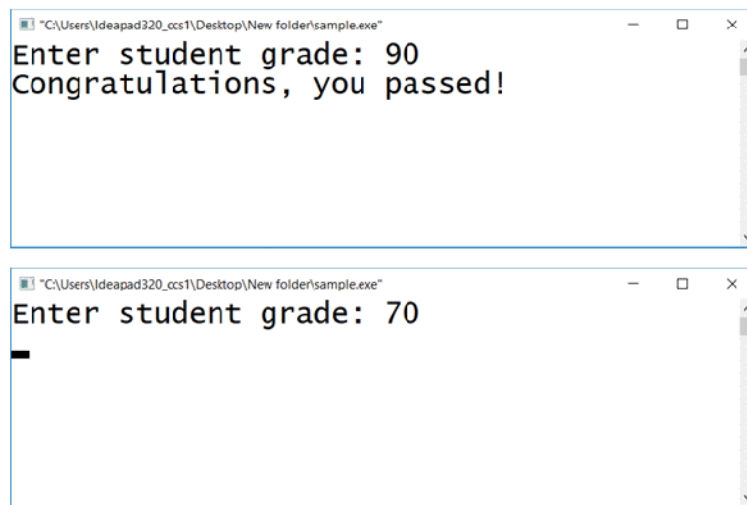
Example:

```
{
    statement1;
    statement 2;
    .....;
    statement_n;
}
```


Sample Program No. 1:

```
/*This program will display "Congratulations, you passed!"
if the student's grade is equal to or greater than 75*/
#include<stdio.h>
main()
{
    int studentGrade;
    printf("Enter student grade: ");
    scanf("%d", &studentGrade);
    if(studentGrade >= 75)
        printf("Congratulations, you passed!");
    getch();
}
```

Output:



Sample Program No. 2:

```
/*This program will ask for the total price purchased by
the buyer. It will also apply 20% discount if the buyer is
a senior citizen*/
#include<stdio.h>
main()
{
    int totalPrice, isSenior;
    float discount=0.0, finalPrice;
    printf("Enter total price: ");
    scanf("%d", &totalPrice);
    printf("Senior Citizen? (0-no; 1-yes): ");
    scanf("%d", &isSenior);
    if(isSenior == 1)
        discount = 0.20;
    finalPrice = totalPrice - totalPrice * discount;
    printf("Discount: %f\n", totalPrice * discount);
    printf("Total charge: %f", finalPrice);
    getch();
}
```

Output:

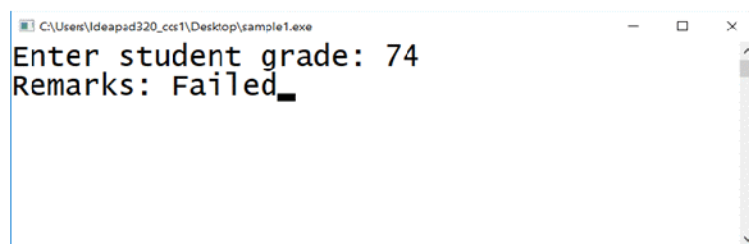
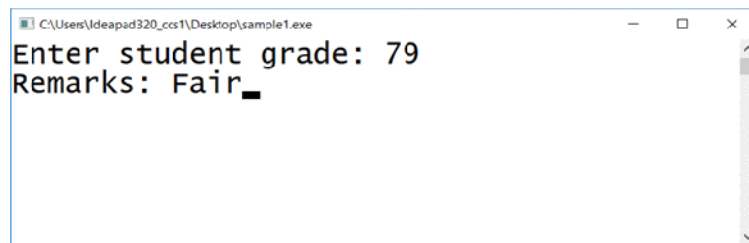
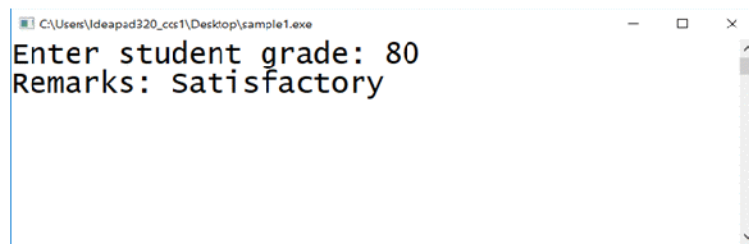
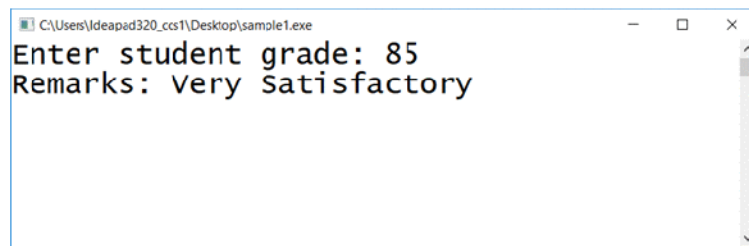
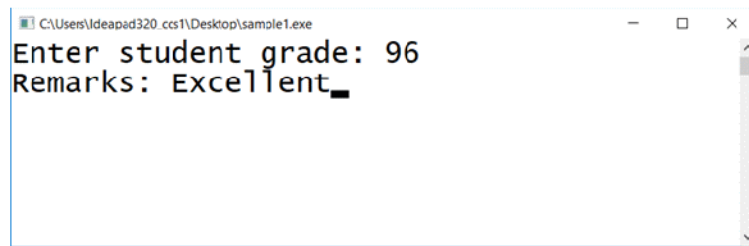
```
"C:\Users\Ideapad320.ccs1\Desktop\New folder\sample.exe"
Enter total price: 1250
Senior Citizen? (0-no; 1-yes): 0
Discount: 0.000000
Total charge: 1250.000000
```

```
"C:\Users\Ideapad320.ccs1\Desktop\New folder\sample.exe"
Enter total price: 1250
Senior Citizen? (0-no; 1-yes): 1
Discount: 250.000004
Total charge: 1000.000000_
```

Sample Program No. 3:

