
PROGRAMMING
LANGUAGE
C++
PREPROCE
FUNCTIONS
OOPS
OPERATORS
BASICS
CONTAINERS
FUNCTION OVERLOAD
INHERITANCE
ARRAY
STRINGS

LET'S LEARN C++

"Talk Is Cheap Show Me The Code "

Author

Wan Salmizi bin Wan Mahmood

Zarina binti Musa

Fauziah binti Basok

BIODATA PENULIS

Wan Salmizi bin Wan Mahmood merupakan seorang pensyarah di Jabatan Teknologi Maklumat dan Komunikasi (JTMK), Politeknik Sultan Mizan Zainal Abidin (PSMZA), Dungun Terengganu. Beliau berasal daripada Kota Bharu, Kelantan dan merupakan pemegang Ijazah Kejuruteraan Elektrik (Teknologi Maklumat) dari Universiti Tun Hussein Onn Malaysia (UTHM). Beliau mula berkhidmat di PSMZA bermula tahun 2004 dan berpengalaman selama 18 tahun mengajar dalam bidang pengaturcaraan khususnya Pengaturcaraan C++ dan Asas Pengaturcaraan. Selain mengajar, beliau juga pernah terlibat menjadi penggubal soalan peperiksaan akhir peringkat Politeknik.

Zarina binti Musa berkhidmat sebagai pensyarah di Jabatan Teknologi Maklumat dan Komunikasi, Politeknik Sultan Mizan Zainal Abidin, Dungun Terengganu.Beliau berasal dari Bachok, Kelantan dan merupakan lulusan Ijazah Sarjana Muda Teknologi Maklumat dari Universiti Kebangsaan Malaysia (UKM). Mempunyai pengalaman mengajar dalam bidang Teknologi Maklumat hampir 15 tahun. Beliau juga merupakan penyelaras bagi kursus Programming Fundamentals di Jabatan Teknologi Maklumat dan Komunikasi(JTMK) dan berpengalaman hampir 10 tahun dalam bahasa pengaturcaraan c++.

Fauziah binti Basok merupakan seorang pensyarah yang sedang berkhidmat di Jabatan Teknologi Maklumat dan Komunikasi (JTMK), Politeknik Sultan Mizan Zainal Abidin, Dungun Terengganu. Beliau berasal dari Kota Tinggi, Johor dan merupakan lulusan Ijazah Sarjana Pendidikan Teknik Dan Vokasional dari Universiti Tun Hussein Onn Malaysia (UTHM). Berdasarkan pengalaman mengajar selama 15 tahun dalam bidang Teknologi Maklumat dan Komunikasi, beliau juga merupakan pensyarah bagi kursus pengaturcaraan C++ di Jabatan Teknologi Maklumat dan Komunikasi(JTMK). Selain daripada mengajar, beliau juga mempunyai pengalaman menghadiri beberapa kursus pengaturcaraan C++ bagi meningkatkan kemahiran dalam bidang dan pernah menjadi penceramah kursus persijilan professional w3chool bagi pengaturcaraan C++.

Writer and Editor by Wan Salmizi bin Wan Mahmood, Zarina binti Musa And Fauziah binti Basok

Graphics design by Wan Salmizi

Edition 2023.

Publisher:

POLITEKNIK MALAYSIA
SULTAN MIZAN ZAINAL ABIDIN

Polytechnic Sultan Mizan Zainal Abidin KM08,Jalan Paka,
23000 Dungun,Terengganu.
www.psmza.edu.my
Tel:09-8400800 I Fax: 09-8458781

PREFACE

Assalamualaikum w.b.t dan salam Malaysia madani...

Alhamdulillah, all praise to Allah s.w.t. for his blessing and opportunity for making 'Let's Learn C++ e-book' a reality. Not forgotten, this project wouldn't happen without constant support from fellow writers who have worked hard preparing this e-book since day one.

Through this e-book, users especially teachers and students can make reference to sources easier and faster no matter where they are and make this e-book more meaningful. Multimedia elements and interesting interface concepts make this e-book able to attract the attention of readers thus become an effective added value element in line with the development of the world of technology while internet at the fingertips.

Being hopeful and optimist, may this e-book become handy and accessible reference for teachers and students who intend to empower skill that interest them.

Author
Wan Salmizi Bin Wan Mahmood
Zarina Binti Musa
Fauziah Binti Basok

ABSTRACT

E-book production is one of the digital-based learning mediums that is being used worldwide. Learning through e-books becomes more interactive and able to attract readers. In line with the development of technology and a world without borders, there has been an increase in digital-based reading materials. This project aims to create an e-book titled "Let's learn C++". The development of this e-book has been produced through an interactive application that can be used by lecturers and students as users. In addition, the use of this application also gives freedom to users whether lecturers or students to get reliable reading resources regardless of time and place therefore able to access this e-book easily. This e-book is developed with several software such as Canva, Microsoft Office Word and Heyzin Flipbooks. It is hoped that this e-book will make it easier for students to develop their knowledge and understanding in addition to helping teachers to ensure that the teaching and learning process becomes more interactive and interesting.

Table of Contents

Chapter 1:

INTRODUCTION TO FUNDAMENTALS OF PROGRAMMING

History of C++ Programming

C++ an extension of C, was developed by Bjarne Stroudtrup in early 1980s.

C++ has become popular because the combination traditional C with OOP capability.

Object Oriented programs are easier to understand, correct and modify.

To give C++ instructions to computer, we need editor & compiler.

C++ PROGRAM BASIC STRUCTURE

```
// my first c++ program
# include <iostream>
using namespace std;
int main()
 {
 cout<<" I am smart student\n";
 cout<<"\n I want to score A in C++";
 return 0;
 }
```

COMMENT

PREPROCESSOR DIRECTIVES

HEADER FILES

MAIN() FUNCTION

RETURN STATEMENT

Comment

- Comment purpose is only to allow the programmer to insert notes or descriptions embedded within the source code.

- Single line Comment

```
#include <iostream>
using namespace std;
int main()  //main function
{
cout << "Hello World!";
return 0;//return statement
}
```

- Block/ Multi-line comment

```
/* The code below will print the words Hello World!
    to the screen, and it is amazing */
```

Preprocessor directives

- Include a library or some special instructions to the compiler about some certain terms used in the program

- Different preprocessor directives (commands) perform different tasks.

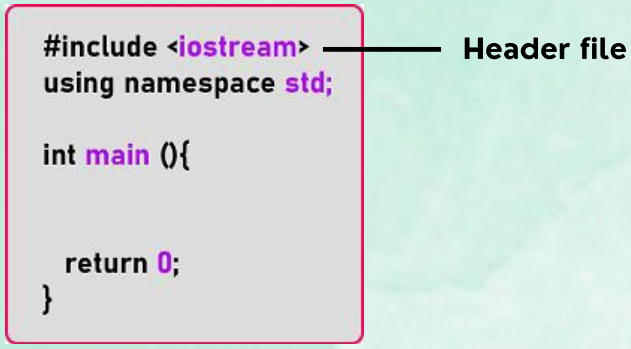
- It is a message directly to the compiler.
- Example:

```
#include <iostream>
#define PI 3.1425
```

Begin with # ——— **No semicolon (;)**

Header File

- Header file, sometimes known as an include file.
- For big projects having all the code in one file is impractical. But if we split the project into smaller files, we share between functions between them by using headers file.



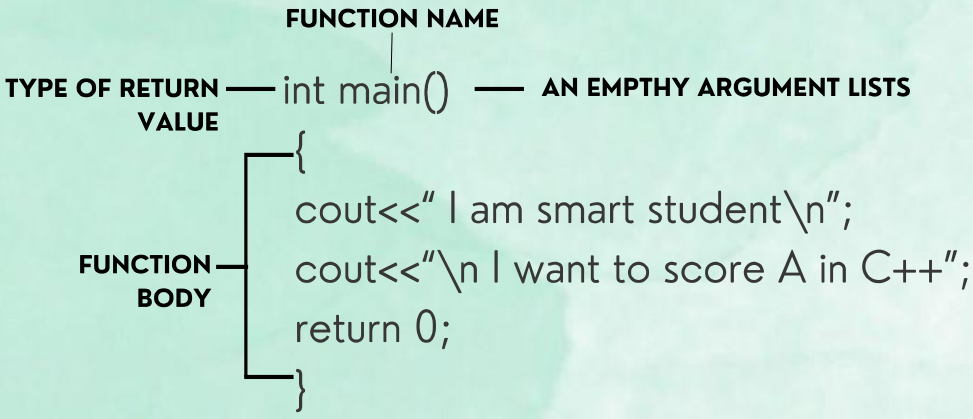
- Purpose header file to hold declarations for other files to use.

Using namespace std;

- All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name std.
- namespace is a collection of related names or identifiers (functions, class, variables) which helps to separate these identifiers from similar identifiers in other namespaces or the global namespace.

Main function and return statement

- Apart of C++ program and identify the start of the program. Exactly one function in a program must be main. The following is structure of main function:



- The return statement causes the main function to finish. Return may be followed by a return code (in our example is followed by the return code with a value of zero).

- A return code of 0 for the main function is generally interpreted as the program worked as expected without any errors during its execution.

IDENTIFIER AND DATA TYPES

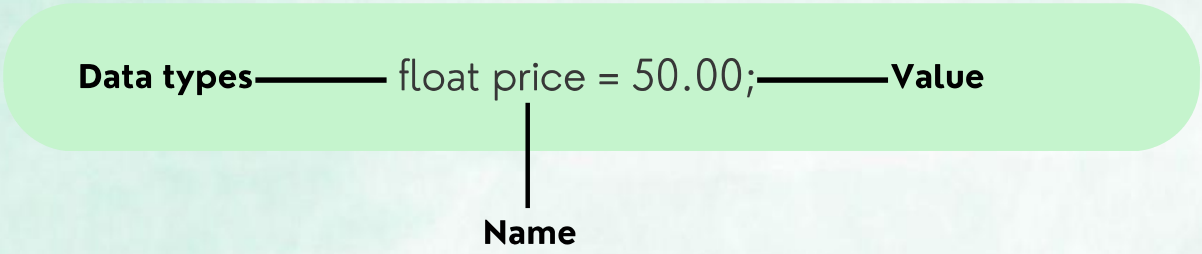
- Identifier is simply references to memory location which can hold data and used to represent variables, constants and name of function (sub-modules) in computer programs.

```
int number = 60; //good variable name
int n = 100; //OK, but not so easy to understand what
                is n actually
```

Variable

- Variable is an identifiers that refers to memory location, which can hold values.

- Variable only store one value at one time. Thus the content of variable may change during the program execution.



- Before use variables, it is must be declared. It's because to tells the compiler the types of data to store. Declaring a variable means specifying both its name and its data type.

- If the variables are more than one have same dat type, we can declare in one line separted with comma.

Constant

- An identifier whose value does not change throughout program execution.
- Allow to give name to a a value that is used several times in a program.
- 2 ways of declaring constant:
  a. Using constant keyword:

```
const float PI=3.142
```

  b. Using preproseccor directive:

```
#define PI 3.142
```

Rules for naming an identifier

Rules	Valid	Invalid
The 1st letter must be small alphabet or underscore	total_weight _name	1number 5_student
Can be a combination of alphabet letter, digit and underscore	ic_number	price$
No blank or white space characters.	ic_number	ic number

Cannot used c++ reserved word.

asm	auto	bool	break
case	catch	char	class
const	const_cast	continue	default
delete	do	double	dynamic_cast
else	enum	explicit	export
extern	false	float	for
friend	goto	if	inline
int	long	mutable	namespace
new	operator	private	protected
public	register	reinterpret_cast	return
short	signed	sizeof	static
static_cast	struct	switch	template
this	throw	true	try
typedef	typeid	typename	union
unsigned	using	virtual	void
volatile	wchar_t	while	

Data Types

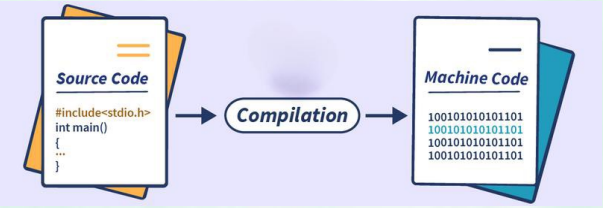
- Data types is classification of a particular type of information. Data types are essential to any computer programming langguage. Different data types have different sizes in memory depending on the machine and compilers.

Data Type	C++ keyword	Syntax	Bit Width	Example value	Range
integer	int	int no1,no2;	32	-1 ,1,34000	-32768-32767
long integer	long	long amount;	64	1000000	-4294967296 to 4294967296
short integer	short	short totalweight;	16	127	-127-128
unsigned integer	unsigned	unsigned sum;	32	200	0 to 65535
character	char	char name;	8	'z'	0 to 255
floating point	float	float amount;	32	20.00	Approximately 6 digits of precision
double floting point	double	double totalweight;	64	100.85321	Approximately 12 digits of precision
boolean	bool	bool found;	N/A	True/false	True/false

COMPILING AND DEBUGGING PROCESS AND ERROR IN PROGRAMMING

Compiling process

- A C++ must be compiled before it can be executed.
- To do this you need a compiler, which will take the source code and translate them into a form understood by the computer.
- Compiler have a stage where the original program (source code) is converted to something called object code which is used to package up procedures and libraries so that the program can executed successfully.



Debugging Process

- In computers, debugging is the process of locating and fixing or bypassing bugs (errors) in computer program code or the engineering of a hardware device.

- To debug a program or hardware device is to start with a problem, isolate the source of the problem, and then fix it.

Errors In Programming

Error	Descriptions	Common examples
Syntax errors/ Compile error	Grammar errors in the use of the programming language	✓ Misspelled variable and function names. ✓ Missing semicolons (;) ✓ Improperly matches parenthesesm square brackets[], and curly braces { } ✓ Incorrect format in selection and loop statements
Run time errors	Occur when a program with no syntax errors asks the computer to do something that the computer is unable to reliably do.	✓ Trying to divide by a variable that contains a value of zero ✓ Trying to open a file that doesn`t exist ✓ There is no way for the compiler to know about these kinds of errors when the program is compiled.
Logic errors	Logic errors occur when there is a design flaw in your program.	✓ Multiplying when you should be dividing ✓ Adding when you should be subtracting ✓ Opening and using data from the wrong file ✓ Displayin the wrong message ✓

Exercise Chapter 1

1. Write, compile and execute the program below. Don't forget to include the pre-processor command and also the header files. Correct the errors if any.

```
int main()

{
    cout<<"Welcome\nto\nC++!\n";
}
```

2. Write a simple C++ program that will produce an output like the one shown below.

```
C++  is a
very
interesting
language
```

3. Change the distance in miles to kilometer, where a mile is equal to 1.609 kilometer.

4. Identify the input, output and process, write the algorithm and pseudo code, and draw the flowchart to calculatethe area of a circle.

5. Identify the input, output and process, write the algorithm and pseudo code, and draw the flowchart to calculate the total of fine for late returning of library books. The rate is RM 0.20 per day.

6. Write a program that prints the numbers 1 to 4 on the same line. Write the program using the following methods.

a) Using one cout statement with no conversion specifiers.
b) Using one cout statement with four conversion specifiers.
c) Using four cout statements.

Chapter 2:

BASIC PROGRAM ELEMENT

IDENTIFIERS

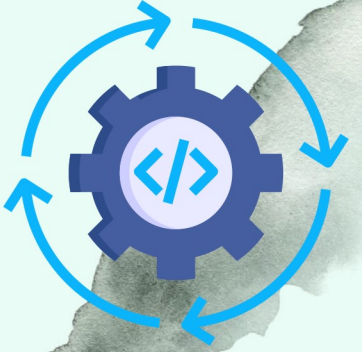
Identifier is simply references to memory location when can hold data and used to represent variables, consrants and name of function (sub-modules) in computer programs.

