LECTURE 05

Selection

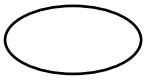
Outcomes

1. To understand how decisions are made in a computer using selection construct.
2. To write simple C program using `if-else` and `switch` statements.

Contents

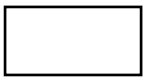
1. Programming Pseudocode and Flowchart

Before we go further to make decisions in C programming, we have to understand first about programming pseudocode and flowchart. We have to learn this because good programming practise is developed through good pre-code planning and organization. This is assisted by the use of pseudocode or program flowcharts. **Flowcharts** are written with program flow from the top of a page to the bottom. Each command is placed in a box of the appropriate shape, and arrows are used to direct program flow. The following basic shapes are often used in flowcharts:

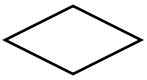


An oval indicates the **beginning** or **end** of a program.

A parallelogram is a point where there is **input** or **output** from the program.



A rectangle indicates **the assignment of a value** to a variable, constant, or parameter. The assign value can be **the result of computation**.



A diamond indicates a point where a **decision** is made.

Arrows indicate the **direction** and order of a program execution.

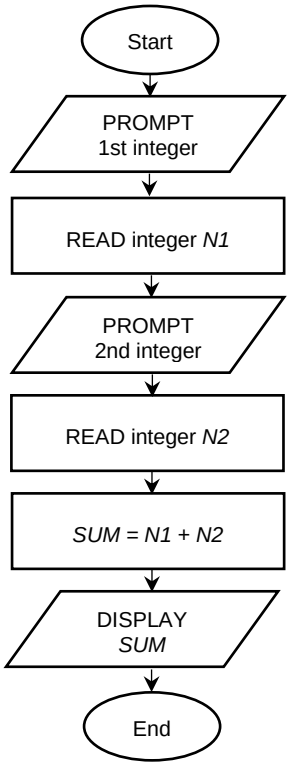
Circle can be used to **combine** flow lines.

On the other hand, **pseudocode** is a method of describing computer algorithms using a combination of natural language and programming language. It is essentially an intermittent step towards the development of the actual code. It allows the programmer to formulate their thoughts on the organization and sequence of a computer algorithm without the need for actually following the exact coding syntax. Although pseudocode is frequently used there are no set of rules for its exact implementation. In general, here are some rules that are frequently followed when writing pseudocode:

1. The usual C symbols are used for arithmetic operations (`+, -, *, /`).
2. Certain keywords are used, such as PRINT, WRITE, READ, etc. Other examples are:
   Input: INPUT or READ or GET
   Output: PRINT or DISPLAY or SHOW or PROMPT
   Compute: COMPUTE or CALCULATE or DETERMINE
   Initialize: SET or INIT
   Add one: INCREMENT or BUMP
   Decisions: TEST or IF/THEN/ELSE or WHILE/DO
3. Indentation (tab) should be used to indicate branches and loops of instruction.

Lets say we have a simple example. **Write a program that obtains two integer numbers from the user. It will print out the sum of those numbers**. What will you do to answer this question? The get a clear picture how to write a complete C program, I suggest you to write a flowchart or pseudocode in first place. However, if you are good enough to solve the problem and can picturized what the question need, you are allowed to not use this flowchart and pseudocode. Vice versa, the flowchart and pseudocode are used after writing programming code to explain those code. Usually, these flowchart and pseudocode are found in the report/thesis/journal etc. to overall visualized the programming code. The flowchart and pseudocode are very helpful for other people to understand your codes since it explains the flow of the program. It is very important rather than you copy all of your C codes which does not have any flow to explain. Below is the example of flowchart and pseudocode.