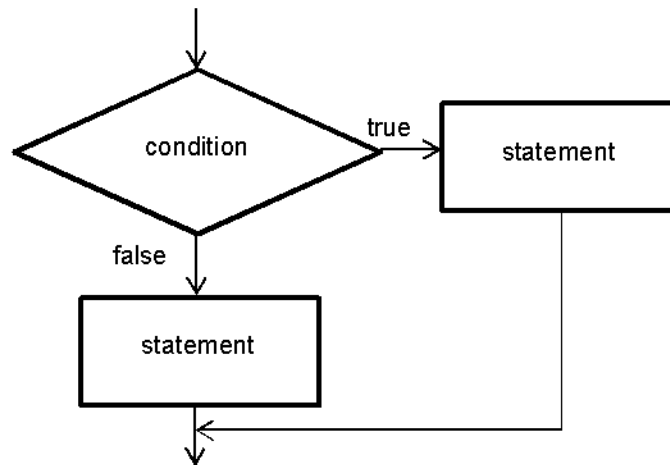
```
/*
This program will display the corresponding descriptive
rating for a particular grade of a student based from the
following values:
Numerical Rating      Descriptive Rating
95-100                Excellent
85-94                 Very Satisfactory
80-84                 Satisfactory
75-79                 Fair
74 and below          Failed
*/
#include<stdio.h>
main()
{
    int studentGrade;
    printf("Enter student grade: ");
    scanf("%d", &studentGrade);
    if(studentGrade >= 95 && studentGrade<=100)
        printf("Remarks: Excellent");
    if(studentGrade >= 85 && studentGrade<=94)
        printf("Remarks: Very Satisfactory");
    if(studentGrade >= 80 && studentGrade<=84)
        printf("Remarks: Satisfactory");
    if(studentGrade >= 75 && studentGrade<=79)
        printf("Remarks: Fair");
    if(studentGrade < 75)
        printf("Remarks: Failed");
    getch();
}
```

Output:



The `if...else` Statement

The `if` selection statement performs an indicated action only when the condition is true; otherwise the action is skipped. The `if...else` selection statement allows you to specify that different actions are to be performed when the condition is true and when it's false. The flowchart below illustrates the flow of control in the `if...else` statement.



The general form of the `if...else` statement is:

```
if (expression)
    statement;
else
    statement;
```

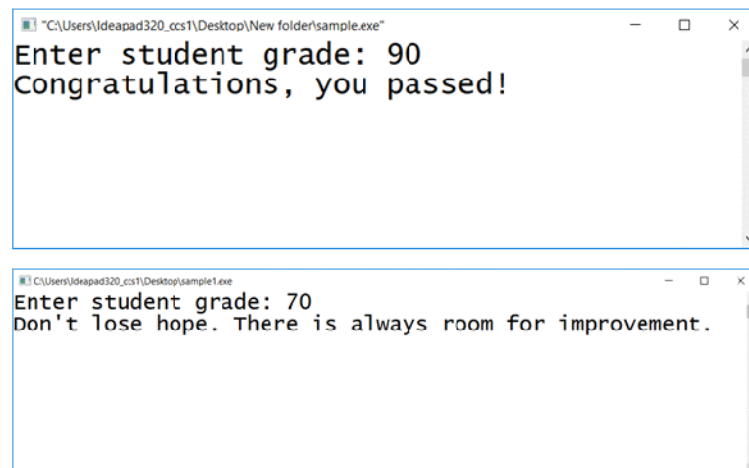
where:

- expression is any relational, logical and equality expression that evaluates to a true (1) or false (0) value.
- statement may either a single C statement or a block of statements. A block starts with left brace and ends with right brace

Sample Program No. 1:

```
/*This program will display "Congratulations, you passed!"
if the student's grade is equal to or greater than 75.
Otherwise, it displays "Don't lose hope. There is always
room for improvement" if the student grade is below 75.*/
#include<stdio.h>
main()
{
    int studentGrade;
    printf("Enter student grade: ");
    scanf("%d", &studentGrade);
    if(studentGrade >= 75)
        printf("Congratulations, you passed!");
    else
        printf("Don't lose hope. There is always room for
        improvement.");
    getch();
}
```

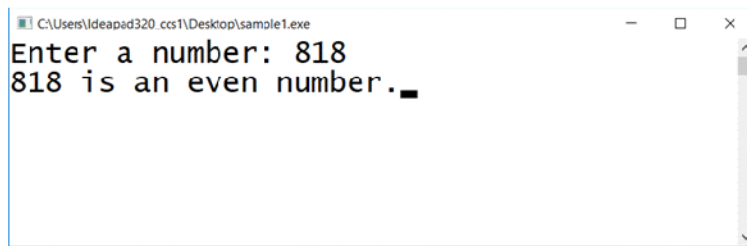
Output:



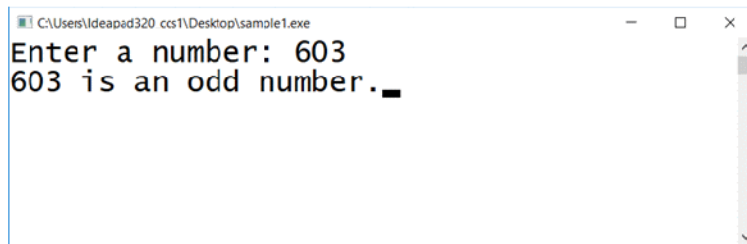
Sample Program No. 2:

```
/*This program determines if the input number is odd or
even number*/
#include<stdio.h>
main()
{
    int num, rem;
    printf("Enter a number: ");
    scanf("%d", &num);
    rem = num%2;
    if(rem==0)
        printf("%d is an even number.", num);
    else
        printf("%d is an odd number.", num);
    getch();
}
```

Output:



```
C:\Users\Ideaped320 ccs1\Desktop\sample1.exe
Enter a number: 818
818 is an even number.
```



```
C:\Users\Ideaped320 ccs1\Desktop\sample1.exe
Enter a number: 603
603 is an odd number.
```

The Nested if...else Statement

Nested if...else statements test for multiple cases by placing if...else statements inside if...else statements. This is sometimes referred to as “*an if within an if*.”