VARIABLES

- Variable is an identifiers that refers to memory location, which can hold values.
- Variable only store one value at one time.
- Thus the content of variable may change during the program execution. Before it is used, it must be declared with its data type.
- Syntax: data type variable_name
- **Example: int sum;**

  float price;

  char name;



- Single declaration:

  Example:

  int quiz1;

  int quiz2;

  int quiz3;

- **Multiple declaration**- variable having the **same data type** can be grouped and declared using the **single declaration** statement.

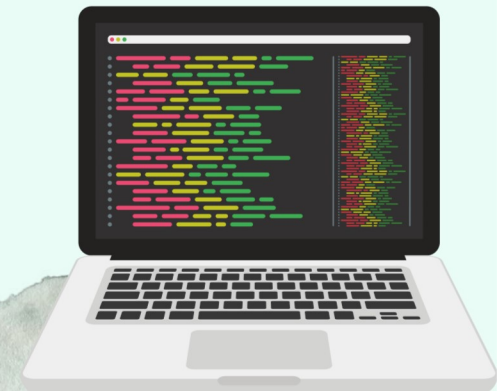
- For example, single declaration as shown above can be declared as a single statement

- Example:

  int quiz1,quiz2,quiz3;

  Initialize quiz1;

  quiz1=10;



INITIALIZE VARIABLE

- When declaring a regular local variable, its value is by default undetermined.
- There are two ways to do this in C++:
- known as c-like, is done by appending an equal sign followed by the value to which the variable will be initialized:

type identifier = initial_value ;

- For example, if we want to declare an int variable called a initialized with a value of 0 at the moment in which it is declared, we could write:

```
int a = 0;
```

```
// initialization of variables
#include <iostream>
using namespace std;
int main ()
{
int a=5; // initial value = 5
int b(2); // initial value = 2
int result; // initial value undetermined
a = a + 3;
result = a - b;
cout << result<< endl;
cout<<"result";
        return 0;
}
```

Output:
6
result

PROBLEM OF UNINITIALIZED VARIABLES

A variable that has not been given a known value (usually through initialization or assignment) is called an uninitialized variable.

If the memory values are not defined by the user at the start of the code's execution, the CPU will set the variable value to anything that is acceptable in computer programming language, this is usually termed as garbage value.

If a garbage value is set for a variable, then the whole logic of the program changes and will result in an incorrect value as the output.

EXAMPLE

```
#include <iostream>
using namespace std;

int main()
{

 int a,b,c,d,e,f,g,h,i,j;

 cout<<a<<endl;
 cout<<b<<endl;
 cout<<c<<endl;
 cout<<d<<endl;
 cout<<e<<endl;
 cout<<f<<endl;
 cout<<g<<endl;
 cout<<h<<endl;
 cout<<i<<endl;
 cout<<j<<endl;

 return 0;
}
```

OUTPUT :

```
4233342
7077712
4233248
1972793325
7077560
-2
-440363777
1972817088
7077836
7077560
```

KEYWORD

Keywords (also called reserved words)

Reserved words in the C++ language for specific use.

Keywords that identify language entities such as **statements, data types, language attributes**, etc.

Have **special meaning to the compiler**, cannot be used as identifiers (variable, function name).

Should be typed in **lowercase**.

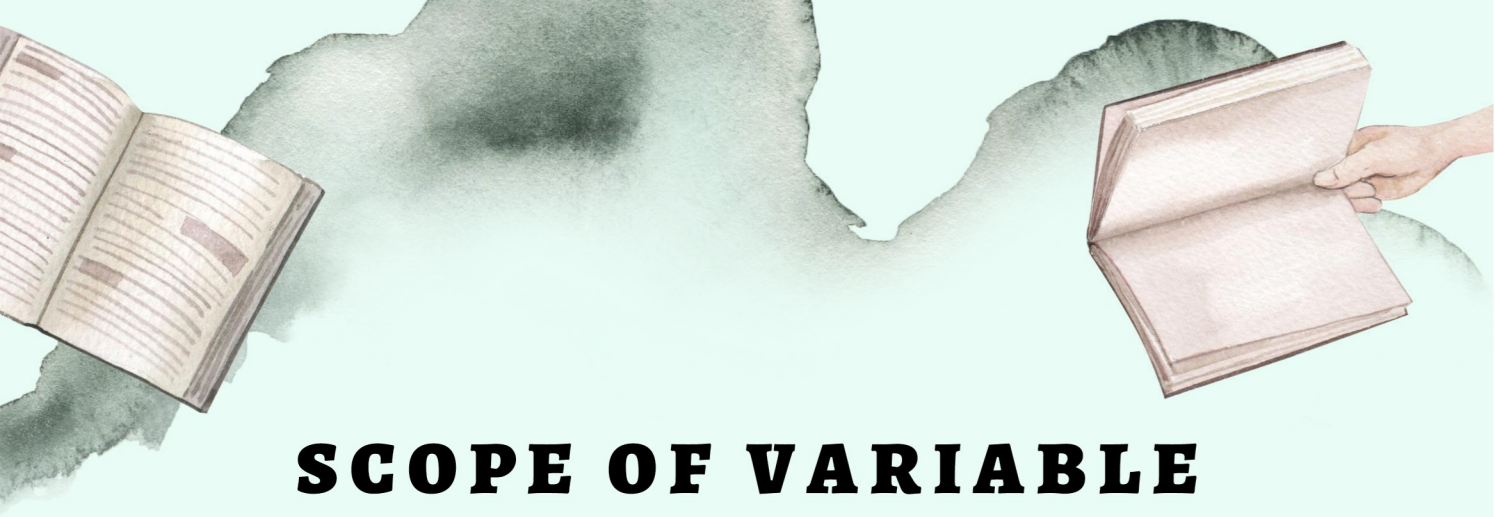
Example: const, double, int, main, void,printf, while, for, else **(etc..)**

KEYWORDS COMMON TO THE C AND C++PROGRAMMING LANGUAGES

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

C++ ONLY KEYWORDS

asm	bool	catch	class
const_cast	delete	dynamic_cast	explicit
false	friend	inline	mutable
namespace	new	operator	private
protected	public	reinterpret_cast	static_cast
template	this	throw	true
try	typeid	type name	using
virtual	wchat_t		

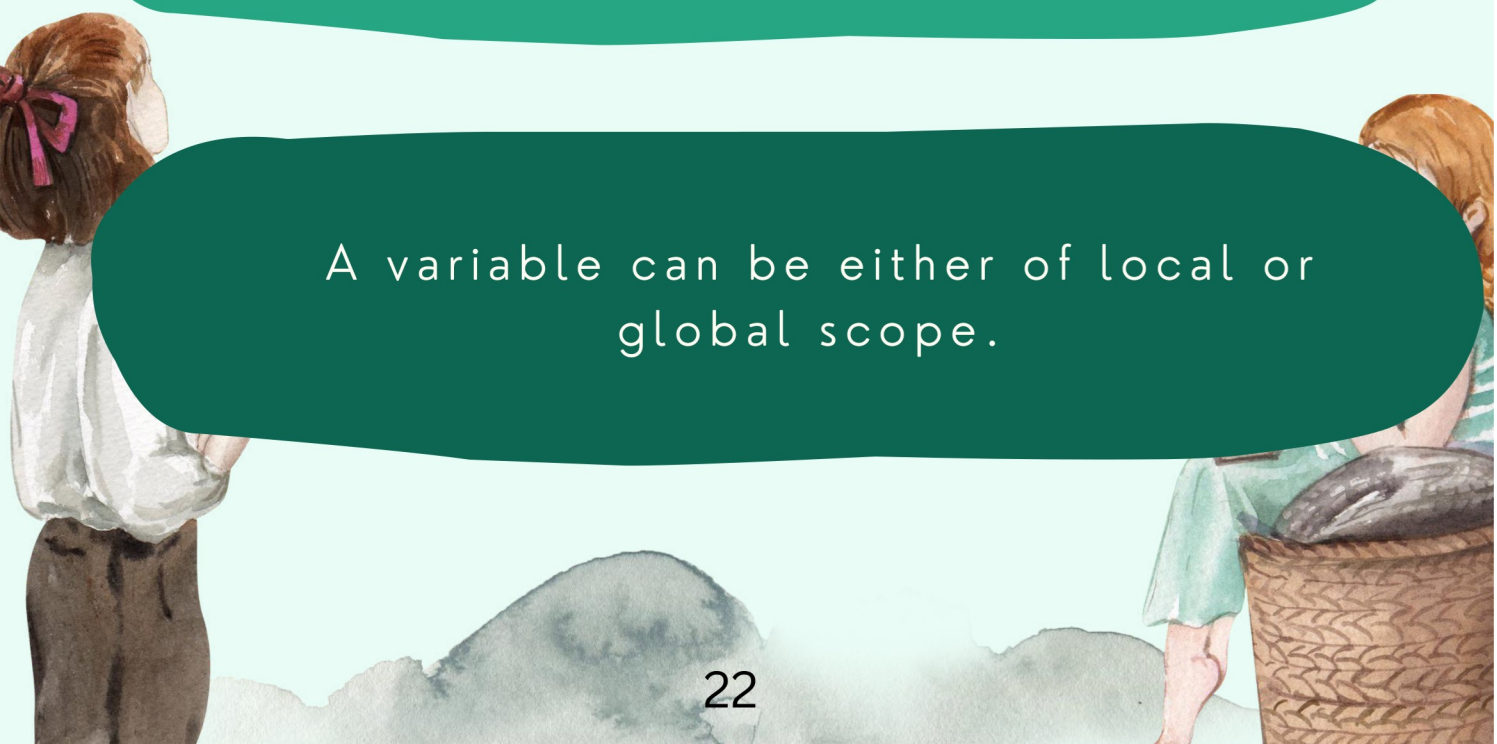


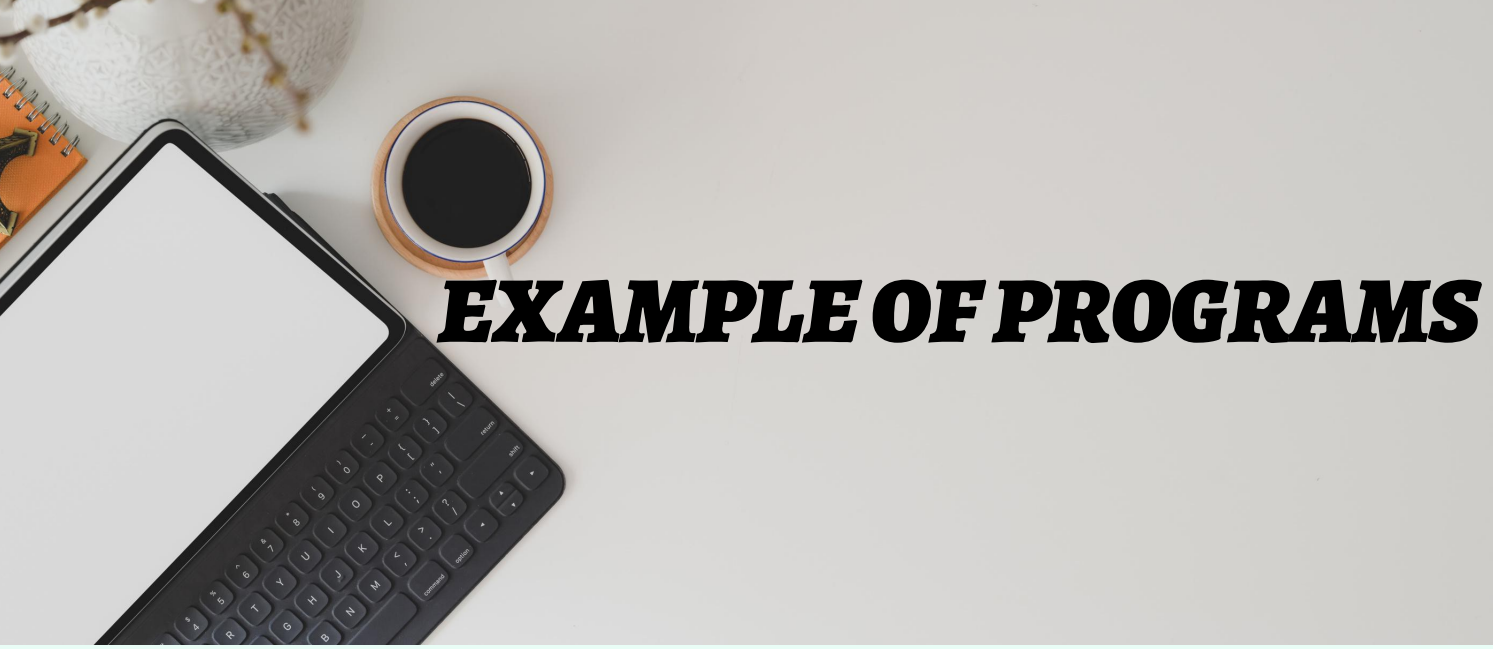
SCOPE OF VARIABLE

All the variables that we intend to use in a program must have been declared with its type specifier in an earlier.

we did in the previous code at the beginning of the body of the function main when we declared that a, b, and result were of type int.

A variable can be either of local or global scope.





EXAMPLE OF PROGRAMS

```
#include<iostream>
using namespace std;

int num1,num2;
char name;

int main()
{
float total;
cout<<"First number is:";
cin>>num1;
cout<<"Second number is:";
cin>>num2;
total=num1+num2;
cout<<"TOTAL OF TWO NUMBERS IS:"<<total;
return 0;
}
```

Global Variable

Local Variable

DIFFERENCES BETWEEN LOCAL AND GLOBAL VARIABLE

LOCAL VARIABLE	GLOBAL VARIABLE
variable declared within the body of a function or a block.	variable declared in the main body of the source code and outside all functions.
variables is limited to the block enclosed in braces ({}) where they are declared	variables can be referred from anywhere in the code, even inside functions, whenever it is after its declaration.

INPUT AND OUTPUT STATEMENT

- Input/output statement operation is fundamental to any programming language.

- In c++ **cin** and **cout** are used to input data and to output the data.

  cin>> is used to get data from keyboard.
  cout<< is used to send the output to the screen.

- Preprocessor directive **#include<iostream>** must be used in order to handle the input **cin>>** and output **cout<<** statement.

