Computer Programming C++ (66111)

Instructors: Dr.Laui Malhis
Miss.Haya Sammaneh
Eng. Muhannad Al-Jabi
Eng.Anas Toameh

What Is a Computer?

- Computer programs
  - It is a set of ordered instructions written by people called computer programmers to do a certain task.

Instruction A is executed before instruction B, as long as instruction A is located before instruction B.

- Software
  - Instructions to command computer to perform actions and make decisions
- Hardware
  - Various devices comprising (making) computer
    - Keyboard, screen, mouse, disks, memory, CD-ROM, processing units, etc.

Computer Organization

- Five logical units of computer
  - **Input unit**
    - Obtains information from input devices
      - Keyboard, mouse, microphone, scanner, etc.
  - **Output unit**
    - Places information processed by computer on output devices
      - Screen, printer, etc.

3

Computer Organization (Cont.)

- Five logical units of computer (Cont.)
  - **Memory unit (RAM-Random Access Memory)**
    - Rapid access
    - Relatively low capacity
    - Often called memory or primary memory
  - **Secondary storage unit**
    - Long-term
    - high-capacity
    - Stores inactive programs or data
    - Secondary storage devices
      - Hard drives, CDs, DVDs
    - Slower to access than primary memory
    - Less expensive per unit than primary memory

4

Computer Organization (Cont.)

- Five logical units of computer (Cont.)
  - **Central processing unit (CPU)**
    - supervises other sections of computer
    - Used to fetch an instruction from memory and executes it.
    - consists of :
      - Registers (each of which can hold a number)
      - Control unit (CU)
      - Arithmetic and logic unit (ALU) : Performs arithmetic calculations and logic decisions

5

Machine Languages, Assembly Languages and High-Level Languages

- Three types of computer languages
  - Machine language (Low Level Language)
    - Only language computer directly understands
    - Generally consist of strings of numbers 0s and 1s
  - Assembly language (Low Level Language)
    - not understand to computers
      - Convert to machine language by translator programs (assemblers)
    - Example
      -
```
load    a
add     b
store   z
```

6

Machine Languages, Assembly Languages and High-Level Languages (Cont.)

- Three types of computer languages (Cont.)
  - High-level languages
    - Similar to everyday English
      - Uses common mathematical notations
    - Single statements to do some tasks
    - Converted to machine language by translator programs (compilers)
    - Example
      - `z = a + b`

7

How computers work

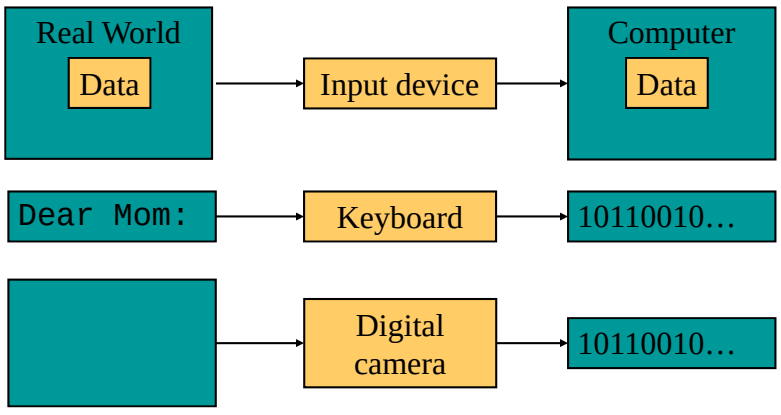
- Modern computers have several components; CPU, RAM, Hard disk, and inputs/outputs (I/O)devices.
- The role of the central processor unit (CPU) is to fetch an instruction from memory and executes it.
- The CPU has its own small workspace, consisting of several registers, each of which can hold a number.
- One register holds the memory address of the next instruction, and the CPU uses this information to fetch the next instruction.
- After it fetches an instruction, the CPU stores the instruction in another register and updates the first register to the address of the next instruction.
- Everything stored in a computer is stored as a number.
- computer programs have to be expressed as machine language.

8

Numbers and Number Systems

9

Introduction



10

Number Systems

- Decimal -- 10 symbols (0,1,2,3,4,5,6,7,8,9)
- Binary -- 2 symbols (0,1)
- Octal -- 8 symbols (0,1,2,3,4,5,6,7)
- Hexadecimal -- 16 symbols (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F) , A = 10 , B = 11 ….F = 15

11

Converting with fraction, examples

- Binary to Decimal

Ex: $(1101.1)_2 \rightarrow 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} = (13.5)_{10}$