A nested if...else statement may be confusing because it can be difficult to know what else associates with what if. But C language provides a very simple rule for resolving this type of situation. The rule of thumb is the “**else is linked to the closest preceding if that does not already have an else statement associated with it.**”

Sample Program:

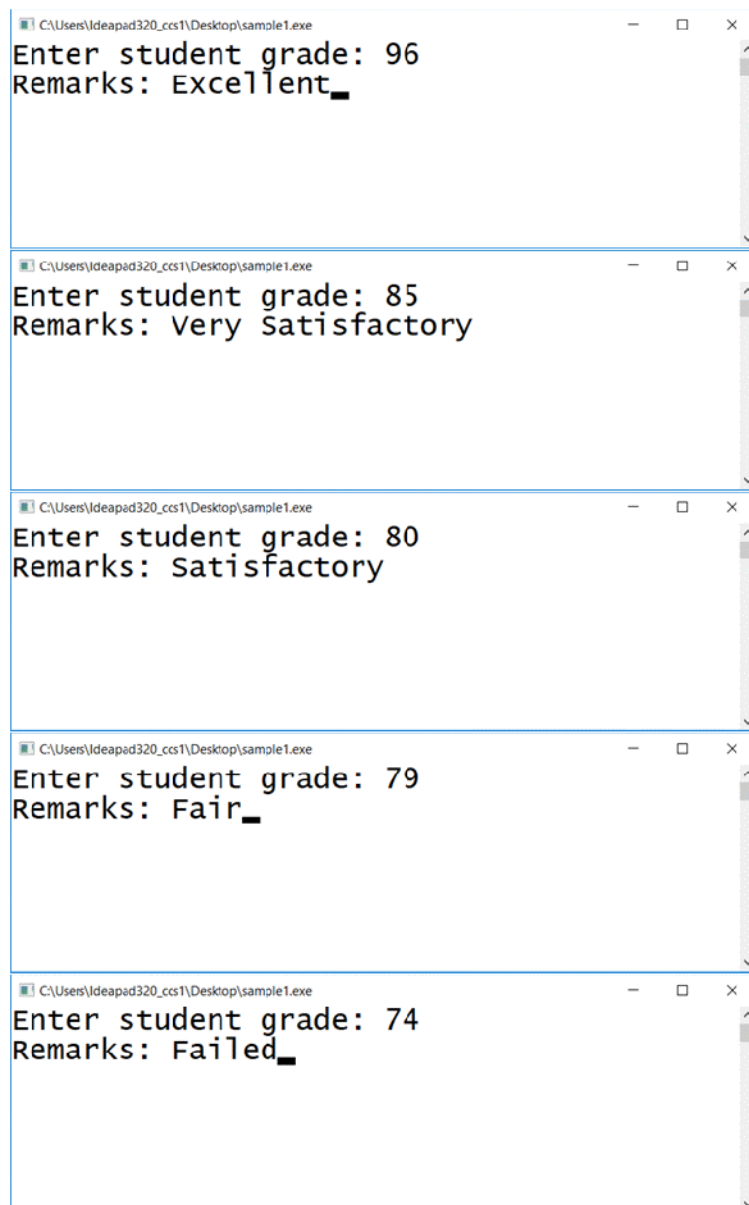
```
/*
This program will display the corresponding descriptive
rating for a particular grade of a student based from the
following values:
Numerical Rating      Descriptive Rating
95-100                Excellent
85-94                 Very Satisfactory
80-84                 Satisfactory
75-79                 Fair
74 and below          Failed
*/
#include<stdio.h>
main()
{
    int studentGrade;
    printf("Enter student grade: ");
```

```

scanf("%d", &studentGrade);
if(studentGrade >= 95 && studentGrade<=100)
    printf("Remarks: Excellent");
else if(studentGrade >= 85 && studentGrade<=94)
    printf("Remarks: Very Satisfactory");
else if(studentGrade >= 80 && studentGrade<=84)
    printf("Remarks: Satisfactory");
else if(studentGrade >= 75 && studentGrade<=79)
    printf("Remarks: Fair");
else if(studentGrade < 75)
    printf("Remarks: Failed");
getch();
}

```

Output:



The switch Statement

In the previous sections, the `if` single-selection statement and the `if...else` double selection statement are discussed. Occasionally, an algorithm will contain a series of decisions in which a variable or expression is tested separately for each of the constant integral values it may assume, and different actions are taken. This is called multiple selection.

C provides the `switch` multiple-selection statement to handle such decision making. The `switch` statement consists of a series of case labels, an optional default case and statements to execute for each case.

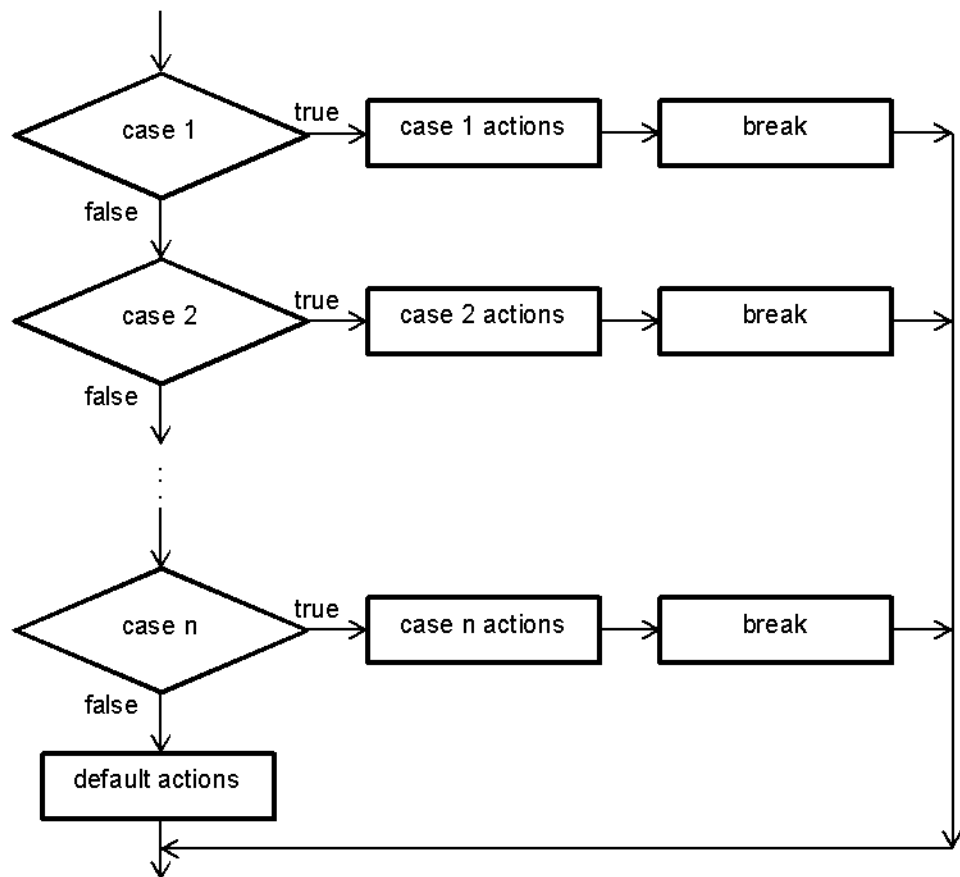
The general form of the `switch` statement is:

```
switch (variable)
{
    case constant1:
        statement ;
        break;
    case constant2:
        statement ;
        break;
    :
    :
    default:
        statement;
}
```