- The other type of input is gets **(variableName)** and **cin.getline(variablename,length).**

- The gets**(variableName)** should be handle by preprocessor directive **#include<stdio>**

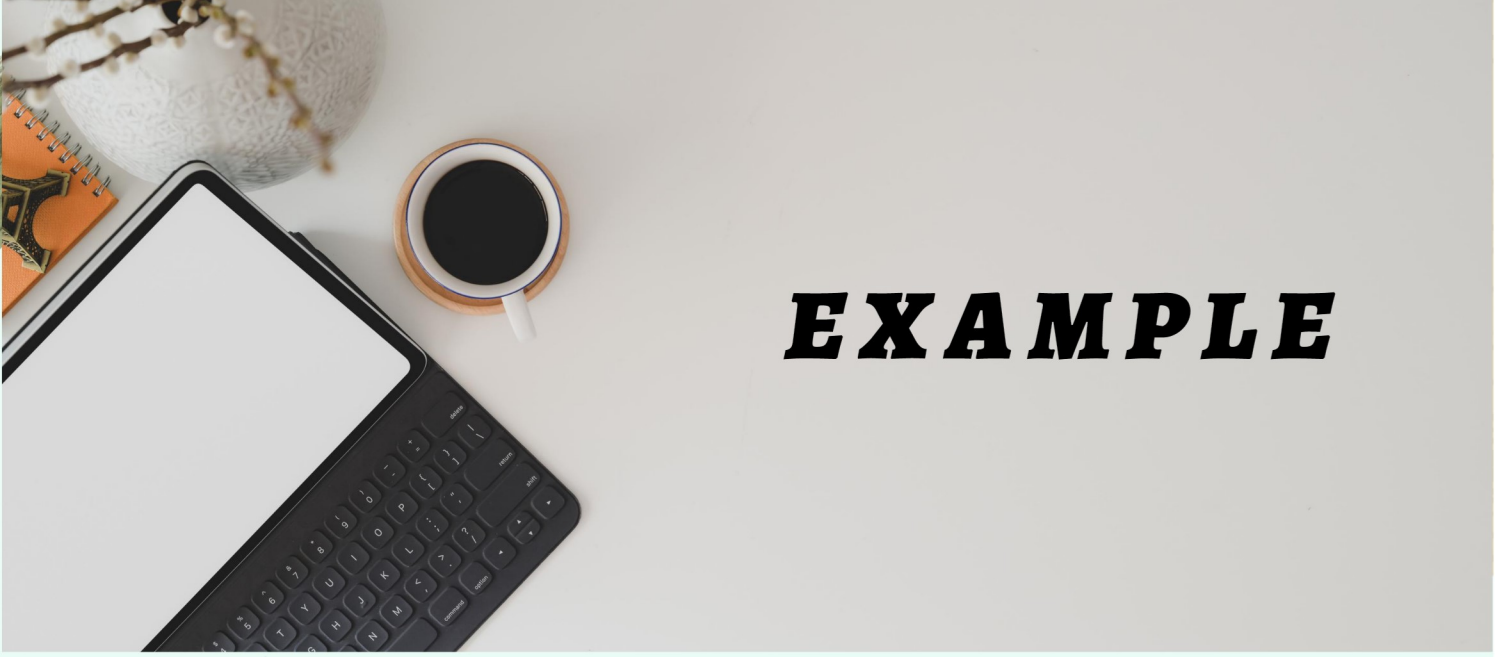
- While **cin.getline(variablename,length)** should be handle by preprocessor directive #include<string>

EXAMPLE OF INPUT STATEMENT

TASK	USAGE/SAMPLE INPUT DATA
To input numeric data type	**int num1;** **float num2;** **cin>>num1>>num2; //input 10 5.5**
To input 2 character data types	**char letter, symbol;** **cin>>letter>>symbol; //input A#**
To input sequence of characters (string)	**char custName[50];** **gets(custName);** **string custName;**
	char custName[50]; **cin.getline(custName,50);** **cin>>custName;**

EXAMPLE OF OUTPUT STATEMENT

TASK	USAGE/SAMPLE INPUT DATA
To display labels Enter 2 number:	**cout<<"Enter 2 numbers";**
To display result from an arithmetic expression	**cout<<5*2;** **cout<<sum/count;**
To display label Total is: and a value stored in variable total. Complete output is Total is:10	**int total=10;** **cout<<"Total is:"<<total;**
To display formatted output and predefined symbol, endl \n	**cout<<"\n";** **cout<<endl;**



EXAMPLE

```
// cin with strings
#include <iostream>
#include <string>
using namespace std;
int main ()
{
string mystr;
cout << "What's your name? ";
getline (cin, mystr);
cout << "Hello " << mystr << ".\n";
cout << "What is your favorite team? ";
getline (cin, mystr);
cout << "I like " << mystr << " too!\n";
return 0;
}
```

OPERATOR AND EXPRESSION

Operator

- Operator is a symbol that causes the compiler to take an action.

- Operators act an operands and in c++ all operands are expressions.

- In c++ there are several different categories of operators. These categories are:

1. Assignment operator
2. Arithmetic operator
3. Increment and Decrement operator
4. Relational operator
5. Logical operator

ASSIGNMENT OPERATOR

- C++ has several assignment operators. The most commonly used assignment operator is : =

- Assignment operator using = has the general form:
  identifier = expression

- Where identifier generally represent variable and expression represent constant, a variable or a more complex expression.

ASSIGNMENT STATEMENT

- An assignment statement is used to place a value into another variable coded directly within the program.

- In other words, the variable gets the value initially assigned to it in the program (hard-coded) and not from the value keyed in by the user.

- Example:

```
quantity=20;
price=5.50;
amount=basic_pay+overtime;
```

ARITHMETIC OPERATOR

Operator	Meaning	Example
+	addition	x + y
-	subtraction	x - y
*	multiplication	x * y
/	division	x / y
%	modulo	x % y

COMPOUND ASSIGNMENT

Expression	is equivalent to
value += increase;	value = value + increase;
a -= 5;	a = a - 5;
a /= b;	a = a / b;
price *= units + 1;	price = price * (units + 1);

```
a=10; b=2;
cout<<a/=b;//5
cout<<a=a/b;//error
a=a/b;
cout<<a;//5
```

INCREMENT AND DECREMENT OPERATOR

- increase operator (++) and the decrease operator (--)
- increase operator (++) increase by one the value stored in a variable.
- the decrease operator (--) reduce by one the value stored in a variable.
- They are equivalent to +=1 and to -=1, respectively.

```
c++;
c+=1;
c=c+1;
```

- The three of them increase by one the value of c.

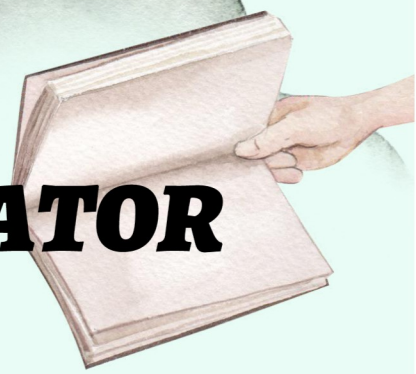
Example 1

```
B=3;
A=++B;
// A contains 4, B contains 4
```

Example 2

```
B=3;
A=B++;
// A contains 3, B contains 4
```

In Example 1, B is increased before its value is copied to A. While in Example 2, the value of B is copied to A and then B is increased.



RELATIONAL OPERATOR

- Relational operators (>, <, >=, <=)(==, !=,)
- Here is a list of the relational operators that can be used in C++:

- > Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to
- == Equal to
- != Not equal to

- Here there are some examples:

(7 == 5) **// evaluates to false.**
(5 > 4) **// evaluates to true.**
(3 != 2) **// evaluates to true.**
(6 >= 6) **// evaluates to true.**
(5 < 5) **/ / evaluates to false.**

- Of course, instead of using only numeric constants, we can use any valid expression, including variables. Suppose that: a=2, b=3 and c=6,

(a == 5) **// evaluates to false since a is not equal to 5.**
(a*b >= c) **// evaluates to true since (2*3 >= 6) is true.**
(b+4 > a*c) **// evaluates to false since (3+4 > 2*6) is false.**
((b=2) == a) **// evaluates to true.**

LOGICAL OPERATOR

- **Logical operators (!, &&, ||)**

- The Operator ! is the C++ operator to perform the Boolean operation NOT.

- Basically, it returns the opposite Boolean value of evaluating its operand. For example:

!(5 == 5) // evaluates to false because the expression at its right (5 == 5) is true.

!(6 <= 4) // evaluates to true because (6 <= 4) would be false.

!true // evaluates to false

!false // evaluates to true.

LOGICAL OPERATOR

<table>
<tr><th>OPERATOR</th><th>EXPLAIN</th><th>EXAMPLE</th></tr>
<tr><td>&&</td><td>The operator && corresponds with Boolean logical operation AND. This operation results true if both its two operands are true, and false otherwise.</td><td>
<table>
<tr><th>a</th><th>b</th><th>a && b</th></tr>
<tr><td>true</td><td>true</td><td>true</td></tr>
<tr><td>true</td><td>false</td><td>false</td></tr>
<tr><td>false</td><td>true</td><td>false</td></tr>
<tr><td>false</td><td>false</td><td>false</td></tr>
</table>
</td></tr>
<tr><td>||</td><td>The operator || corresponds with Boolean logical operation OR. This operation results true if either one of its two operands is true, thus being false only when both operands are false themselves.</td><td>
<table>
<tr><th>a</th><th>b</th><th>a||b</th></tr>
<tr><td>true</td><td>true</td><td>true</td></tr>
<tr><td>true</td><td>false</td><td>true</td></tr>
<tr><td>false</td><td>true</td><td>true</td></tr>
<tr><td>false</td><td>false</td><td>false</td></tr>
</table>
</td></tr>
</table>

DIFFERENTIATE ASSIGNMENT (=) AND EQUALITY(==) OPERATOR

ASSIGNMENT OPERATOR	EQUALITY OPERATOR
assigns the value at its right to the variable at its left. a=b+5	compares whether true or false value. a==b+5

EXAMPLE 1

Suppose that:
a=2, b=3

a=b+5	a==b+5
a=3+5	2==3+5
a=8	2==8
	false

EXAMPLE 2

Suppose that:
a=2, b=3

((b=2) == a)

•we first assigned the value 2 to b and then we compared it to a, that also stores the value 2, so the result of the operation is true.

EXAMPLE 3

Suppose that:
a=2, b=3

((b=2) == a)
((2)==2)
True.


PRECEDENCE OF OPERATORS

- When writing complex expressions with several operands, we may have some doubts about which operand is evaluated first and which later.

- For example, in this expression:
  a = 5 + 7 % 2
- a = 5 + (7 % 2) // **with a result of 6, or**
- a = (5 + 7) % 2 // **with a result of 0**

- The correct answer is the first of the two expressions, with a result of 6.

- There is an established order with the priority of each operator, and not only the arithmetic ones (those whose preference come from mathematics) but for all the operators which can appear in C++.

- From greatest to lowest priority, the priority order is as follows:


PRECEDENCE OF OPERATORS

- When writing complex expressions with several operands, we may have some doubts about which operand is evaluated first and which later.

- For example, in this expression:
  a = 5 + 7 % 2
- a = 5 + (7 % 2) // **with a result of 6, or**
- a = (5 + 7) % 2 // **with a result of 0**

- The correct answer is the first of the two expressions, with a result of 6.

- There is an established order with the priority of each operator, and not only the arithmetic ones (those whose preference come from mathematics) but for all the operators which can appear in C++.

- From greatest to lowest priority, the priority order is as follows:

OPERATORS PRECEDENCE

Operations :

( )

! + + _ _

* / %

+ -

< <= > >=

= = !=

& &

| |

=

Highest priority

Lowest priority

EXPRESSION

- Anything that evaluates to a value is an expression in c++.
- This expression is said to return a value.
- Thus, 3+2 return a value of 5 and so is an expression.
- All expression are statement

Example:

```
x=a+b+c;
Num1+num2;
```

Exercise Chapter 2

1. Prepare the problem analysis, algorithm, flowchart, pseudo code and program that asks the user to enter two numbers, obtains the two numbers from the user, and prints the sum, product, difference and modulus of the two numbers.

2. Prepare the problem analysis, algorithm, flowchart, pseudo code and program that reads in the radius of a circle and prints the circle's diameter, perimeter, and area. Use the constant value 3.14159 for "pi".

3. What is the value of the expressions below? Show your workings.
(a). x = (2 * 2 + (3 * 3 – (6 / 2)));
(b). i = 7 / 2 + 9 * 1.5 – 10.0;
(c) x = 5.0 / 2 * 2 + 5 / 2 * 2;

4. i, j and k are integer variables with the values i = 2, j=5 and k=15. State whether expressions below are TRUE or FALSE.
(a). i > j - k
(b). (j < l) || (k > j)
(c) (i > 0) && (j < k) || (k < i) (d) (i >= 1) && (j == 5)
(e) ! (i > j)
(f) i – j > k

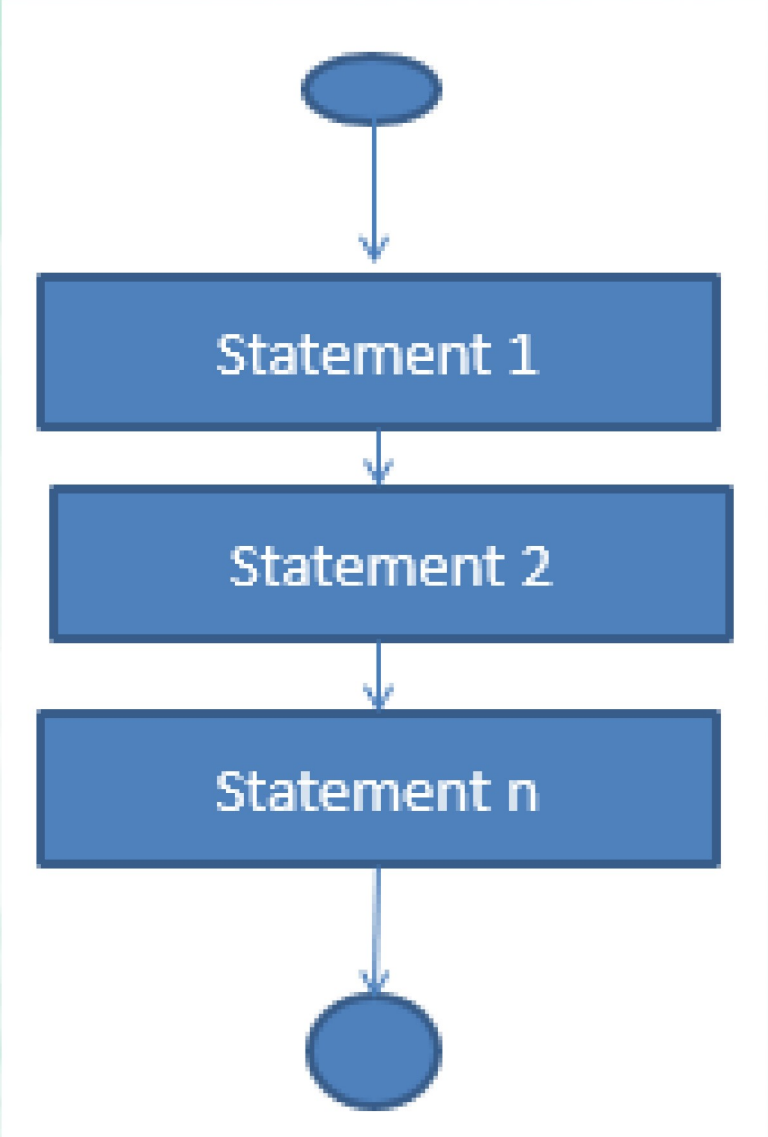
Chapter 3:

PROGRAM CONTROL

SEQUENTIAL STRUCTURE

- **One entry point and one exit point.**
- **No choice are made and no repetition.**
- **One after another without leaving out any single statement.**

FLOWCHART



EXAMPLE OF SEQUENTIAL STATEMENT

```
#include<iostream>
using namespace std;
int main()
{
    float payrate=8.5;
    float hourWorked=25.0;
    float wages;

    wages= hourWorked * payrate;
 cout<<"\n Wages = RM:"<<wages<<endl;

return 0;

}
```

Output:

Wages = RM:212.5

EXAMPLE OF SEQUENTIAL STATEMENT

```
include<iostream>
using namespace std;
int main()
{
        int num1,num2;
        float total;

        cout<<“NUMBER 1:”;
        cin>>num1;
        cout<<“NUMBER 2:”;
        cin>>num2;

        total=num1+num2;
        cout<<“TOTAL OF TWO NUMBERS IS:”
        <<total<<endl;


return 0;

}
```