Flowchart



Pseudocode

Input: Two integers

Output: The sum of two integers

1. Start/Begin
2. PROMPT user for the first number
3. READ first integer *N1*
4. PROMPT user for the second number
5. READ second integer *N2*
6. Add *N1* and *N2* and set equal to *SUM* (*SUM = N1 + N2)*
7. DISPLAY the *SUM*
8. End

C Program

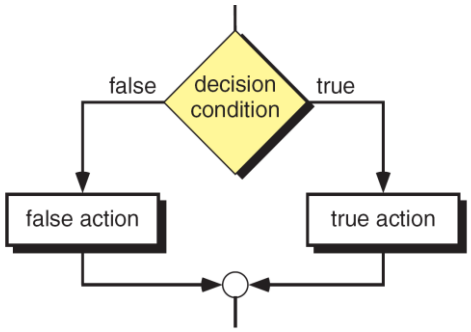
```
include<stdio.h>
int main()
{
int N1, N2, SUM;
printf("1st No");
scanf("%d",&N1);
printf("2nd No");
scanf("%d",&N1);
SUM = N1 + N2;

printf("%d",SUM);
return 0;
}
```

In the above example, to solve the question, we can see that there are eight (8) steps including start and end stages in flowchart and pseudocode. When you get this flowchart or pseudocode, it is a little easier for you to convert it to C program. For example, if we are using pseudocode method, we know that there are two integers for input and the sum of two integers is the output. Therefore, we declare three (3) `int` for `int N1` (input number 1), `int N2` (input number 2), and the summation of these two integers (`int SUM`). Then, we changed the PROMPT and DISPLAY (in stage 2, 4, and 7) to `printf` and READ (in stage 3 and 5) to `scanf`. At stage 6, we put equation `SUM = N1 + N2` since the question ask the summation of two inserted integers. This stage is the result of computation. You

can use any suitable wording for display output such as PRINT, DISPLAY, SHOW, or PROMPT. It is all up to you to choose those wording as long as it is understandable.

2. Two-Way Selection

The basic decision statement in the computer is the two-way selection. The decision is described to the computer as a conditional statement that can answered either **true** or **false**. If the answer is true, one or more action statements are executed. If the answer is false, then a different action is executed. The two-way selection decision logic is shown in the below figure.



The conditional statement can be any statement that gives either true or false answer. For example, `if (a == 0)`, `if (a > 0)`, `if (a < 0)`, `if (a == 5 && b == 1)`, `if (a > 10 || b < 30)`, and etc. For an easy understanding, I have two (2) examples to differentiate the two-way selection which are: (i) basic `if-else` statement and (ii) `if-else` statement (nested). Before we go further, I hope every one of you have familiar with the symbol of operators (that we have learned in the previous chapter) because the selection construct will use those operators in frequent.

`if` statement (single selection)

Before we go to the two-way selection, lets see how `if` statement for single selection is done. The basic syntax for `if` statement is if the answer of a condition is true, one or more action statements are executed. This syntax is simplified in the right fugure.

```
if Statement:
if (<condition>)
    statement 1
```

OR

```
if Statement:
if (<condition>) {
    statement 1
    …
    statement k
}
```

Question:
Develop a program for the following problem: A system to determine type of integer number.
The input for the system is any number in integer.
The output is: If the number is equal to 0 (zero), then display “Given number is zero”.

Following is the example of solution for this question.

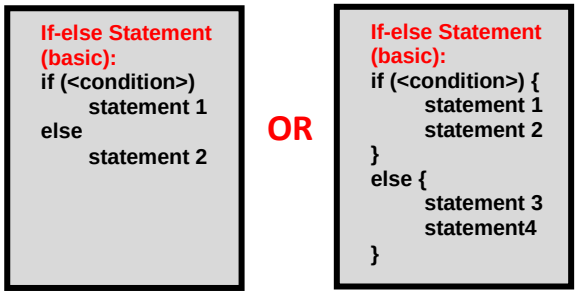
```
/*
C Program to demonstrate if statement
```

```
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{
      int number;
      printf("Enter  a number:\n");
      scanf("%d", &number);
      if (number == 0)
            printf("Given number is zero\n");
      return 0;
}
```

```
Enter a number
0
Given number is zero
```

(i) basic `if-else` statement

The basic syntax for `if-else` statement is if the answer of a condition is true, one or more action statements are executed. If the answer is false, then a different one or more actions are executed. This terminology is illustrated in the right fugure.