In a `switch` statement, a variable is successively tested against a list of integer or character constants. If a match is found, a statement or block of statements is executed. The default part of the `switch` statement is executed if no matches are found.

The `break` statement is used to terminate the statement sequence associated with each case constant. This means that it ignores other case constants and their respective statement sequence and jumps to the end of switch statement terminated by the symbol `}`.

The general `switch` multiple-selection statement (using a `break` in each case) is flowcharted in the following figure. The flowchart makes it clear that each `break` statement at the end of a case causes control to immediately exit the switch statement.



There are three important things to know about switch statements:

1. The `switch` statement differs from `if` statement in such a way that `switch` statement can only test for equality whereas `if` statement can evaluate a relational or logical expression.
2. No two case constants in the same `switch` can have identical values.
3. If character constants are used in the `switch` statement, they are automatically converted to their integer values.

Sample Program:

```
/*This program allows the user to select your phone brand and
displays a message.*/
#include<stdio.h>
main()
{
    char phone;
    printf("PHONE BRAND\n");
    printf("a. Apple\n");
    printf("b. Samsung\n");
    printf("c. Lenovo\n");
    printf("d. Asus\n");
    printf("e. Oppo\n");
    printf("f. Huawei\n");
    printf("g. None of the above\n");
    printf("Choose your phone brand: ");
    scanf("%c", &phone);
    switch(phone)
    {
        case 'a':
            printf("Wow!");
            break;
        case 'b':
            printf("Awesome!");
            break;
        case 'c':
            printf("Nice!");
            break;
        case 'd':
            printf("Great!");
            break;
        case 'e':
            printf("Elegant!");
            break;
        case 'f':
            printf("Cool!");
            break;
        case 'g':
            printf("I guess it's special!");
            break;
        default:
            printf("Invalid input...");
    }
    getch();
}
```

Output:

```
C:\Users\Ideapad320_ccs1\Desktop\sample1.exe
PHONE BRAND
a. Apple
b. Samsung
c. Lenovo
d. Asus
e. Oppo
f. Huawei
g. None of the above
Choose your phone brand: b
Awesome!_
```

```
C:\Users\Ideapad320_ccs1\Desktop\sample1.exe
PHONE BRAND
a. Apple
b. Samsung
c. Lenovo
d. Asus
e. Oppo
f. Huawei
g. None of the above
Choose your phone brand: k
Invalid input..._
```



SUMMARY

- C Language provides selection structures to control the flow of program.
- The three types of selection structures are single selection, double selection and multiple selection.
- The `if` statement is used for single selection, the `if...else` statement for double selection and the `switch` statement for multiple selection.



EXERCISES

Name: _____ Year & Section: _____ Score: _____

I. Evaluation

Directions: Give the output of the program below if the respective values are entered for variables `num1` and `num2`.

```
#include<stdio.h>
int num1, num2;
main()
{
    scanf("%d%d", &num1, &num2);
    if(num1 >= 0 && num2 <= 100)
    {
        if(num1 < 50 || num2 > 80)
            printf("Don Mariano Marcos");
        else
            printf("Memorial");
    }
    else
    {
        if(num2 >=0 && num2 <= 100)
            printf("State");
        else
            printf("University");
    }
}
```

OUTPUT

1. `num1 = 90; num2 = 80;` _____
2. `num1 = 50; num2 = 30;` _____
3. `num1 = -1; num2 = 200;` _____
4. `num1 = -1; num2 = 10;` _____
5. `num1 = 0; num2 = -1;` _____

Directions: Give the output of the following C program. Place your answer in the box provided.

```
#include<stdio.h>
main()
{
    int num1 = 7, num2 = 7;
    if(num1 == 1)
    if(num2 == 2)
    printf("%d\n", num1+num2);
    else
    printf("%d\n", num1-num2);
    printf("%d %d\n", num1, num2);
}
```





PROGRAMMING EXERCISES

Write a C Program for the following:

1. Input grade in each subject in BSCS First Year (see list below):

- Introduction to Computing
- Computer Programming 1
- Art Appreciation
- Purposive Communication
- Mathematics in the Modern World
- Kontekstwalisadong Komunikasyon sa Filipino
- Fundamentals of Physical Fitness

Then display the summary of numerical ratings and remarks of all subjects. Use the following table for the numerical rating for each grade.

1.00 = 98-100
1.25 = 95-97
1.50 = 92-94
1.75 = 89-91
2.00 = 86-88
2.25 = 83-85

2.50 = 80-82
2.75 = 77-79
3.00 = 75-76
5.00 = Failed
INC = Incomplete
D - Dropped

Sample Output:

```
ACADEMIC RECORD
*****
Introduction to Computing: 88
Computer Programming 1: 90
Art Appreciation: 95
Purposive Communication: 90
Mathematics in the Modern World: 70
Kontekstwalisadong Komunikasyon sa Filipino: 92
Fundamentals of Physical Fitness: 94
*****

SUMMARY
```

Subject	Grade	Numerical Rating	Remarks
Introduction to Computing	88	2.00	Passed
Computer Programming 1	90	1.75	Passed
Art Appreciation	95	1.25	Passed
Purposive Communication	90	1.75	Passed
Mathematics in the Modern World	70	5.00	Failed
Kontekstwalisadong Komunikasyon sa Filipino	92	1.50	Passed
Fundamentals of Physical Fitness:	94	1.50	Passed