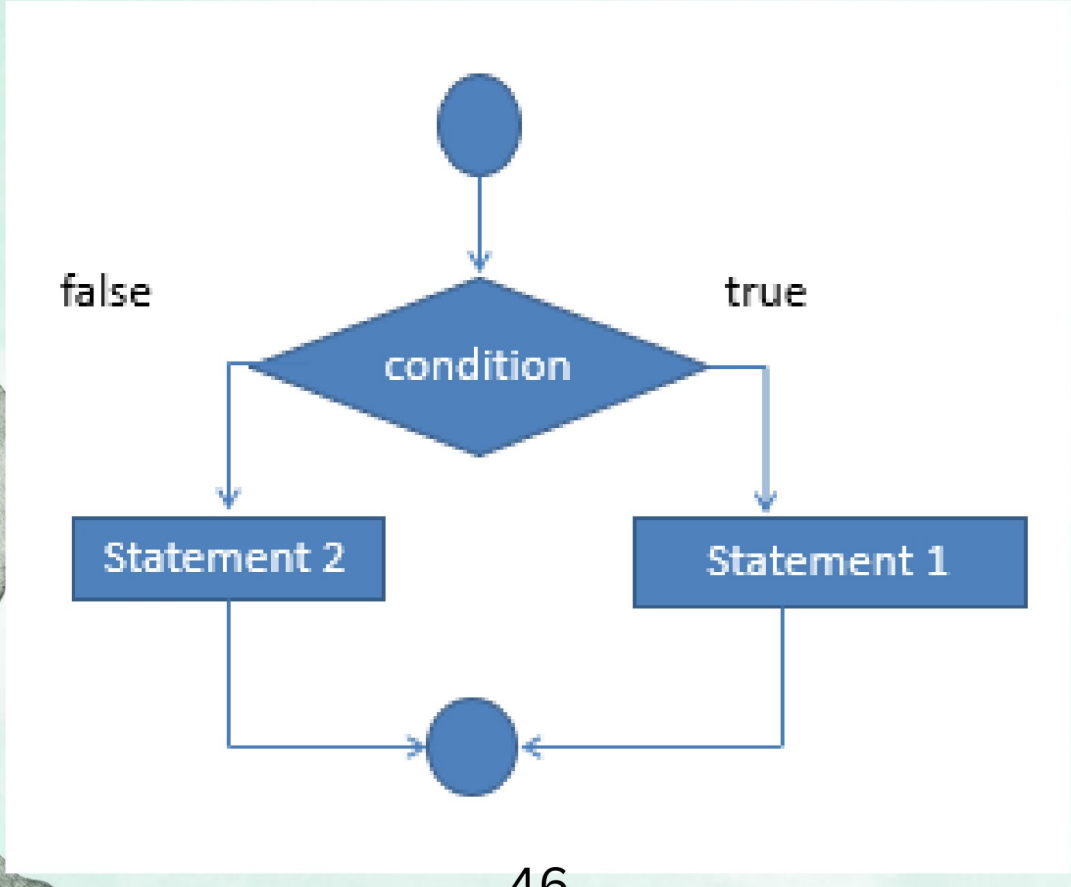
Output:

```
NUMBER 1:10
NUMBER 2:5
TOTAL OF TWO
NUMBERS IS:15
```

SELECTION STRUCTURE

- **used to select between two or more options.**
- **else statement will never exist in a program without if statement.**
- **If tested condition is TRUE, the entire block of statements following the if is performed.**
- **If the tested condition is FALSE, the entire block of statement following the else is performed**

FLOWCHART



SYNTAX

```
if(condition)
{
    true statement
}
else
{
    false statement
}
```

Example of if..else statement

```
#include<iostream>
using namespace std;
int main()
{
      int age;
      cout<<"\n Key in your age:";
      cin>>age;

      if(age>21)
     {
     cout<<"Qualifiied to have driving
license";
      }
     else
     {
     cout<<"Not qualified to have driving
license";
      }
      return 0;
}
```

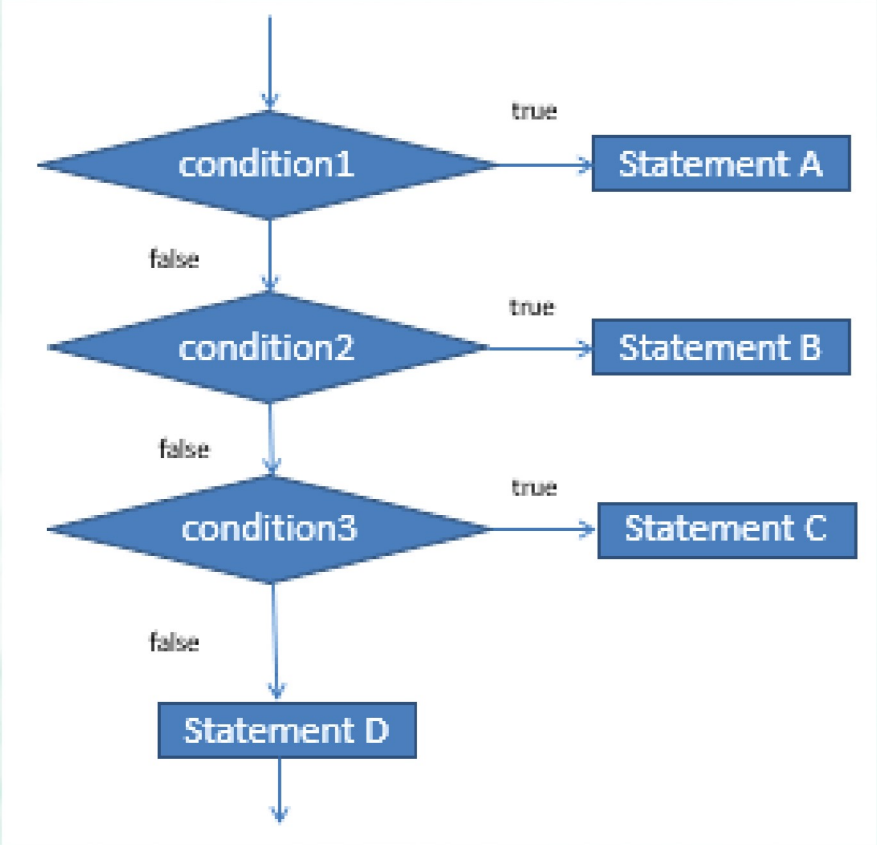
Output:
Key in your age:25
Qualifiied to have driving license

Output:
Key in your age:18
Not qualified to have driving license

Output:
Key in your age:21
Not qualified to have driving license

Nested if..else

```
if(condition 1)
{
Statement A
 }
else if(condition 2) {
 Statemnet B
 }
else if(condition 3)

{
 Statement C
}
else
 {
 Statement D
 }
```

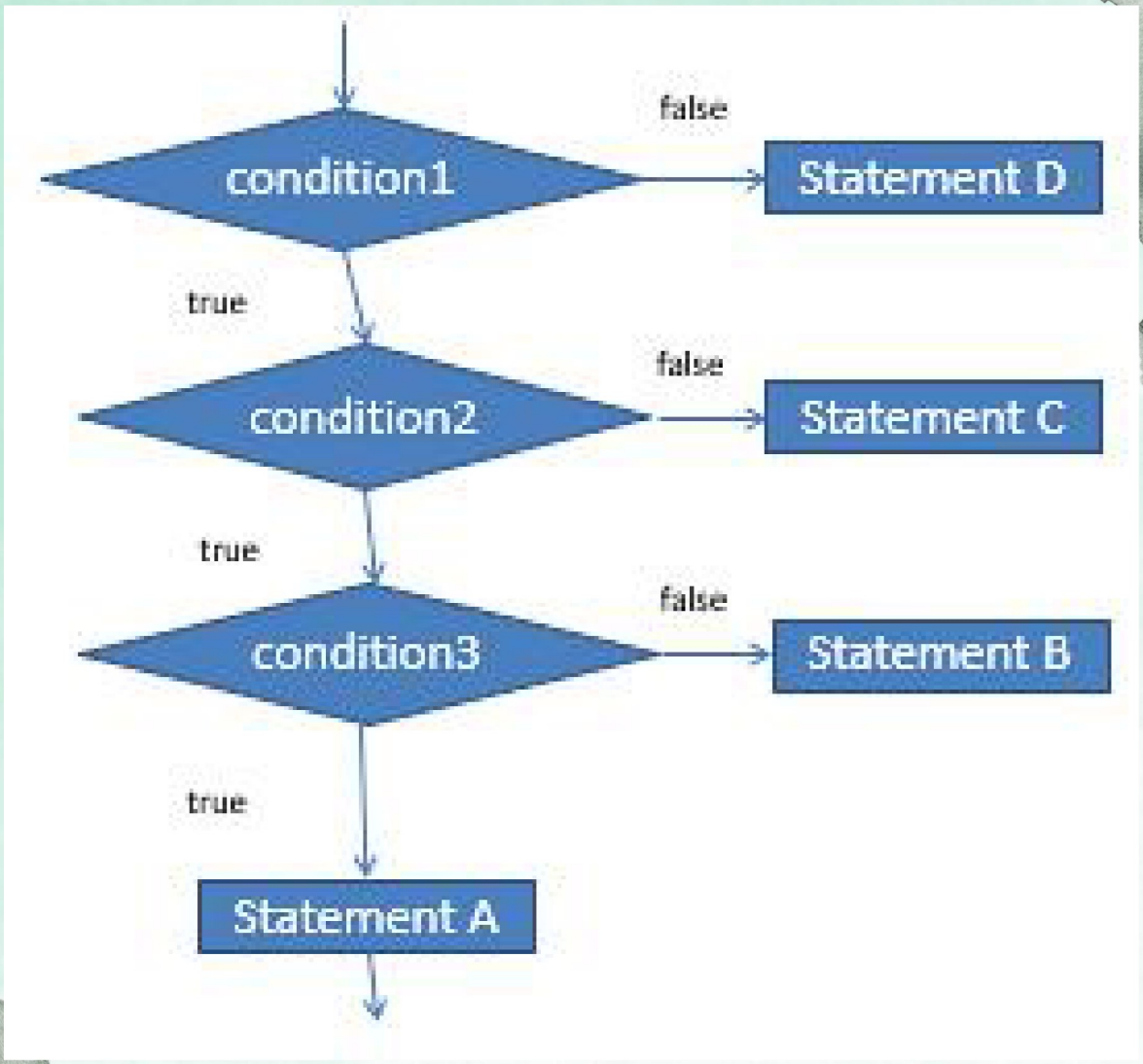


EXAMPLE OF NESTED IF..ELSE STATEMENT

```
#include<iostream>
using namespace std;
int main()
{
        float cgpa;
        cout<<"\n CGPA:";
        cin>>cgpa;
        if(cgpa < 0.00 || cgpa > 4.00)
        {
        cout<<"Invalid data";
        {
        else if(cgpa > = 3.5 && cgpa < = 4.00)
                                {
        cout<<"Dean List";
         }
        else if(cgpa > 2.00 && cgpa < 3.50)
         {
       cout<<"Pass";
        }
       else if(cgpa > = 1.80 && cgpa < 2.00)
       {
      cout<<" Conditional Pass;;
       }
      else
      {
      cout<<" Fail";
       }
      return 0;
}
```

Output:
CGPA:2.5
Pass

NESTED IF..ELSE



NESTED IF..ELSE

```
If(condition 1)
{
        If(condition 2)
        {
        If(condition 3)
       {
       Statement A
       }
       else
       {
       Statement B
       }
       }
       else
       {
       Statement c
       }
       }
       else
       {
       Statement D
}
```

EXAMPLE OF STATEMENT

```
#include<iostream>
using namespace std;
int main()
{
        float cgpa,salary;
        int year;

         cout<<"\n YEAR:";
         cin>>year;
         cout<<"\n CGPA:";
         cin>>cgpa;
         cout<<"\n SALARY:";
         cin>>salary;
         if(year>1)
         {
         if(cgpa>=3.00)
        {
        if(salary<=500)
        {
        cout<<"Your application is under consideration";
         }
        else
        {
        cout<<"Not success because salary is more than RM500.00";
         }
         }
         else
         {
         cout<<" Not success because cgpa is less than 3.00";
        }
        }
        else
        {
        cout<<" Not success because year of study is less than 1";
        }
return 0;
}
```

```
OUTPUT:
YEAR:2
CGPA:3.5
SALARY:250
YOUR APPLICATION IS UNDER CONSIDERATION
```

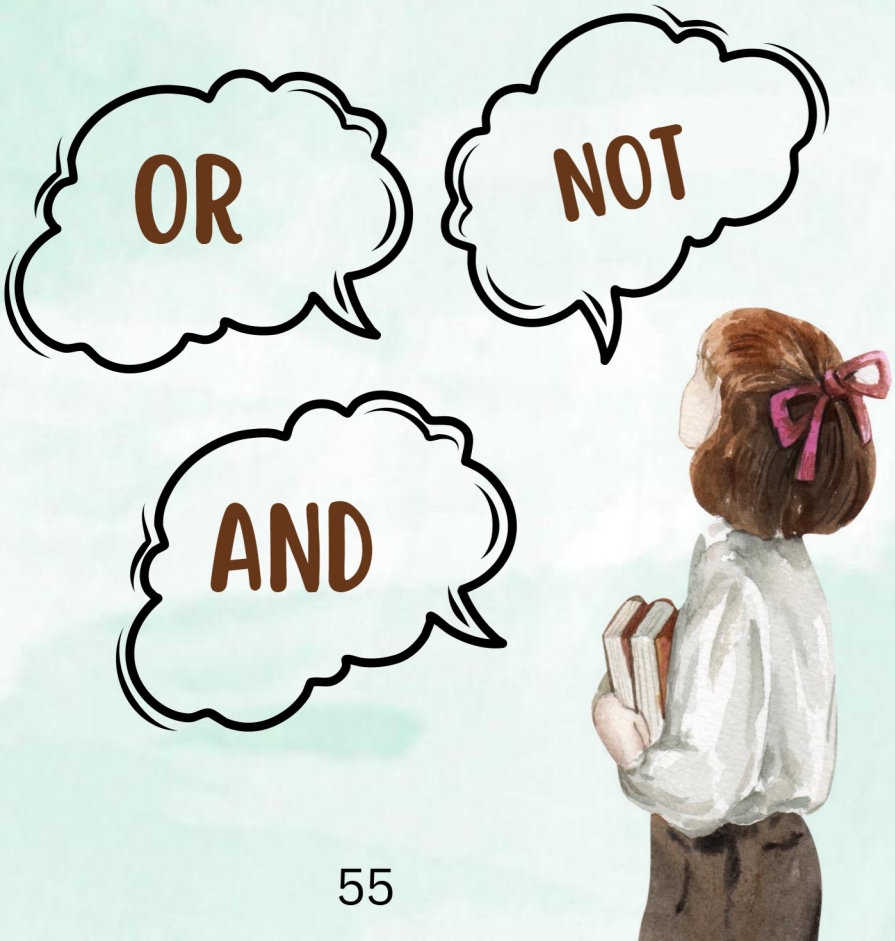
EXAMPLE OF STATEMENT

```
#include<iostream>
using namespace std;
int main()
{
        float cgpa,salary;
        int year;

         cout<<"\n YEAR:";
         cin>>year;
         cout<<"\n CGPA:";
         cin>>cgpa;
         cout<<"\n SALARY:";
         cin>>salary;
         if(year>1)
         {
         if(cgpa>=3.00)
        {
        if(salary<=500)
        {
        cout<<"Your application is under consideration";
         }
        else
        {
        cout<<"Not success because salary is more than RM500.00";
         }
         }
         else
         {
         cout<<" Not success because cgpa is less than 3.00";
        }
        }
        else
        {
        cout<<" Not success because year of study is less than 1";
        }
return 0;
}
```

```
OUTPUT:
YEAR:2
CGPA:3.5
SALARY:250
YOUR APPLICATION IS UNDER CONSIDERATION
```

USAGE OF LOGICAL OPERATOR

Symbol (pseudocode)	Symbol (c++ language)	Example	Result	Description
AND	&&	(1>3)&&(10<20)	FALSE	Both sides of the condition must be true.
OR	||	(1>3)||(10<20)	TRUE	Either one of the condition must be true.
NOT	!	!(10<20)	FALSE	Change the operation either from true to false or vise versa.



If condition (num1< num2) is true AND condition (num1 > num3) is true then print “First number is the largest number” will be displayed.