Question:

Develop a program for the following problem: A system to determine type of integer number.

The input for the system is any number in integer.

The output are: If the number is equal to 0 (zero), then display “Given number is zero”. If the number is not equal to 0, then display “Given number is not zero”.

Below is the example of solution for this question.

```
/*
C Program to demonstrate basic if-else statement
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{
```

```
        int number;
        printf("Enter  a number:\n");
        scanf("%d", &number);
        if (number == 0)
              printf("Given number is zero\n");
        else
              printf("Given number is not zero\n");
        return 0;
}
```

```
Enter a number:
27
Given number is not zero
```

(ii) `if-else` statement (nested)

When `if-else` is included within an `if-else`, it is known as `nested if-else` statement as shows in the right figure. There is no limit to how many levels can be nested, but if there are more than three, they become difficult to read.

if-else Statement (nested):

```
if (<condition>) {
    if (<condition>)
        statement
    else
        statement
}
else
        statement
```

Question:

Develop a program for the following problem: A system to determine type of integer number.

The input for the system is any number in integer.

The output are: If the number is equal to 0 (zero), then display “Given number is zero”. If the number is greater than 0, then display “Given number is positive”. Then, if the number is odd number, display “Given number is odd number”. If the number is even number, display “Given number is even number”. If the number is less than 0, then display “Given number is negative”.

Below is the example of solution for this question.

```
/*
C Program to demonstrate if-else statement (nested)
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{
        int number;
        printf("Enter  a number:\n");
```

```
	scanf("%d", &number);
	if (number == 0)
		printf("Given number is zero\n");
	else if (number > 0)
	{
		printf("Given number is positive\n");
			if (number%2 == 0)
				printf("Given number is even number\n");
		else
				printf("Given number is odd number\n");
	}
	else
		printf("Given number is negative\n");
	return 0;
}
```

```
Enter a number;
21
Given number is positive
Given number is odd number
```

3. Multi-Way Selection

C has two different ways to implement multiway selection. The first one is by using `else-if` or `if-else-if` (other name) statement. The other one is by using `switch` statement.

(i) `if-else-if` statement

The basic syntax for `if-else-if` statement for multiple choices is if the answer of a condition is true, one or more action statements are executed. If the answer is false, then a different condition is checked. If the answer of a condition is true one or more actions are executed. If the answer of condition is false, then another different condition is checked. This kind of statement is illustrated in the right fugure. For this implementation, you can use many `else-if` to check many conditions if you want.

if-else-if Statement (multiple choices):

```
if (<condition>) {
        statement
        statement
}
else if (<condition>)
        statement
else
        statement
```

Question:
Develop a program for the following problem: A system to determine type of integer number.
The input for the system is any number in integer.
The output are: If the number is equal to 0 (zero), then display “Given number is zero”. If the number is greater than 0, then display “Given number is positive”. If the number is less than 0, then display “Given number is negative”. If the number is not equal to 0 (zero), is not less than 0, and is not less than 0, then display “Given number is not zero, positive number and negative number”.
Following is the example of solution for this question.

```
/*
C Program to demonstrate if-else-if statement for multiple choices
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{
      int number;
      printf("Enter  a number:\n");
      scanf("%d", &number);
      if (number == 0)
            printf("Given number is zero\n");
      else if (number > 0)
            printf("Given number is positive\n");
      else if (number < 0)
```

```
            printf("Given number is negative\n");
      else
            printf("Given number is not zero, positive number and negative
            number\n");
      return 0;
}
```

```
Enter a number:
-90
Given number is negative
```

(ii) `switch` statement

`Switch` statement is using `case` and `default` syntax as illustrated in the right figure. It can replace the use of `if-else-if` statement for multiple choices condition but with several limitations.