2. A simple Body Mass Index (BMI) Calculator app. Create a BMI calculator application that reads the user's weight in pounds and height in inches (or, if you prefer, the user's weight in kilograms and height in meters), then calculates and displays the user's BMI. The formulas for calculating BMI are:

$$BMI = \frac{weightInPounds \times 703}{heightInInches \times heightInInches}$$

or

$$BMI = \frac{weightInKilograms}{heightInMeters \times heightInMeters}$$

Also, the application should display the following information from the Department of Health and Human Services/National Institutes of Health so the user can evaluate his/her BMI:

BMI VALUES

Underweight: less than 18.5

Normal: between 18.5 and 24.9

Overweight: between 25 and 29.9

Obese: 30 or greater

Sample Output:

```
Enter weight in kilograms: 67.5
Enter height in meters: 1.63
BMI: 25.21
Remarks: Overweight
```


3. A simple calculator app in C that allows the user to select an operation and enter two integers. It will then display the appropriate result.

Sample Output:

```
*****
CALCULATOR
*****
a. Addition
b. Subtraction
c. Multiplication
d. Division
e. Modulus Division
*****
Enter your choice: c
Enter first number: 10
Enter second number: 5
The product is 50.
```

4. A program that will display in words the number entered by the user. Consider numbers from 1 to 20 only.

Sample Output:

```
Enter a number: 16
The number in words is sixteen.
```

MODULE 5 – ITERATION STATEMENTS



In this chapter, you will be able to:

- Use the `for`, `while` and `do-while` iteration statements to execute statements several times depending on the given condition
- Provide the initialization, condition and the increment/decrement statements in order to execute statements depending on the problem given

Iteration Statements

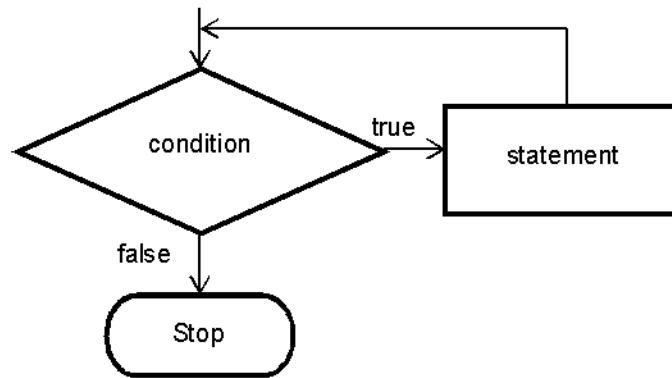
Most computer programs involve statements that need to be repeated. For example, displaying numbers from 1-10 is easily done in a single statement by typing each number, but to display numbers from 1-10,000 is a different matter. Typing numbers from 1 up to 10,000 is tedious and not practical. To solve this, C programming offered iteration (or looping) statements allowing a group of statements (or instructions) to be executed repeatedly either for a specific number of times or until it meets a certain condition. That is, as long as the condition evaluates to `True`, the group of statements (or instructions) is executed.

There are two types of iteration: **counter-controlled iteration** wherein the number of times a loop will execute (identified by a control/counter variable) is already known in advance and **sentinel-controlled iteration** wherein the precise number of iterations (or loops) is not known in advance.

The various iteration or looping statements used in C programming are: **for loop**, **while loop** and **do-while loop**. When using any of the iteration statements, **initialization** and **condition** must be provided. For the **for loop**, **increment/decrement** is also a must to avoid non-stop execution of a loop (sometimes called an infinite loop). **Initialization** is an assignment statement used to set the loop's counter. **Condition** is a relational Boolean expression determining when the loop will terminate. **Increment/Decrement** defines how the loop's counter will change each time the loop is executed.

The for Statement

The `for` statement or `for` loop is considered as a predefined loop. The number of times it executes a block of code is already predetermined in the loop's definition. The flowchart below illustrates the flow of control in the `for` statement.



The general form of the `for` statement is:

```
for(initialization; condition; increment/decrement)
{
    statement;
}
```

Sample Program No. 1:

```
/* This program will display numbers 1 to 5 */

#include<stdio.h>
main()
{
    int ctr;

    for(ctr=1; ctr<=5; ctr++)
    {
        printf("%d", ctr);
        printf("\n");
    }
    getch();
}
```

Program Output:



```
"C:\Users\Ideapad320_ccs2\Desktop\Iteration Statements\sample.exe"
1
2
3
4
5
```

Sample Program No. 2:

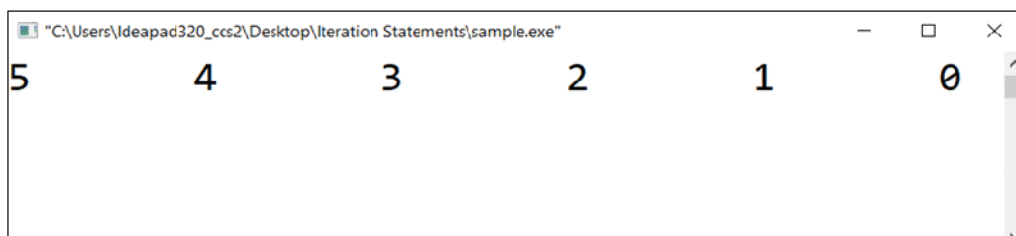
```
/* This program will display numbers
from 5 to 0 separated by a tab */

#include<stdio.h>

main()
{
    int ctr;

    for(ctr=5; ctr>=0; ctr--) {
        printf("%d", ctr);
        printf("\t");
    }
    getch();
}
```

Program Output:



```
"C:\Users\Ideapad320_ccs2\Desktop\Iteration Statements\sample.exe"
5      4      3      2      1      0
```

Sample Program No. 3:

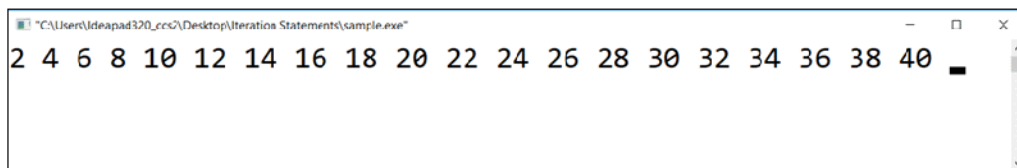
```
/* This program will display positive
even numbers 2 to 20 separated by a space */

#include<stdio.h>

main()
{
    int ctr;

    for(ctr=2; ctr<=40; ctr+=2) {
        printf("%d ", ctr);
    }
    getch();
}
```