If one of the condition is not true then print “First number is the largest number” will not be displayed.

AND

```
IF ((NUM1 › NUM2 ) && ( NUM1 › NUM3 ))
COUT‹‹“FIRST NUMBER IS THE LARGEST NUMBER”;
```

if the sales are more than 5000 or working hours are more than 81, bonus RM500 will be given. If either condition is not fulfilled, still the amount of RM500 will be given as bonus.

If both condition are false, then bonus will not be given.

OR

```
IF ( ( SALES › 5000 ) || (HOURSEWORKED › 81 ))
BONUS = 500;
ELSE
```

SWITCH

- The switch..case structure consists of a series of case label and optional default case which can be included for option that is not listed in the case label.

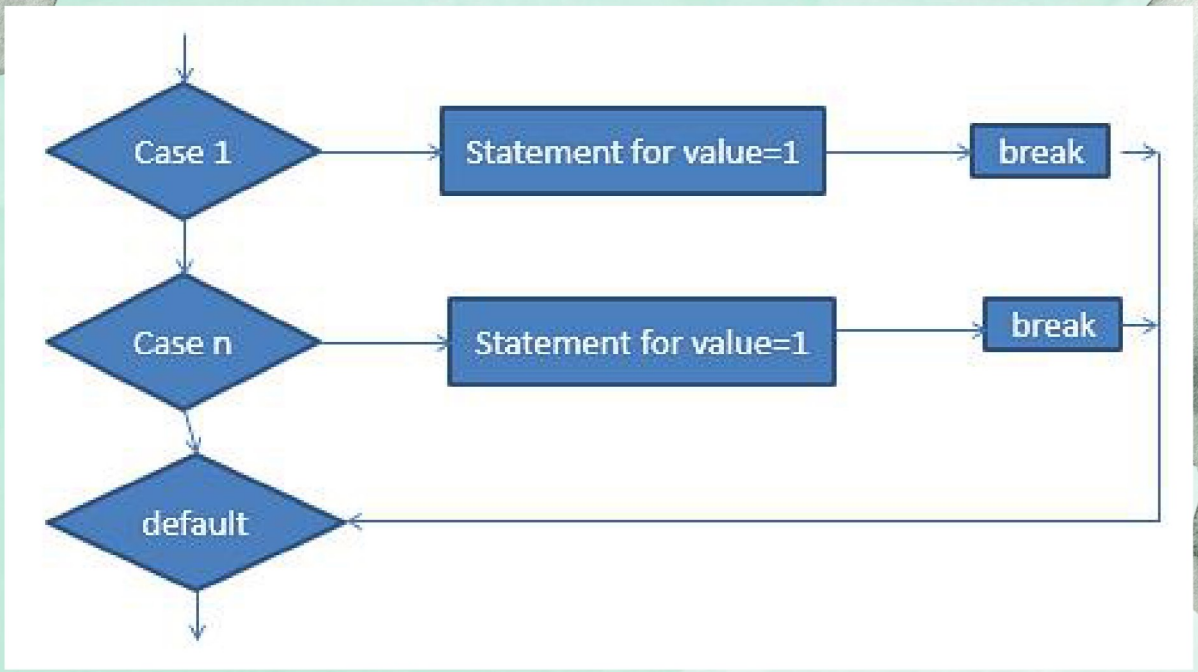
- Switch which is followed by variable name in parentheses called controlling expression.

- The break keyword must be included at the end of each case statement.

- The break statement causes program control to proceed with the first statement after the switch structure.

- Without break statement, all command for that case and next case label will be executed.

- Default is an option that will only be executed if suitable case cannot be found.



SYNTAX

```
switch(expression)
{
case value 1:
        statement if expression =value 1
        break;
case value n:
        statement if expression =value n
        break;
default:
       statement if none of the expression is matched
}
```

EXAMPLE WITH BREAK STATEMENT

```
#include<iostream>
using namespace std;
int main()
{
            int num;
            cout<<"\n Enter a number:";
            cin>>num;

           switch(num)
          {

          case 1:
                    cout<<"Welcome";
          break;
          case 2:
                     cout<<"Bye";
          break;
          default:
          cout<<"Wrong Number";
          }
return 0;
```

Output:
Enter a number:10
Wrong number

EXAMPLE WITHOUT BREAK STATEMENT

```
#include<iostream>
using namespace std;
int main()
{
        int num;
        cout<<"\n Enter a number:";
        cin>>num;

        switch(num)
        {

         case 1:
                cout<<"Welcome";
         case 2:
                 cout<<"Bye";
        default:
        cout<<"Wrong Number";
        }
return 0;
```

Output:
Enter a number:1
WelcomeByeWrong Number

EXAMPLE OF PROGRAM

```
#include<iostream>
using namespace std;
int main()
{
        char grade;
        cout<<"\n Enter a grade (A-D):";
        cin>>grade;  //b

        switch(grade)
        {
        case 'A':
                cout<<"Minimun marks is 80";
        break;
        case 'B':
                cout<<"Minimun marks is 60";
        break;
        case 'C':
                cout<<"Minimun marks is 40";
       break;
       case 'D':
                cout<<"Minimun marks is 25";
       break;
       default:
               cout<<"Mark between 0-24";
       }
return 0;
}
```

Output:
Enter a grade (A-D):C
Minimun mark is 40

EXPLANATION

- A,B,C, and D are the possible values that can be assigned to the variable grade.
- In each case of constant, A to D a statement or sequence of statement would be executed.
- You would have noticed that every line has statement under it called break.
- The break is the only thing that stop a case statement from continuing all the way down through each case label under it.
- In fact if you do not put break in, the program will keep going down in the case statement. Into other case labels, until it reaches the end of break.
- However the default part is the codes that would be executed if there were no matching case of constant's value.

RULES FOR SWITCH STATEMENT

1. The value of case statement must be an integer or a character constant.
2. Case statement arrangement is not important.
3. Default can exist earlier (but compiler will place it at the end)
4. Can't use condition or limit.
5. Why switch is the best way to write a program:
    - easy to read
    - It is a more spesific statement to implement multiple alternative decisions.
    - It is used to make a decision between many alternative.

COMPARISON OF NESTED IF STATEMENT AND SWITCH..CASE STATEMENT

Nested if statement	switch.. case statement

#include<iostream>
using namespace std;
int main()
{
int code;
cout<<"Enter code (1,2 or 3)";
cin>>code;
 	
if(code==1)
cout<<"Your favourite flavor is vanila";
 	
else if(code==2)
cout<<"Your favourite flavor is chocolate";
 	
else if(code==3)
cout<<"Your favourite flavor is strawbery"
 	
else
cout<<"please choose code 1,2 or 3 only";
 	
return 0;
}
	#include<iostream>
	using namespace std;
	int main()
	{
	int code;
	cout<<"Enter code (1,2 or 3)";
	cin>>code;
	switch(code)
	{
	case1:
	cout<<"Your favourite flavor is vanila";
	break;
	case2:
	cout<<"Your favourite flavor is chocolate";
	break;
	case 3:
	cout<<"Your favourite flavor is strawbery";
	default:
	cout<<"please choose code 1,2 or 3 only";
	}
	return 0;
	}

Exercise Chapter 3

1. If the payment RM5,000 to RM10,000, display a message that the customer will be given a free ticket to Langkawi. Otherwise, display a message that the customer will only be given a free lunch at Quality Hotel. Translate the statement above to program segment.

2. You are given the following requirement:
   a. Your program is able to read the amount of sales for a sale executive.

   b. If the sale is more than RM10,000 then the commission will be 5% of the sale. Otherwise, the commission is only 3% of the sales.

   c. Your program is able to calculate the amount of commission by multiply the percentage of commission with sale.

   d. Your program is able to display the amount of commission to the sale executive.

3. Write a c++ program to find a maximum value of three integers which are a, b and c.

4. You're given the following requirements:
   a. Your program is able to read the item code from the keyboard.

   b. The item code determines the price for each item in the shop. The following table shows the prices and charges of 7 items in the shop.

Item Code	Price (RM)	Charge(%)
A	54.00	5
B	65.00	5
C	82.00	5
D	103.00	7
E	150.00	7
F	245.00	10
H	250.00	10

c. If the item code is not matched, an error message is dispayed as follows:

"Error, this item code is not in the list."

d. For each selected item, the charge is calculated by multiplying the item price and charge rate.

e. Then the payment is a summation of item price and charge. Your program is to display this payment.

5. NASA Consultant Co Ltd wants to develop a program that can help the consultant to give advice regarding the home loan. The program receives two inputs from the consultant, which are the amount of House Cost in Ringgit Malaysia and years of term applied. Based on these two inputs, the program should be able to calculate the monthly repayment. The table for the interest calculation is provided as follows:

Year of Term Applied	Interest (%)
More than or equal to 25 years	7.5
More than or equal to 20 years	6.5
More than or equal to 15 years	5.5
More than or equal to 10 years	4.5
Less than 10 years	4.0

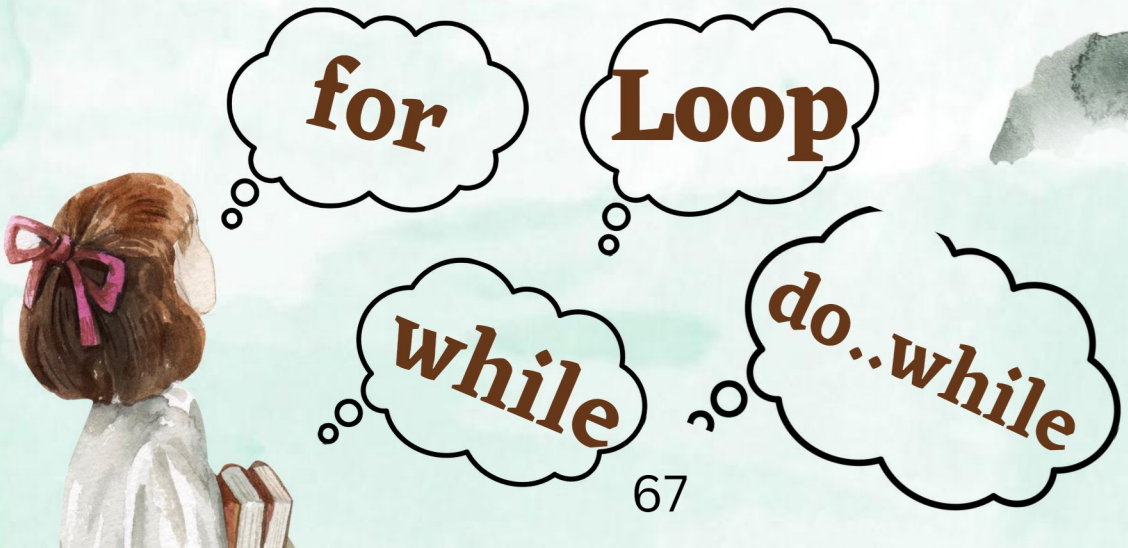
The formula for the monthly repayment calculation are as follows:
Interest (RM) = Interest (%) x House cost (RM)
House cost with Interest = House Cost (RM) + Interest (RM)
Monthly Repayment = House Cost with Interest / (Year of Term Applied x 12)

REPETITION STRUCTURE

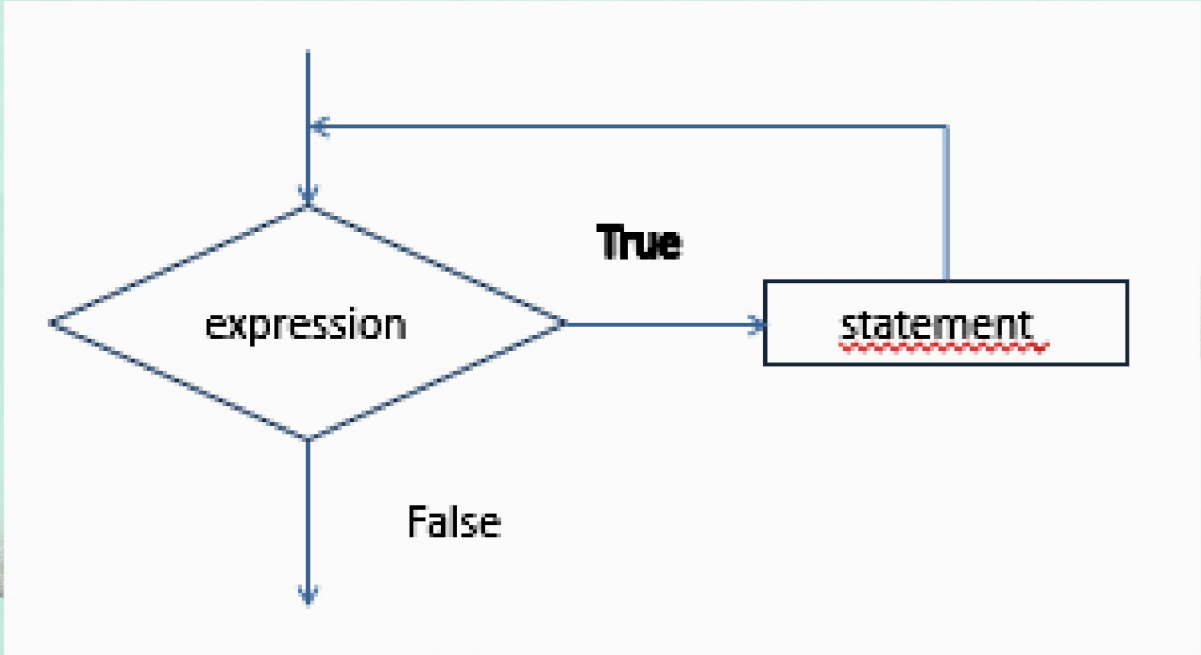
- **Why repetition is needed??**
- **Suppose we want to add five numbers to find their average.**
- **From what we have learned so far, we could proceed as follows.**

```
cin>>num1>>num2>>num3>>num4>>num5;
sum= num1+num2+num3+num4+num5;
average=sum/5;
```

- **How about we want to add 100 or 1000 numbers ??**
- **Therefore, repetition structure offers a better way of performing those task.**

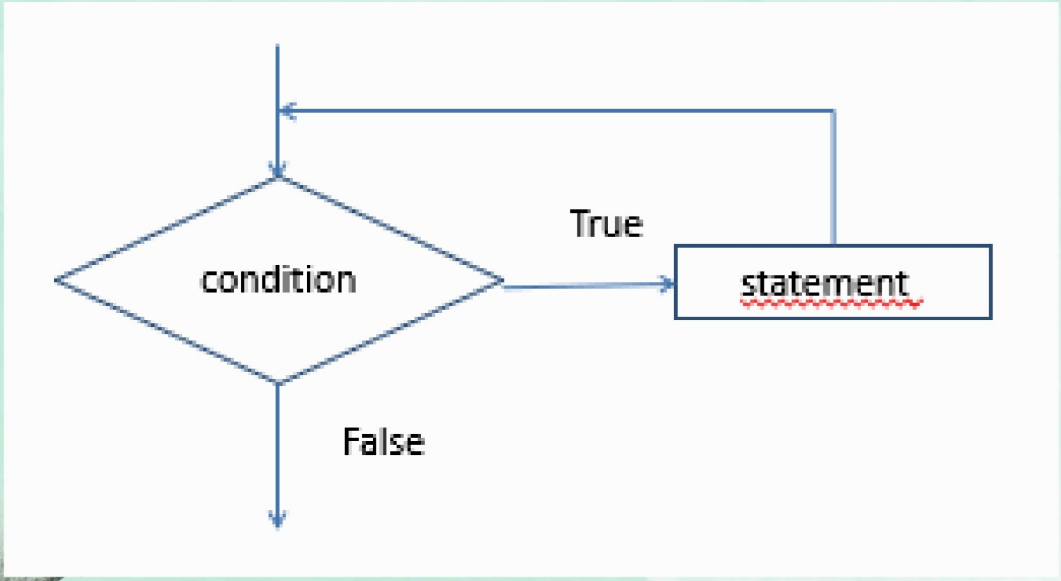


FOR STATEMENT




```
#include<iostream>
using namespace std;

int main()
{
	int num, i,sum;
	sum=0;
	for(i=0;i<5;i++)
	{
	cout<<"Enter number:";
	cin>>num;
	sum=sum+num;
	}
	cout<<"Total of numbers are:"<<sum;
return 0;
}
```