Syntax:
```
switch (condition) {
        case condition1:
                statement1;
                break;
        case condition2:
                statement2;
                break;
        …
        default:
                statement3;
                break;
}
```

- The **`switch (condition)`** and **`case (condition1)`** must be only **integer** or **character** type. For example:

```
switch (car)     //valid char
switch (month)   //valid char
switch (1)       //valid int
case 1:          //valid int
case A:          //valid char
case 4.2:        //invalid float
```

- **Logical operators** cannot be used with switch statement. For instance:

```
case k>20 && k=20:    //invalid
```

To illustrate the usage of `switch` statement, lets have the following example.

Question:
Develop a program that converts a numeric score to a letter grade. The grading scales are:
Perfect = 100% A = 90-99% B = 80-89% C = 70-79% D = 60-69% F = less than 60%
Following is the example of solution for this question.

```
/*
C Program to demonstrate switch statement
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
```

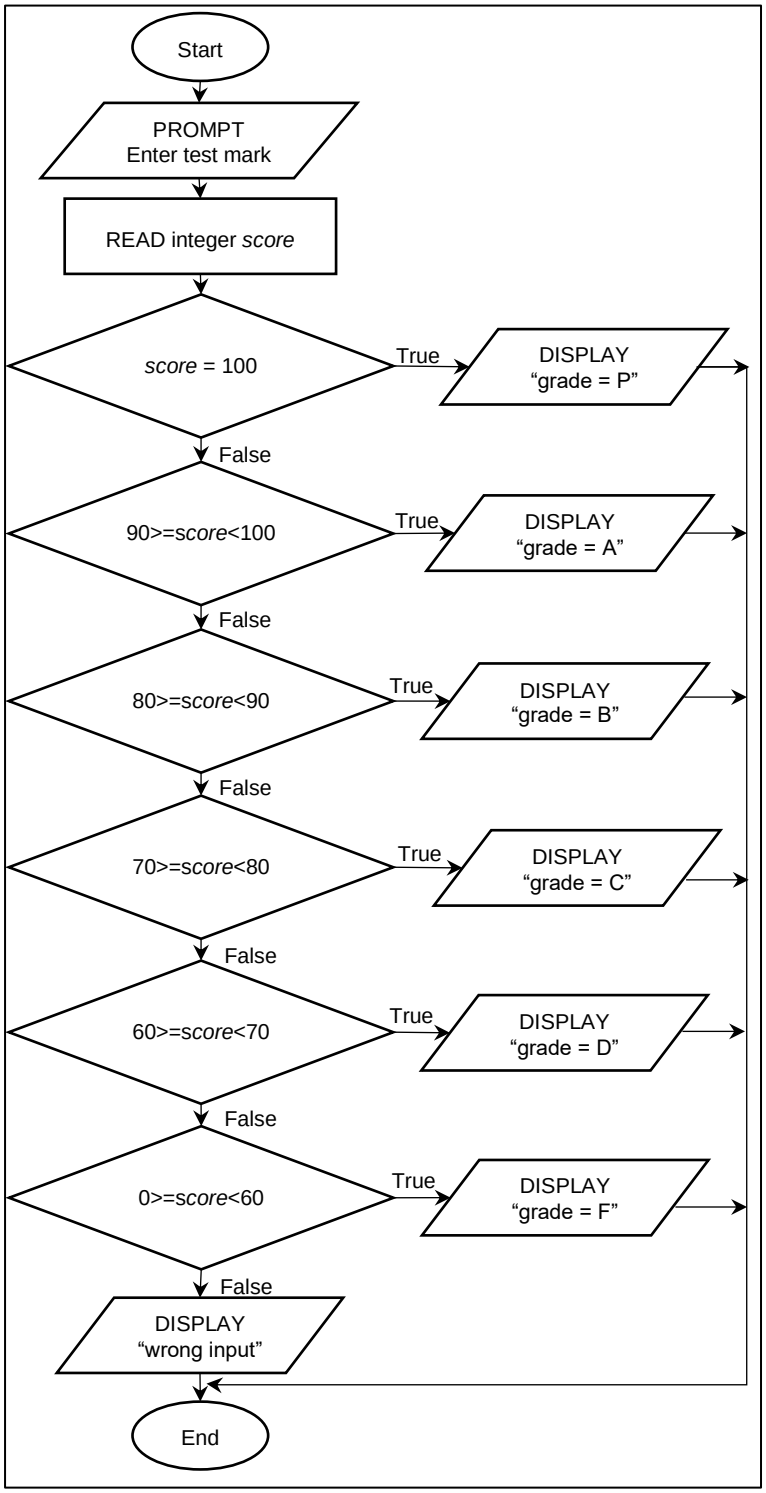
```
{
	int score, grade;
	printf("Enter test score (0-100): ");
	scanf("%d",&score);
	grade = score/100;
	switch (grade)
	{
		case 10: printf("\nThe grade is: Perfect\n");
			 break;   //if this break remove
		/*
		put break to close switch loop if case 10. If not, lets say
		case 10, it will execute case 10 and finish at case 9.
		*/
		case 9:  printf("\nThe grade is: A\n");
			 break;
		case 8:  printf("\nThe grade is: B\n");
			 break;
		case 7:  printf("\nThe grade is: C\n");
			 break;
		case 6:  printf("\nThe grade is: D\n");
			 break;
		case 5:  printf("\nThe grade is: F\n");
			 break;
		case 4:  printf("\nThe grade is: F\n");
			 break;
		case 3:  printf("\nThe grade is: F\n");
			 break;
		case 2:  printf("\nThe grade is: F\n");
			 break;
		case 1:  printf("\nThe grade is: F\n");
			 break;
		default: printf("\nWrong input. Please try again!\n");
			 break;
	}
	return 0;
}
```