Program Output:



Sample Program No. 4:

```
/* This program will display the square and
cube of numbers 1 to 10 */

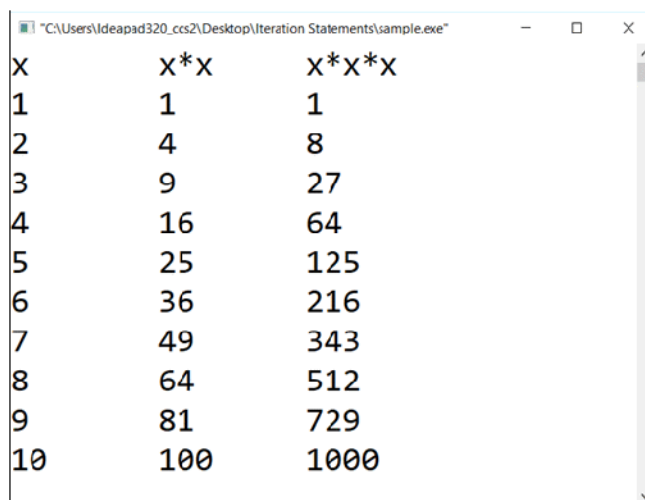
#include<stdio.h>

main()
{
    int ctr;

    printf("x\tx*x\tx*x*x");
    for(n=1; n<=10; n++) {
        printf("\n%d\t%d\t%d", n, n*n, n*n*n);
    }

    getch();
}
```

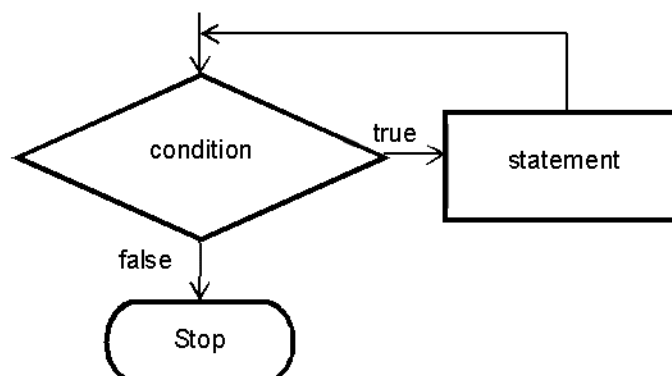
Program Output:



X	X*X	X*X*X
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1000

The while Statement

The `while` statement or `while` loop is considered to be a pre-test kind of loop as the condition is evaluated prior to execution of the block of codes. This means that the minimum number of iteration is zero in the case that the condition evaluates to `False`. Below illustrates the flow of control of the `while` statement which is similar with the `for` statement.



The general form of the `while` statement is:

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

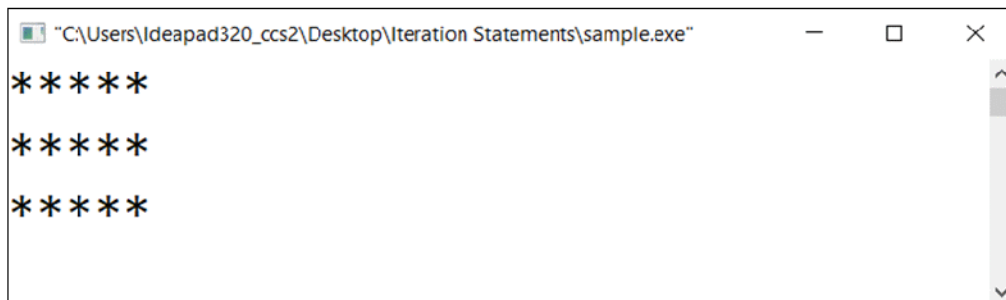
Sample Program No. 1:

```
/* This program will display the five asterisk
symbol in a line three times */

#include<stdio.h>
main()
{
    int ctr=1;

    while(ctr<=3) {
        printf("*****");
        printf("\n");
        ctr++;
    }
    getch();
}
```

Program Output:



```
*****
*****
*****
```

Sample Program No. 2:

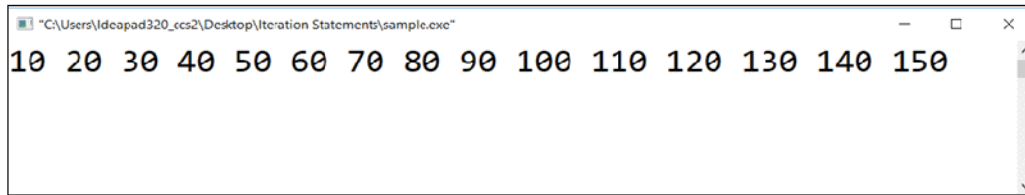
```
/* This program will skip count by 10's up
to 150 */

#include<stdio.h>
main()
{
    int n=10;

    while(n<=150) {
        printf("%d ", n);
        n+=10;    // or n=n+10;
    }
    getch();
}
```

```
}
```

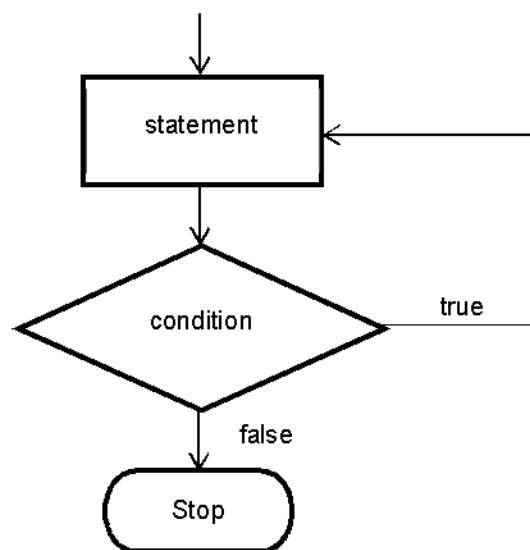
Program Output:



```
"C:\Users\Ideapad320_ccs2\Desktop\Iteration Statements\sample.exe"
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150
```

The do-while Statement

The do-while statement or do-while loop, in contrast with the while loop, is a post-test kind of loop as the block of codes is executed at least once before the condition is evaluated. Use this iterative statement in the case that you would like the block of codes to be executed at least once, afterwards determine whether the user wishes to repeat the execution or not.