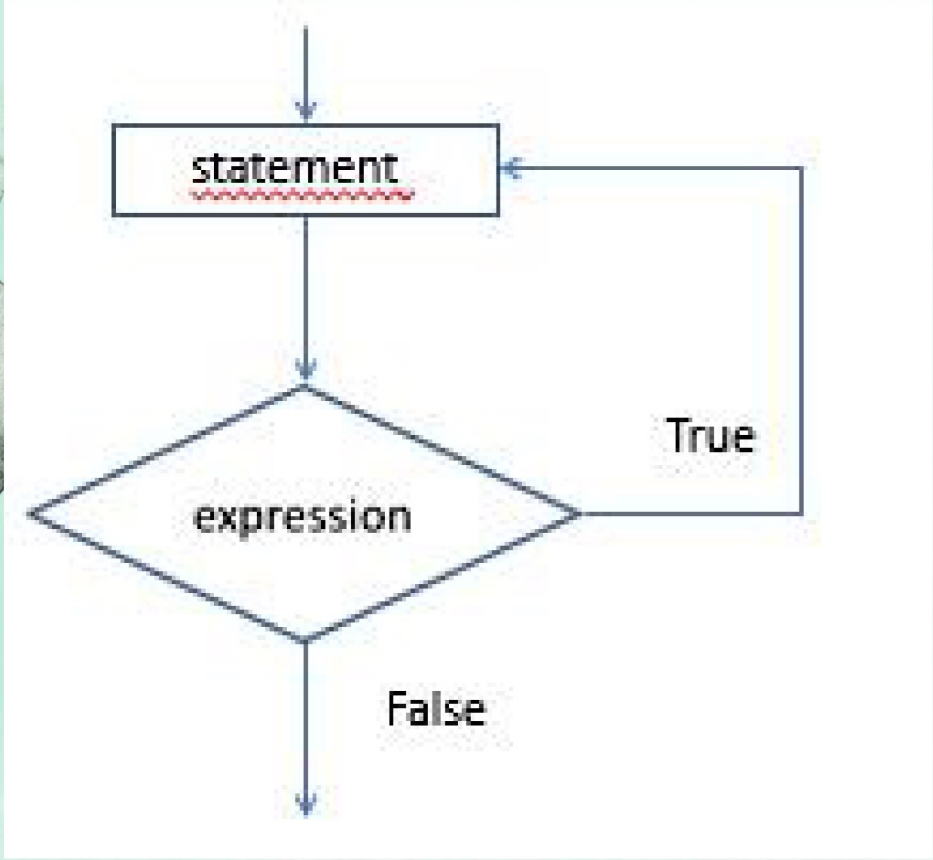
WHILE STATEMENT




```
#include<iostream>
using namespace std;

int main()
{
      int num, i,sum;
      i=0;
      sum=0;
      while(i<5)
      {
     cout<<“Enter number:”;
     cin>>num;
     sum=sum+num;
     i++;
     }
      cout<<“Total of  numbers are:”<<sum;
return 0;
}
```

Output:
Enter number:10
Enter number:2
Enter number:12
Enter number:4
Enter number:2
Total of numbers are:30

DO... WHILE STATEMENT



General form:

```
    do
       {
       statement block;
       }
while(expression);
```

EXAMPLE

```
#include<iostream>
using namespace std;

int main()
{
    int num, i,sum;
    i=0;
    sum=0;
    do
    {
    cout<<“Enter number:”;
    cin>>num;
    sum=sum+num;
    i++;
    }
    while(i<5);
    cout<<“Total of numbers are:”<<sum;
return 0;
}
```

Output:
Enter number:5
Enter number:8
Enter number:12
Enter number:3
Enter number:10
Total of numbers are:38

CONTINUE AND BREAK STATEMENT

- **Two commands that can be included in looping processes are continue and break.**
- **When included continue statement in the body of the loop causes the program to return immediately to the start of the loop and ignore any remaining part of the body.**
- **The break statement causes the program to immediately exit from the loop and resume at the next of the program.**

Difference between the loops structure

for	while	do..while

#include<iostream>
using namespace std;
 	
int main()
{
for(i=0;i<3;i++)
{
cout<<"welcome!";
}
return 0;
}
	#include<iostream>
	using namespace std;
	
	int main()
	{
	i=0;
	while(i<3)
	{
	cout<<"welcome!";
	i++;
	}
	return 0;
	}
		#include<iostream>
		using namespace std;
		
		int main()
		{
		i=0;
		do
		{
		cout<<"welcome!";
		i++;
		} while(i<3);
		return 0;
		}

EXAMPLE OF PROGRAM (CONTINUE)

```
#include<iostream>
using namespace std;

int main()
{
    int num, i,sum;
    i=0;
    sum=0;
    while(i<5)
    {
    i++;
    if(i==4)
    continue;
    cout<<"Enter number "<<i<<":";
    cin>>num;
    sum=sum+num;
    }
cout<<“Total of numbers are:”<<sum;
return 0;
}
```

Output:

```
Enter number 1:10
Enter number 2: 5
Enter number 3:12
Enter number 5:3
Total of numbers are:30
```

EXAMPLE OF PROGRAM (BREAK)

```
#include<iostream>
using namespace std;

int main()
{
      int num, i,sum;
      i=0;
      sum=0;
      while(i<5)
      {
      cout<<“Enter number”;
      cin>>num;
      sum=sum+num;
      i++;
      if(i==3)
      break;
      }
      cout<<“Total of numbers are:”<<sum;
return 0;
}
```

Output:

```
Enter number:5
Enter number:20
Enter number:10
Total of numbers are:35
```

Chapter 4:

ARRAY, STRUCTURE AND POINTER

ARRAY

- Array is a collection of data storage locations, which holds the same type of data. Each storage location is called an element of the array.

- General form to declare an array

```
Data_type  Array_name  [ Array_size];
```

- Data_type is the specifier that indicates what data type.

- Array_name is the name of the declared array.

- Array_size defines how many elements the array can contain

EXAMPLE

float price[10];

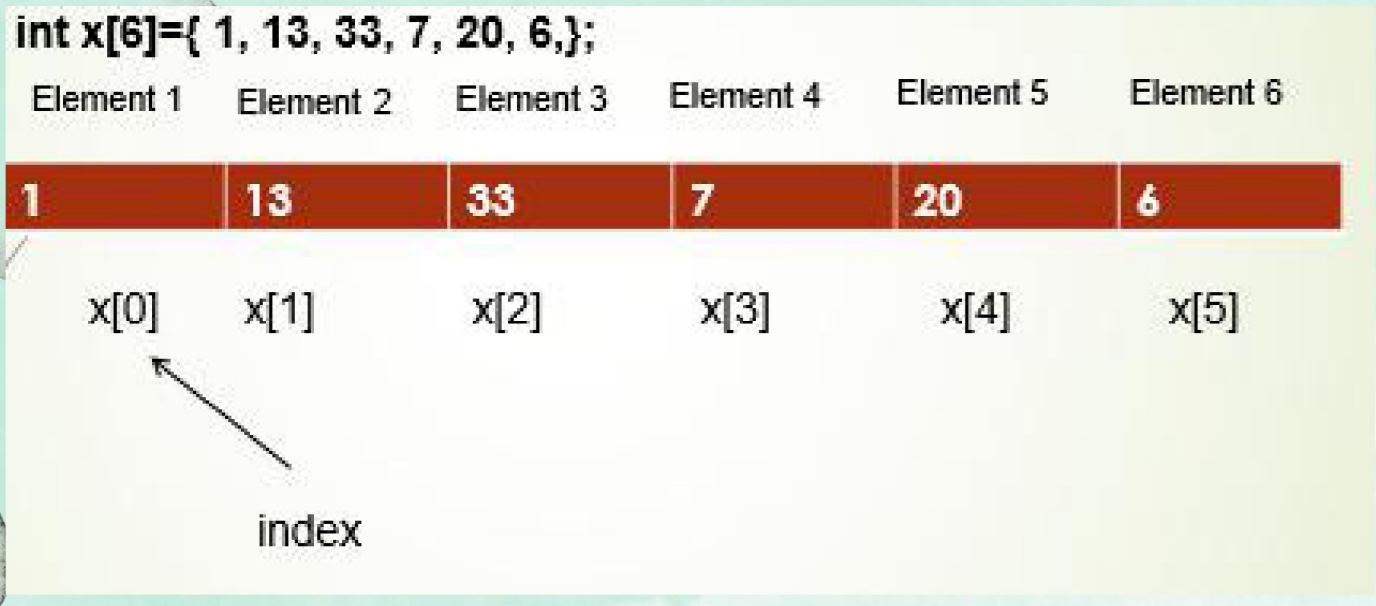
float specifies as the data type
price is the name of the declared array
the size of the array is 10

char name [50];
fatimah
String name[50];
Fatimha
Ahmad
Kamariah
hazim
int age [20];

```
int x[6]={ 1, 13, 33, 7, 20, 6};
```

- **The integer inside the braces { and } are assigned to the corresponding elements of the array.**

ILLUSTRATE ONE DIMENSIONAL ARRAY



EXAMPLE

int x [9] = {0,3,7,3,0,20,61,17,800 };

double x [8] = {1.2, 1.4, 1.7, 2.6, 3.4, 2.6, 1.2, 3.1};

char x[9] = {'a','b','c','d','e','f','g','h','i'};

EXAMPLE OF ONE DIMENSIONAL

```
# include <iostream>
using namespace std;
int main()
{
float price[3];
cout <<“price 1:”;
cin>>price[0];
cout <<“price 2:”;
cin>>price[1];
cout <<“price 3:”;
cin>>price[2];
cout<<“TOTAL
PRICE:”<<price[0]+price[1]+price[2];
}
return 0;
}
```

EXAMPLE OF ONE DIMENSIONAL

```
# include <iostream>
using namespace std;
int main()
{
float price[3];
int i;
total=0;
for( i=0;i<3;i++)
{
cout <<“price :“<<0+i<<;
cin>>price[i];

total=total+price[i];
}
cout<<“TOTAL PRICE:”<<total;
return 0;
}
```

TWO DIMENSIONAL ARRAY

- **two-dimensional array has two subscripts**
- **Type ArrayName [sizeofrow][sizeofcolumn];**

  Example : float price[4][5];

INITIALIZING MULTIDIMENSIONAL ARRAYS

```
FLOAT MARK[5][ 2]={5,2.5,4,0,6,8,10,9.5,5.5,6};

INT BIL[3][4]={{1,2,3,4},{11,12,13,14},{21,22,23,2}};

 INT AGE[ 2][  ]={{10,20,30},{10,30,55}};
```

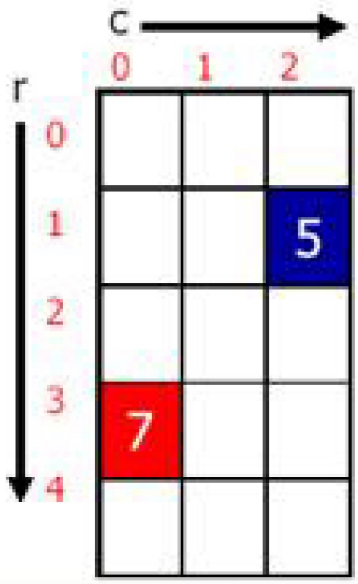
ILLUSTRATE TWO DIMENSIONAL ARRAY

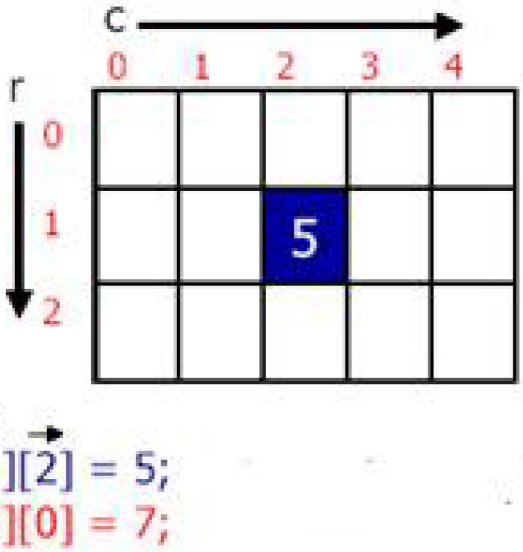
float mark[5][2]={5,2.5,4,0,6,8,10,9.5,5.5,7};

mark[0][0]=5 mark[0][1]=2.5 mark[1][0]=4
mark[1][1]=0 mark[2][0]=6 mark[2][1]=8
mark[3][0]=10 mark[3][1]=9.5 mark[4][0]=5.5
mark[4][1]=7

	[0]	[1]
[0]	5	2.5
[1]	4	0
[2]	6	8
[3]	10	9.5
[4]	5.5	7

ACCESSING A TWO DIMENSIONAL ARRAY



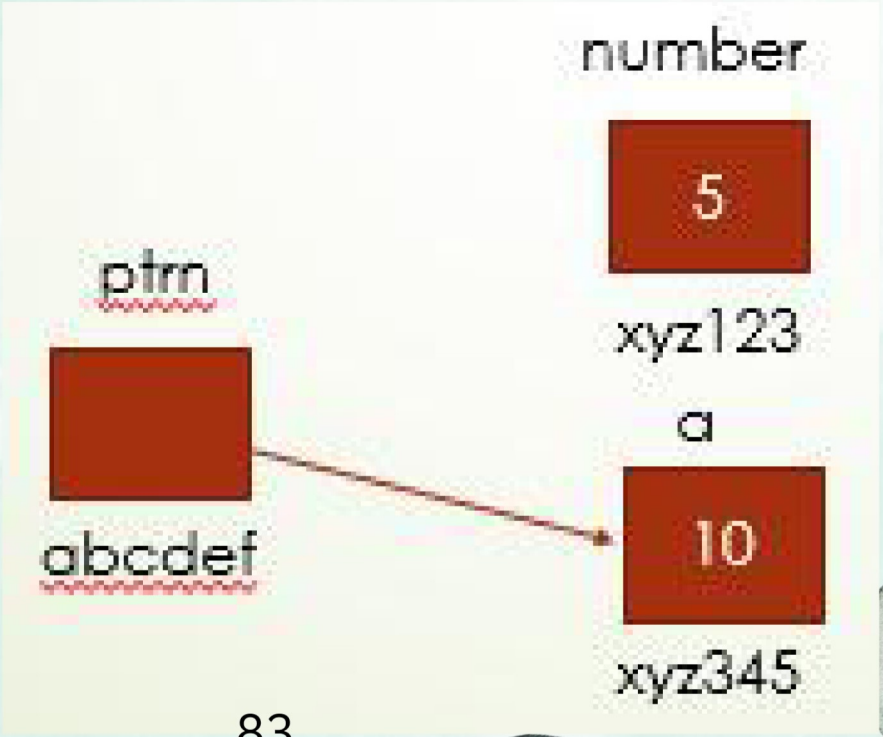


EXAMPLE OF PROGRAM

```
# INCLUDE <IOSTREAM>
USING NAMESPACE STD;
INT MAIN()
{
FLOAT MARK[5][2]={5,2.5,4,0,6,8,10,9.5,5.5,7};
INT I,J;
FOR( I=0;I<5;I++)
{
 FOR(J=0;J<2;J++)
 {
COUT <<“MARK :“<<MARK[I][J]<<ENDL;
 }
}
RETURN 0;
}
```

POINTER

- **A variable which value is used to point to another variable.**
- **Able to assign different values to a pointer variable and the value contained by a pointer must be an address that indicates the location of another variable in the memory.**
- **pointer is also called an address variable.**



WHAT & WHY OF POINTERS

- We use pointers as a technique:
  - to develop lighter & faster programs.
  - to optimally use the memory space.