Lets change this code to `if-else` statement.

```
/*
C Program to demonstrate switch statement vs. if-else
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{
      int score;
      printf("Enter test score (0-100): ");
      scanf("%d",&score);
      if (score == 100)
            printf("\nThe grade is: Perfect\n");
      else if (score >= 90 && score < 100)
            printf("\nThe grade is: A\n");
      else if (score >= 80 && score < 90)
            printf("\nThe grade is: B\n");
      else if (score >= 70 && score < 80)
            printf("\nThe grade is: C\n");
      else if (score >= 60 && score < 70)
            printf("\nThe grade is: D\n");
      else if (score < 60 && score >= 0)
            printf("\nThe grade is: F\n");
      else
            printf("\nWrong input. Please try again!\n");
      return 0;
}
```

You can try this two codes. How is the output? It is same right? Therefore, you can choose either to use `if-else-if` statement or `switch` statement if you have multiple choices to be computed. But again, you must be aware of the implementation `switch` statement because it has some restricted on evaluating condition which I have mentioned. How about the flowchart for this codes? Here is the flowchart and pseudocode for the above program for your reference.

Flowchart



Pseudocode

Input: Student Mark
Output: Mark Grade

1. Start/Begin
2. PROMPT user to enter test mark
3. READ mark *score*
4. if (*score* = 100)
   DISPLAY “grade P”
   end if
5. else if (90 >= *score* < 100)
   DISPLAY “grade A”
   end if
6. else if (80 >= *score* < 90)
   DISPLAY “grade B”
   end if
7. else if (70 >= *score* < 80)
   DISPLAY “grade C”
   end if
8. else if (60 >= *score* < 70)
   DISPLAY “grade D”
   end if
9. else if (0 >= *score* < 60)
   DISPLAY “grade F”
   end if
10. else
    DISPLAY “wrong input”
    end if
11. End