The general form of the do-while statement is:

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


Sample Program No. 1:

```
/* This program will add two numbers and will
ask the user if he desires to add again. */

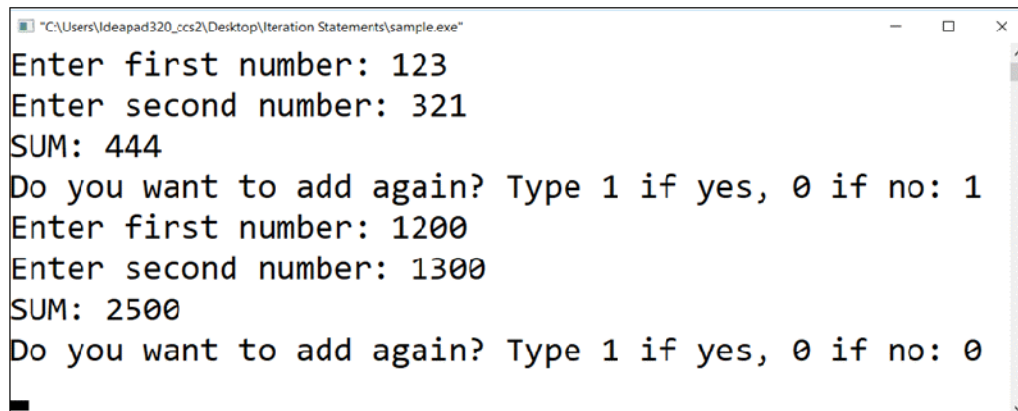
#include<stdio.h>
main()
{
    int n1, n2, choice;

    do {
        printf("Enter first number: ");
        scanf("%d", &n1);
        printf("Enter second number: ");
        scanf("%d", &n2);
        printf("SUM: %d", n1+n2);

        printf("\nDo you want to add again? ");
        printf("Type 1 if YES, 0 if NO: ");
        scanf("%d", &choice);
    } while(choice==1);

    getch();
}
```

Program Output:



```
"C:\Users\Ideapad320_ccs2\Desktop\Iteration Statements\sample.exe"
Enter first number: 123
Enter second number: 321
SUM: 444
Do you want to add again? Type 1 if yes, 0 if no: 1
Enter first number: 1200
Enter second number: 1300
SUM: 2500
Do you want to add again? Type 1 if yes, 0 if no: 0
```

Sample Program No. 2:

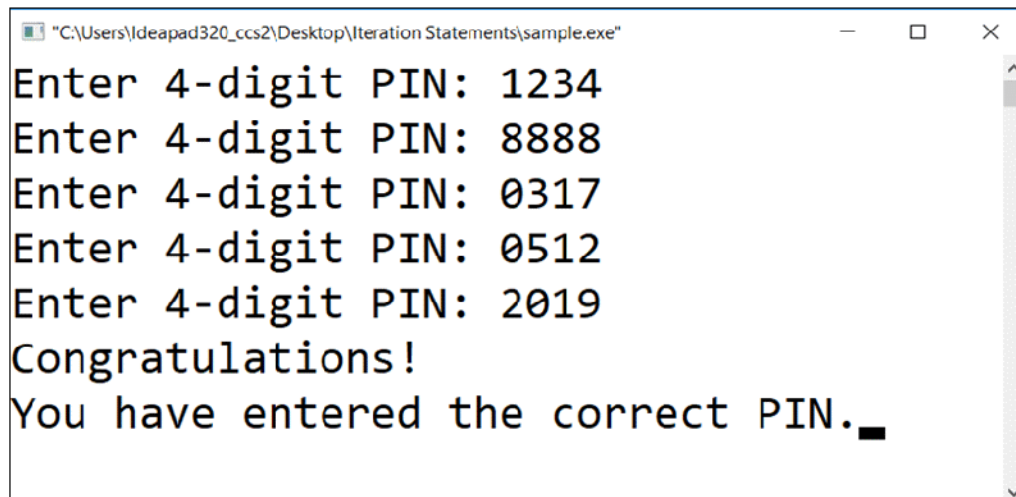
```
/* This program will ask for the 4-digit PIN
until the correct PIN is entered. The correct
PIN is 2019 */

#include<stdio.h>
#define correctPIN 2019
main()
{
    int pin;

    do {
        printf("Enter 4-digit PIN: ");
        scanf("%d", &pin);
    }while(pin!=correctPIN);

    printf("Congratulations!\n");
    printf("You have entered the correct PIN.");
    getch();
}
```

Program Output:

A screenshot of a Windows command prompt window. The title bar at the top reads "C:\Users\Ideapad320_ccs2\Desktop\Iteration Statements\sample.exe". The window contains the following text: "Enter 4-digit PIN: 1234", "Enter 4-digit PIN: 8888", "Enter 4-digit PIN: 0317", "Enter 4-digit PIN: 0512", "Enter 4-digit PIN: 2019", "Congratulations!", and "You have entered the correct PIN." followed by a cursor. A vertical scrollbar is visible on the right side of the window.

```
"C:\Users\Ideapad320_ccs2\Desktop\Iteration Statements\sample.exe"
Enter 4-digit PIN: 1234
Enter 4-digit PIN: 8888
Enter 4-digit PIN: 0317
Enter 4-digit PIN: 0512
Enter 4-digit PIN: 2019
Congratulations!
You have entered the correct PIN.█
```

Nested Iteration Statements

An iteration statement can be nested within another type of iteration statement. Nested iteration (or loops), are used to cycle through a matrix or tabular data and multi-dimensional arrays. Since a table is a matrix of rows and columns, iteration is needed to loop through the rows, then across the columns. This is sometimes referred to as “*a loop within a loop.*”