- Pointers are also names (variable) – special ones.

- Can be likened to representatives of other variables - these include ordinary variables, arrays, structures.

- Instead of directly pass or use the variables, pointers are used – why?
  - Pointers are more lighter than the other variables.
  - Pointers are more flexible – point to variable A then point to B, and point to C, and so on.

- no need to copy the values of the pointed variables, but only addresses(location in RAM).

- If the variables contain data (values), pointers only contain the addresses of the variables that they point to – hence the name is pointer.

DECLARING POINTER

- The general form of a pointer declaration:

```
data type  *pointer_name;
```

- Data type specifies the type of data to which the pointer points.
- Pointer name is the name of the pointer variable
- Asterik (*) indicates the variable as a pointer.

Example :

```
char *ptrc;

int *no;

float *ptrnum2;
```

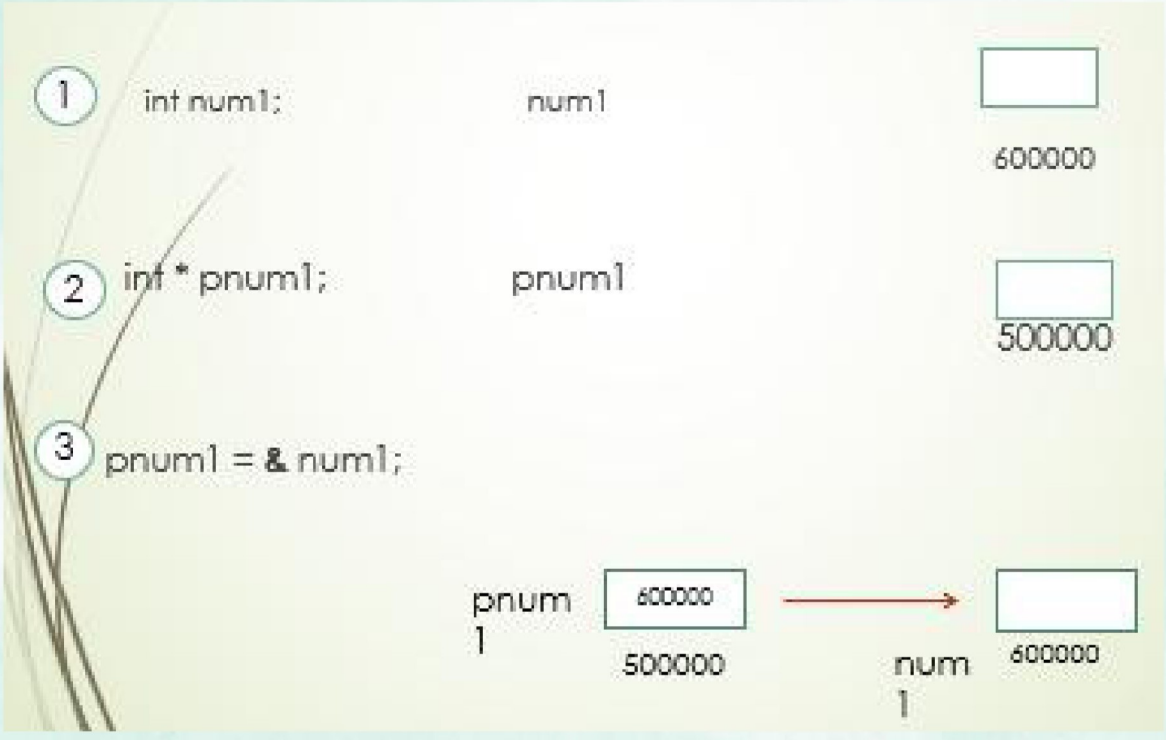
ASSIGN THE ADDRESS OF A VARIABLE TO THE POINTER

syntax:
pointerName=&VariableName;

Example:
ptrc=&c;
no=#
ptrnum2=&num1;

WORKING WITH POINTERS

- **How do we represent num1 (type of int) with a pointer?**



EXAMPLE OF PROGRAM

```
#include<iostream>
using namespace std;
int main()
{
int number=5;
int *ptrn;
ptrn=&number;

cout<<"The value of variable number is:"<<number;
cout<<"\nThe address of variable number is:"<<&number;
cout<<"\nThe address of variable number is:"<<ptrn;
cout<<"\nThe value of variable number is:"<<*ptrn;
cout<<"\nThe address of pointer:"<<&ptrn;
cout<<endl;
return 0;
}
```

```
The value of variable number is:5
The address of variable number is:0012FF7C
The address of variable number is:0012FF7C
The value of variable number is:5
The address of pointer:0012FF6A
```

STORING & DISPLAYING VALUES USING POINTERS

```
int main()
{

int num1, num2;

int * pnum2;
pnum2 = & num2;

num1= 10;
*pnum2 = 20;
*pnum2 = 50;


  cout<<"\nNum1: "<< num1;
cout<<"\nNum2: "<< *pnum2;

return 0;
}
```

```
int main()
 {

 char sym1, sym2;

char * psym2;
psym2 = & sym2;

cin>> sym1;//c
cin>> * psym2;//a

cout<<"\nSymbol 1: sym1;
cout<<"\nSymbol 2: "<< *psym2;

return 0;
}
```

EXERCISE 1:

´Write a program to perform addition of two numbers using pointer. Your program must have two integer variables x, y and two pointer variables p and q. Assign the addresses of x and y to p and q respectively and then assign the sum of x and y to variable sum. Lastly, print the sum of two numbers that entered by the user.

EXERCISE 2:

For each of the following, write a program that performs the indicated task.

- Declare the variable fptr to be a pointer to an object of type float.
- Declare the floating point variables num1 and num2.
- Assign 100.20 to num1 as initial value.
- Assign the address of variable num1 to pointer variable fptr.
- Print the value of object pointed to by fptr.
- Assign the value of the object pointed to by fptr to variable num2.
- Print the value of num2.
- Print the address of num1.
- Print the address stored in fptr.

NEW OPERATOR

- **Provides a dynamic storage allocation in computer memory location.**
- **Can be used to create variables that have no identifiers to serve as their names.**
- **The no identifiers variables are referred to via pointer.**

Syntax:
pointerName=new dataType;

Example:
ptrP=new int;

DELETE OPERATOR

- **Eliminates a dynamic variable and return the memory to the free store.**
- **This operation can be done if a program no longer needs the dynamic variable.**
- **Now the free store memory can be reused to create a new dynamic variable.**

Syntax:

delete pointer_name;

EXAMPLE

```
#include<iostream>
using namespace std;
int main()
{
int *pointer1, *pointer2;

pointer1=new int;
*pointer1=80;

pointer2=pointer1;

cout<<"The value of *pointer1 is:"<< *pointer1<<endl;
cout<<"The value of *pointer2 is:"<< *pointer2<<endl;

delete pointer1;

cout<<"The value of *pointer1 is:"<< *pointer1<<endl;
cout<<"The value of *pointer2 is:"<< *pointer2<<endl;

cout<<"The address of *pointer1 is:"<< &pointer1<<endl;
cout<<"The address of *pointer2 is:"<< &pointer2<<endl;

return 0;
}
```

```
The value of *pointer1 is:80
The value of *pointer2 is:80
The value of *pointer1 is:-572662307
The value of *pointer2 is-572662307
The address of *pointer1 is:0012FF7C
The address of *pointer1 is:0012FF78
```

RELATIONSHIP BETWEEN ARRAYS AND POINTERS

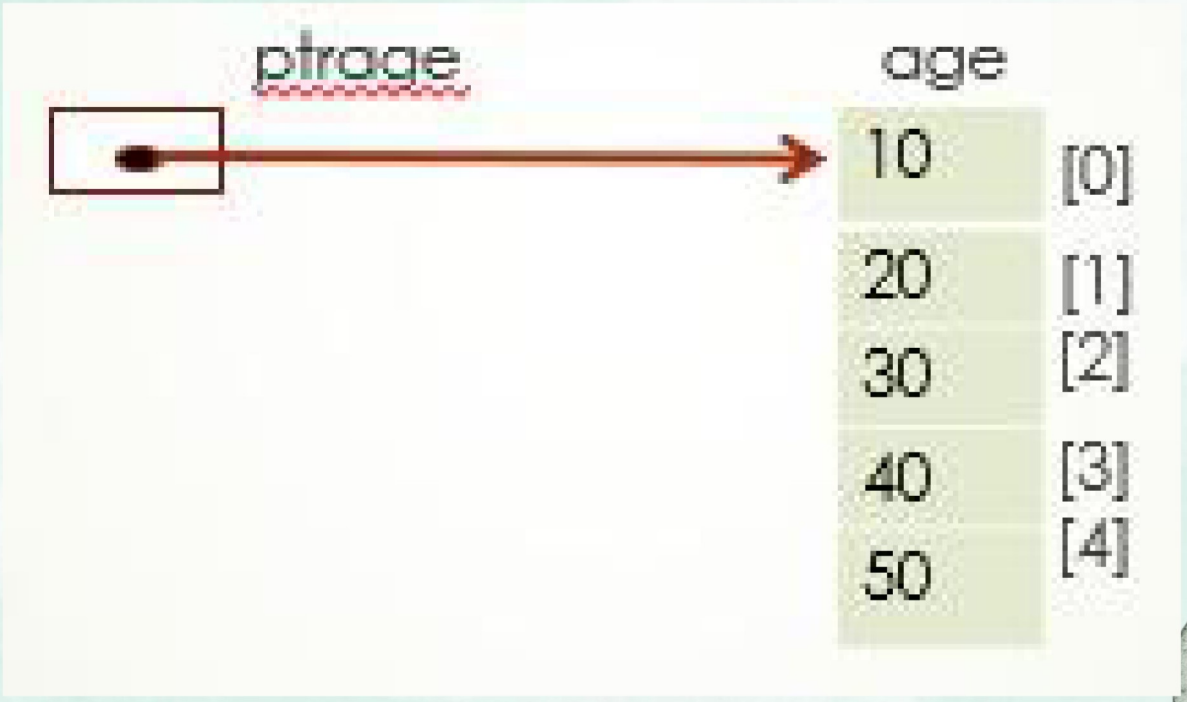
Syntax:

pointerName=arrayName;

Example:

```
int age[5]={10,20,30,40,50};
int *ptrage;
ptrage=age;
```

We can visualize this array like this:



RELATIONSHIP BETWEEN ARRAYS AND POINTERS

- **Array name is starting address of array**

```
int vals[] = {4, 7, 11};
```

starting address of vals: 0x4a00

```
cout << vals;          // displays 0x4a00
cout << vals[0];       // displays 4
```

- **Array name can be used as a pointer constant**

```
int vals[] = {4, 7, 11};
cout << *vals;   // displays 4
```

- **Pointer can be used as an array name**

```
int *ptrv = vals;
cout << ptrv[1];   // displays 7
```

EXAMPLE OF PROGRAM

```
#include<iostream>
using namespace std;
int main()
{
int array[]={9,8,7,6,5,4};

int *pointer;

pointer=array;

for( int a=0;a<6;a++)
{
cout<<"The address :"<< (pointer+a)<<"while the value is:"<<*(pointer+a)<<endl;
}
return 0;
}
```

```
The address: 0012FF68 while the value is: 9
The address: 0012FF6C while the value is: 8
The address: 0012FF70 while the value is: 7
The address: 0012FF74 while the value is: 6
The address: 0012FF78 while the value is: 5
The address: 0012FF7C while the value is: 4
```

Given:

```
int vals[]={4,7,11};
int *ptrv = vals;
```

- **What is ptrv + n?**

```
cout << *(ptrv+1); // displays 7
cout << *(ptrv+2); // displays 11
```

- **Must use ( ) in expression**

ARRAY ACCESS

- **Array elements can be accessed in many ways.**

Array access method	Example
array name and [ ]	vals[2] = 11;
pointer to array and [ ]	ptrv[2] = 11;
array name and subscript arithmetic	*(vals+2) = 11;
pointer to array and subscript arithmetic	*(ptrv+2) = 11;

- **Array notation vals[i]**

- **is equivalent to the pointer notation *(ptrv + i) or ptrv[i]**

POINTER ARITHMETIC

- Some arithmetic operators can be used with pointers:

  - Increment and decrement operators ++, --
  - Integers can be added to or subtracted from pointers using the operators +, -, +=, and -=
  - One pointer can be subtracted from another by using the subtraction operator -

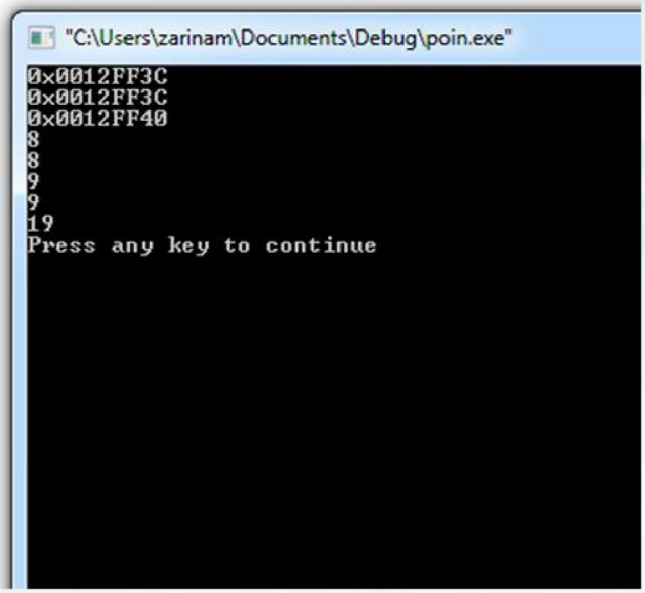
- Assume the variable definitions
  int vals[]={4,7,11};
  int *ptrv = vals;

- Examples of use of ++ and --
  ptrv++; // points at 7
  ptrv--; // now points at 4

- Assume the variable definitions:
  int vals[]={4,7,11};
  int *ptrv = vals;
- Example of use of + to add an int to a pointer:
  cout << *(ptrv + 2)
  - This statement will print 11

EXAMPLE OF PROGRAM

```
#include<iostream.h>

int main()
{

int array[3]={8,9,10};

int *ptrarray;

ptrarray=array;

cout<<array;
cout<<endl;

cout<<ptrarray;
cout<<endl;

cout<<ptrarray+1;
cout<<endl;

cout<<*(ptrarray+0);
cout<<endl;

cout<<ptrarray[0];
cout<<endl;

cout<<*(ptrarray+1);
cout<<endl;

cout<<*ptrarray+1;
cout<<endl;

cout<<*(ptrarray+1)+10;
cout<<endl;

return 0;
```



STRUCTURE

- **Structure is a collection of variable under a single name.**
- **Variable can be of any type: int, float, char etc.**
- **The main difference between structure and array is :**
  - **Array are collection of the same data type.**
  - **Structure is a collection of variable under a single name.**

DECLARING A STRUCTURE

- **The structure is declared by using the keyword struct followed by structure name, also called a tag.**
- **Then the structure members (variable) are defined with their type and variable names inside the open and close braces { and }.**
- **Finally, the closed braces end with a semicolon denoted as ; following the statement.**

EXAMPLE

- **Structure name is customer**

- **Three variables:**
  - **custnum of type int**
  - **salary of type float**
  - **commission of type double**

  } **structure members**

THIS STRUCTURE IS DECLARED AS FOLLOWS:

```
struct customer
{
int custnum;
float salary ;
double commission;
};
```

DECLARE STRUCTURE NAME STUDENT

```
struct student
{
string matric_numb, name
int ic;
float cgpa;
};
```

DECLARE STRUCTURE VARIABLE

- This similar to variable declaration
- For variable declaration, data type is defined followed by variable name.
- For structure variable declaration, the data type is the name of the structure followed by the structure variable name.

```
structure_name   structure_variable _name
```

```
struct customer
    {
     string name;
     int custnum;
     float salary ;
     double commission;
     }; customer cust1;
```

structure variable

ASSIGN VALUE TO A STRUCTURE VARIABLE

THREE ways to assign values to the elements:

i) in the declaration of a record variable

```
customer cust1 = {“akma”,111, 1500, 25.50};
```

ii) direct assignment

```
cust1.name=“akma”;
cust1.custnum = 111;
cust1.salary = 1500;
cust1.commission= 25.50;
```

iii) input statement

```
cin>>cust1.name;
cin>>cust1.custnum;
cin>>cust1.salary;
cin>>cust1.commission;
```

ACCESS VALUES IN STRUCTURE VARIABLE

- ´Elements are accessed by their name.
- Syntax:

  structure_variable.variable_name;
- Example:

  cust1.custnum;
- The dot is called the member selector

ARRAY VS. STRUCTURE

Array	Record
• Homogeneous data structure. • Element access via index. • Is a container for data. • Single and multi-dimensional. • Can hold data only. • Fundamental concept of data structure.	• Heterogeneous data structure. • Element access via name. • Is a user-defined data type. • Single and nested structure. • Can hold data and functions. • Fundamental concept of object oriented programming.

Chapter 5:

FUNCTION

INTRODUCTION OF FUNCTION

- A function is a group of statement to make a large program manageable, programmers modularize them into subprograms. These subprograms are called functions. They can be compiled and tested separately and being reuse in different programs. This modularization is one of the characteristics of successful object-oriented software. There are two types of function which is:

BENEFIT OF USING FUNCTION

- The main program is simplified where it will contain the function call statement. By doing that planning, coding, testing, debugging, understanding and maintaining a computer program will be easier.

- The same function can be reused in another program, which will prevent code duplication and reduce the time of writing the program.

BUILT-IN FUNCTION/ PRE-DEFINED FUNCTION

- Built-in function or predefined function are function used by the programmers in order to speed up program writing. Programmer may use the existing code to perform common task without having to rewrite any code.

- In order for programmer to use pre-defined function, the appropriate library file must be included in the program using the #include directive

- coding example:

```
#include <iostream>
using namespace std;
int main()
{
        char a;
         cout<<“Insert your favaourite character:";
         cin.get(a);
         cout<<"The character is:“;
         cout.put(a);
 return 0;
}
```

Commonly used component

Name	Description
get()	for console input(keyboard) of type char
put()	For console output (screen) of type char

USER DEFINED FUNCTION

- Programmer are able to create their own function since the built-in function in c++ libraries are limited to perform other programming task.

- A user-defined function groups code to perform a specific task. The group of code is given a name by the programmer and defined within the program in which the function is used.

Important component of function

- In order to use user define function, a programmer must satisfy three requirements:

➤ Function prototype
➤ Function definition
➤ Function call

Function prototype

- a declaration for a function that specifies the data type returned by the function, its name, and the data types of the arguments expected by the function. If the called function is placed physically above the calling function, no further declaration is required.

- syntax :

```
return_type function_name(type parameter_list);

Example:

void sum();
int findsum(int.int);
float average(int.int.float.float);
```

Function definition

- A function must be defined before it can carry out the task assigned to it.In other word the statement that perform the task are inside the function definition.
- A function is written once in the program and can be used by any other function in the program.
- The function definition is placed either before the main() or after the main().

BEFORE THE MAIN()

The function prototype is not required.

```
int sum()
{
statement;
}
int main()
{
sum();
}
```

AFTER THE MAIN()

```
int sum()  //function prototype
int main()
{
sum();
}
int sum();
{
statement;
}
```

- syntax :

```
returnType functionName(parameter list)
{
        //statement(s) area;
        return statements
}
```

- returnType - is any valid C++ data type
- functionName - is any valid C++ identifier
- parameter-list - is a list of parameters separated by commas
- statement(s) - is a list of set of instruction or formula end by semi colon
- return statements - is the value returned by the function

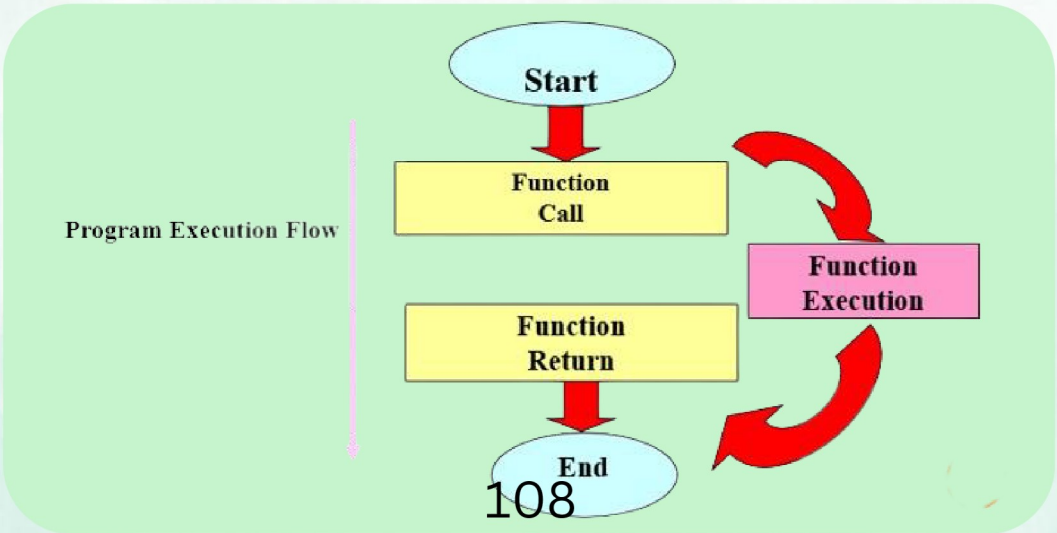
Function call

- To use the function written, you have to call it back in function main().
- When a function is called, the function body is transferred to the called function (function definition) and the task will be exucuted.
- syntax : function_name(parameterArgument)

```
#include <iostream.h>
                Void sum(float,float,float);

                void main ( )
                {
                        float x,y,z;
                        cout <<"enter three numbers:\n" ;
                        cin >>x>>y>>z;
                        sum(x,y,z);
                }
                void sum(float x, float y, float z)
                {
                        cout <<"\n Sum = x + y + z = "<< x+y+z;
                }
```

Making function call

- A function call is an expression that can be used as a single statement or within other statement



PARAMETER

- There are two types of parameters:

➤ Formal parameter
➤ Actual parameter

Formal parameter

- Formal parameter is arguments in the header of function definition.
- Formal parameter contains the value that being passed by actual parameter passed from the function call.
- Example : The following function takes three formal parameters to calculate the sum.

```
void calSum(int num1, int num2, int num3)
{
cout<<"Total:"<<num1+num2+num3;
}
```

Actual parameter

- Actual parameter is a constant , variable or expression in a function call that corresponds to the actual parameter.
- Actual parameters contain values that will be passed to the function definition.
- Example : The following function contains three values which will be passed to the calSum().

```
void main()
{
calSum(10,9,11)
}
void calSum(int num1, int num2, int num3)
{
cout<<"Total:"<<num1+num2+num3;
}
```

Actual parameter

Formal Parameter

PROGRAM SCOPE

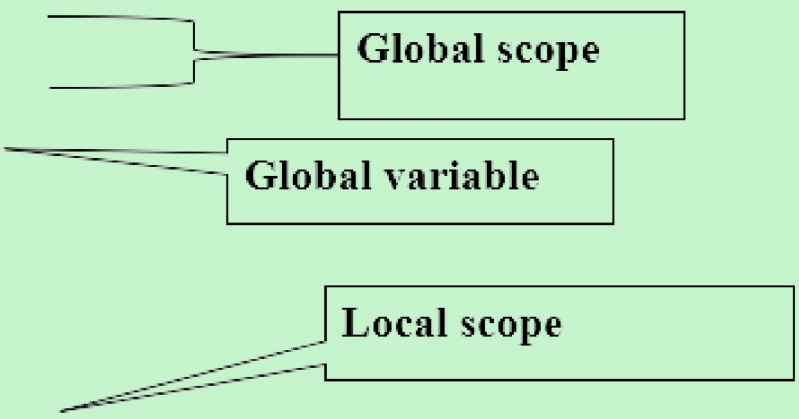
- The scope of an identifier is that part of the program where it can be used. For example variable cannot be used before they are declared, so their scopes begin where they are declared.

FUNCTION SCOPE

- Function scope is the scope of a name consist of that part of the program where it can be used. It begins where the name is declared. If that declaration is inside a function (including the main() function), then the scope extends to the end of the innermost block that contains the declaration.

- Example :

```
void f()
void g()
int x = 11;

main ()
{

int x = 22;
{
int x = 33;
cout << "In block inside main(): x= " << x<<endl;
}
cout << " In main() : x = " <<x<<endl;
cout << " In main (): ::x =" << ::x<<endl;
f();
g();
}
```



LOCAL VARIABLE

- Variable that is declared inside a block. It is accessible only from within that block.

- Example :

```
#include <iostream.h>
float Convert(float);
int main()
{
float TempFer;
float TempCel;
cout << "Please enter the temperature in Fahrenheit: ";
cin >> TempFer;
TempCel = Convert(TempFer);
cout << "\nHere's the temperature in Celsius: ";
cout << TempCel << endl;
return 0;
}

float Convert(float TempFer)
{
float TempCel;
TempCel = ((TempFer - 32) * 5) / 9;
return TempCel;
}
```

ARGUMENT PASSING BY VALUE

- Expression used in the function call is evaluated first and then the resulting value is assigned to the corresponding parameter in the function parameter list before the function begins executing.

- If y has value 9, then the value 9 is passed to the local variable y before the function begins to execute its statements.

- Example :

```
#include <iostream.h>
void sum(int,int);
void main ( )
{
int x,y;
cout <<"enter two numbers:\n" ;
cin >>x>>y;

sum(x,y);
}

void sum(int x, int y)
{
cout <<"\n Sum = x + y = "<< x+y;
 }
```

ARGUMENT PASSING BY REFERENCE

- To pass parameter by reference instead of by value, simply append an ampersand & to the type specifies in the function parameter list. This makes the local variable a reference to the actual parameter passed to it.

- Example :

```
#include <iostream>
using namespace std;
void swap(int&, int&);      // Function prototype
int main()
{
   int a = 1, b = 2;
   cout << "Before swapping" << endl;
   cout << "b = " << b << endl;

   swap(a, b);

   cout << "\nAfter swapping" << endl;
   cout << "a = " << a << endl;
   cout << "b = " << b << endl;
  return 0;
}
void swap(int& n1, int& n2)
{
   int temp;
  temp = n1;
  n1 = n2;
  n2 = temp;
}
```

- ✓ In **main()**, two integer variables are defined.
- ✓ Those integers are passed to a function **swap()** by reference.
- ✓ Compiler can identify this is pass by reference because function definition is **void swap(int& n1, int& n2)** (notice the **&** sign after data type).
- ✓ Here, **n1** and **n2** are the different names given to the argument passed, that is **n1** and **a** is exact same variable (not only their value is same but both name refer to one single variable).
- ✓ Similarly, **n2** and **b** is exact same variable.

output:

```
Before swapping
a = 1
b = 2

After swapping
a = 2
b = 1
```

RETURN(VOID)

- Functions return a value or return void. Void is a signal to the compiler that no value will be returned.

```
return 3;
return (y > 3);
return (Function1());
```

These are all legal return statements, assuming that the function Function1( ) itself returns a value. The value in the second statement, return (y > 3), will be zero if y is not greater than 3 or it will be 1. What is returned is the value of the expression, 0 (false) or 1 (true), not the value of x.

- To return a value from a function, write the keyword return followed by the value you want to return. The value might itself be an expression that returns a value.

FUNCTION AND POINTER

```
#include <iostream>
using namespace std;
void swap(int*, int*);      // Function prototype
int main()
{
   int a = 1, b = 2;
   cout << "Before swaping" << endl;
   cout << "a = " << a << endl;
   cout << "b = " << b << endl;
   swap(&a, &b);        // &a is address of a and &b is address of b

   cout << "\nAfter swaping" << endl;
   cout << "a = " << a << endl;
   cout << "b = " << b << endl;
   return 0;
}
void swap(int* n1, int* n2)
 {
   int temp;
   temp = *n1;
   *n1 = *n2;
   *n2 = temp;
}
```

Output:

Before swapping
a = 1
b = 2

After swapping
a = 2
b = 1

Since the address is passed instead of value, dereference operator must be used to access the value stored in that address.

The *n1 and *n2 gives the value stored at address n1 and n2 respectively.

Since n1 contains the address of a, anything done to *n1 changes the value of a in main() function as well. Similarly, b will have same value as *n2.

ARRAY

- At some point, we may need to pass an array to a function as a parameter.
- In C++, it is not possible to pass the entire block of memory represented by an array to a function directly as an argument but what can be passed instead is its address.
- In practice, this has almost the same effect, and it is a much faster and more efficient operation.
- To accept an array as parameter for a function, the parameters can be declared as the array type, but with empty brackets, omitting the actual size of the array.
- Example :

```
// arrays as parameters
#include <iostream>
using namespace std;

void printarray (int arg[], int length)
{
 for (int n=0; n<length; ++n)
   cout << arg[n] << ' ';
  cout << '\n';
}

int main ()
{
  int firstarray[] = {5, 10, 15};
  int secondarray[]= {2, 4, 6, 8, 10};
  printarray (firstarray,3);
  printarray (secondarray,5);
}
```

Exercise Chapter 5

1. Observe the function below:

```
float calculate(int hours, int total)
{
      float salary;
      if (hours <= 40)
         salary = 4.5 * hours
      else
         salary = (hours – 40) * 5.5 + 4.5 * 40;
       if (total > 100)
         salary += (total – 100) * 3.0;
       return (salary);
 }
```

What value will the function return if it is called using the values stated below:
(a) calculate(35, 120);
(b) calculate(50, 100);

2. Observe the function below:

```
int func(int p, int q, char r)
{
      int n = 0, I , j = 1;
      switch©
     {
     case '+' : for (i = p; i<=q; i++)
          n += q;
          return n;
     case '*' : for (i = p; i<=q; i++)
           j * = i;
          return j;
     case '^' : for (i = p; i<=q; i++)
          j = j*p;
         return j;
   }
   return (-1);
}
```

For each of the statements below, what is the value of total

(a) total = func(5, 5 ,'^');
(b) total = func(12, 18 ,'+');
(c) total = func(2, 6 ,'*');

Exercise Chapter 5

3. Given a function as shown below:

```
float count_dividend(float savings)
{
    float div;
    div = savings * 0.16;
    return div;
}
```

(a) Give an example of how to call this function.
(b) Give an example of the prototype for this function.

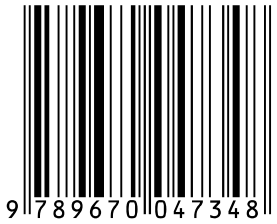
4. Give the header and the prototype for the functions below:
   (a) Minimum function that will receive three integer numbers x, y and z and returns the smallest integer
   (b) Minimum function that will receive three integer numbers x, y and z and displays the smallest integer.
   (c) Average function that will receive two float numbers a, b and returns the average of a and b.

5. Write a function that will display a square using '*'. This function will receive an integer value which is the width and will display the square using the width value. If the width is 4, then the function will display:

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

LET'S LEARN C++

e ISBN 978-967-0047-34-8



POLITEKNIK SULTAN MIZAN ZAINAL ABIDIN

(online)