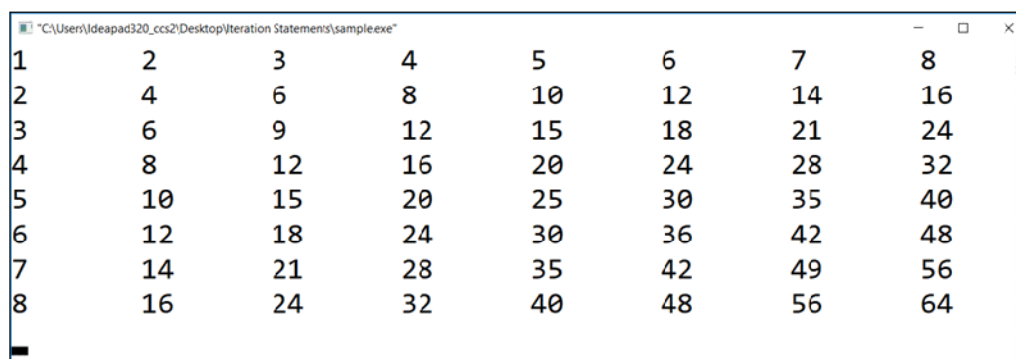
Sample Program No. 1:

```
/* 8x8 Multiplication table */

#include<stdio.h>
#define size 8
main()
{
    int row, col;

    for(row=1; row<=size; row++) {
        for(col=1; col<=size; col++) {
            printf("%d\t", row*col);
        }
        printf("\n");
    }
    getch();
}
```

Program Output:



1	2	3	4	5	6	7	8
2	4	6	8	10	12	14	16
3	6	9	12	15	18	21	24
4	8	12	16	20	24	28	32
5	10	15	20	25	30	35	40
6	12	18	24	30	36	42	48
7	14	21	28	35	42	49	56
8	16	24	32	40	48	56	64



Summary

- C Language provides iteration statements that allow a block of codes to be executed zero or more times, subject to the condition.
- The two types of iteration are counter-controlled iteration and sentinel-controlled iteration.
- The various iteration or looping statements used in C programming are the `for` loop, `while` loop and `do-while` loop.



EXERCISES

Name: _____ Year & Section: _____ Score: _____

I. Fill in the Blanks

Directions: Fill in the missing word/s to make the statement complete.

1. The _____ iteration statement specifies that a statement or group of statements is to be executed repeatedly while some condition remains true.
2. Iteration of a set of instructions a specific number of times is called _____ iteration.
3. When it's not known in advance how many times a set of statements will be repeated, a(n) _____ value can be used to terminate the iteration.
4. Iteration wherein the precise number of iterations (or loops) is not known in advance is called _____ iteration.
5. When using any of the iteration statements, _____ and condition must be provided.
6. _____ is a relational Boolean expression determining when the loop will terminate.
7. _____ defines how the loop's counter will change each time the loop is executed.
8. The minimum number of iteration of a while statement is _____.
9. Use the _____ statement in such cases you would like the block of codes to be executed at least once.
10. A non-stop execution of a loop is called a/n _____.

II. Application

Directions: Identify and correct the error/s in each of the following statements and rewrite the correct statements on the space provided.

```
1. x = 1;
   while (x <= 10);
   ++x;
   }
```

```
2. for (double y = .1; y != 1.0; y += .1) {
   printf("%f\n", y);
}
```

```
3. n = 1;
   while (n < 10) {
   printf("%d ", n++);
}
```

```
4. while ( c <= 5 ) {
   product *= c;
   ++c;
}
```

```
5. int x = 1, total;
   while ( x <= 10 ) {
   total += x;
   ++x;
}
```



PROGRAMMING EXERCISES

Write a C program that will accomplish each of the following tasks.

1. Sum the odd integers between 1 and 99 using a `for` statement. Use the unsigned integer variables `sum` and `count`.
2. Print the integers from 1 to 50 using a `while` loop and the counter variable `x`. Print only five integers per line. [Hint: Use the calculation `x % 5`. When the value of this is 0, print a newline character, otherwise print a tab character.]
3. Write a program that prints the following diamond shape. You may use `printf` statements that print either a single asterisk (*) or a single blank. Maximize your use of iteration (with nested `for` statements) and minimize the number of `printf` statements.

```
      *
     ***
    *****
   *********
  ***********
 * *********
* *******
* *****
* ****
* ***
* **
*
```

4. Write an improved program for a Calculator app. After displaying the result of the operation chosen, it will prompt the user whether if the user will try another operation. If the input is 'y', it will redisplay Screen 1 and proceeds with another operation selected by the user. If the input is 'n', the program terminates or stops execution.

```
*****
CALCULATOR
*****
f. Addition
g. Subtraction
h. Multiplication
i. Division
j. Modulus Division
*****
Enter your choice:
Enter first number:
Enter second number:
<display result here>

Try another one? (y/n):
```

5. Write a program prompt the user to enter grades in the different subjects

- Introduction to Computing
- Computer Programming 1
- Art Appreciation
- Purposive Communication
- Mathematics in the Modern World
- Kontekstwalisadong Komunikasyon sa Filipino
- Fundamentals of Physical Fitness

It will then display the summary of grades, numerical rating and remarks based on the following ratings.

Range	Numerical Equivalent	Remarks
98-100	1.00	Passed
95-97	1.25	Passed
92-94	1.50	Passed
89-91	1.75	Passed
86-88	2.00	Passed
83-85	2.25	Passed
80-82	2.50	Passed
77-79	2.75	Passed
75-76	3.00	Passed
Below 75	5.00	Failed

After displaying the summary, the application will prompt the user to choose whether to try again another record to be processed. If the input is 'y', the application will go back to Screen 1, else if the input is 'n', the application terminates or stops execution.

Sample Output:

Screen 1

```
ACADEMIC RECORD
*****
Introduction to Computing: 88
Computer Programming 1: 90
Art Appreciation: 95
Purposive Communication: 90
Mathematics in the Modern World: 70
Kontekstwalisadong Komunikasyon sa Filipino: 92
Fundamentals of Physical Fitness: 94
*****
```

Screen 2

```
SUMMARY
Subject                               Grade  Numerical Rating  Remarks
Introduction to Computing              88      2.00              Passed
Computer Programming 1                 90      1.75              Passed
Art Appreciation                      95      1.25              Passed
Purposive Communication                90      1.75              Passed
Mathematics in the Modern World        70      5.00              Failed
Kontekstwalisadong Komunikasyon sa Filipino 92      1.50              Passed
Fundamentals of Physical Fitness:      94      1.50              Passed

Do you want to try another one? (y/n) :
```