- Octal to Decimal

Ex: $(673)_8 \rightarrow 6* 8^2 + 7*8^1 + 3*8^0 = (443)_{10}$

- Hexadecimal to Decimal

Ex. $(A9C)_{16} \rightarrow A* 16^2 + 9*16^1 + C * 16^0 =$

$10* 16^2 + 9*16^1 + 12 * 16^0 = (2788)_{10}$

12

Converting, example

- Binary to Octal

Ex: $(011\underline{011}.\underline{101}100)_2 \rightarrow (33.54)_8$

- Binary to Hexadecimal

Ex: $(1111\underline{1011}.\underline{1101}1000)_2 \rightarrow (FB.D8)_{16}$

- Hexadecimal to Binary

Ex. $(A3.B)_{16} \rightarrow (10100011.1011)_2$

13

Converting, example

- Decimal to Binary

EX: $(12.3)_{10} \rightarrow (1100.01001)_2$

12 / 2 = 6 (Remainder 0 (right))
6 / 2 = 3 (Remainder 0)
3 / 2 = 1 (Remainder 1)
1 / 2= 0 (Remainder 1 (left))

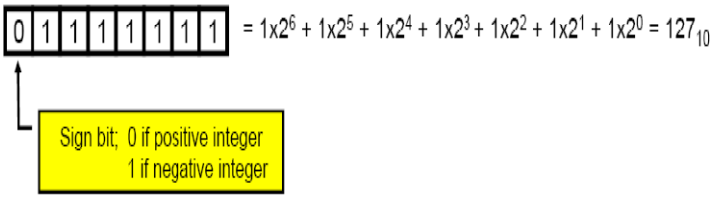
0.3 * 2 = 0.6 (left)
0.6 * 2 = 1.2
0.2 * 2 = 0.4
0.4 * 2 = 0.8
0.8 * 2 = 1.6 (right)

Using the same method to convert the Decimal number to any base.

14

Signed Numbers

- One byte of data can be used to positive store integers up to:



15

Ways to represent negative numbers

- Negative integers are stored via two's complement representation
- 1's complement
  - reverse the bits to get the negative
  - Ex: 1101 → 1's complement → 0010
- 2's complement
  - It happens by reversing the bits (1's complement) then adding 1.
  - Ex: 1101 → 2's complement → 0011
  - Get 1's complement → 0010
  - then add 1 → 0011
  - Example: -7

  (positive value) 7→ (8-digit binary value)00000111→
  (1's complement)11111000→ (2's complement) 11111001

  -7 is stored in computer as 11111001

16

Binary arithmetic – addition and multiplication

- **0 + 0 = 0**
- **0 + 1 = 1**
- 1 + 0 = 1
- 1 + 1 = 0 and carry 1

- Overflow: If there is not enough room to hold the result correctly.
  - If the two numbers are of opposite signs, no overflow can occur. (Why not?)

(Result is smaller than one of them)

- **multiplication**
  - **0 * 0 = 0**
  - **0 * 1 =0**
  - **1 * 0 = 0**
  - **1 * 1 = 1**

17

Concepts of bit, byte and word

Bit:
- is the smallest data item in computers
- **"binary digit" .**
- Can have value 0 or 1.

Bytes:
- are composed of 8 bits

Words:
- The size of a word varies from one computer to another, depending on the CPU.
- For computers with a 16-bit CPU, a word is 16 bits (2 bytes).

18

Standard Alphanumeric Formats

- Problem : Representing text strings, such as
  "`Hello, world`", in a computer

- The standards for representing letters (alpha) and numbers
  - ASCII – American standard code for information interchange

19

Character Code: ASCII and Unicode

- ASCII and Unicode are two computer 'languages' for naming letters
  - The ASCII name for 'a' is '61'
  - Unicode

20

ASCII

- Most widely used coding scheme
- Computer systems can represent up to 256 letters
  - Technical detail: with one 8-bit byte ($2^8 = 256$)
  - ASCII only uses 7 bits ($2^7 = 128$)

21

ASCII Reference Table

Control , Numeric, Alphabetic, Punctuations Codes

MSD / LSD	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P		p
1	SOH	DC1	!	1	A	Q	a	W
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACJ	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

74_{16}

111 0100

22

Unicode: fixing the ASCII problem

- Most common 16-bit form represents 65,536 characters.
- Multilingual: These characters cover the principal written languages of the Americas, Europe, the Middle East, Africa, India, Asia, and Pacifica.

23

Introduction

- C has become one of the most important and popular programming languages.
- ANSI (American National Standards Institute) and ISO (International Organization)standard for C
- Many have moved from C to the C++ language.
- C programs tend to be compact and to run quickly.
- C is a portable language.
- C is powerful and flexible.

24

Introduction-cont.

- C++ brings object-oriented programming tools to the C language.
  - Objects: reusable software components
  - Object-oriented programs
    - Easier to understand, correct and modify
- C++ is nearly a superset of C, meaning that any C program is, or nearly is, a valid C++ program, too.

25

Other Programming Languages

- Java
- FORTRAN
- COBOL
- Pascal
- .NET platform
- Visual Basic .NET
- Visual C++
- C#
  - Based on C++ and Java

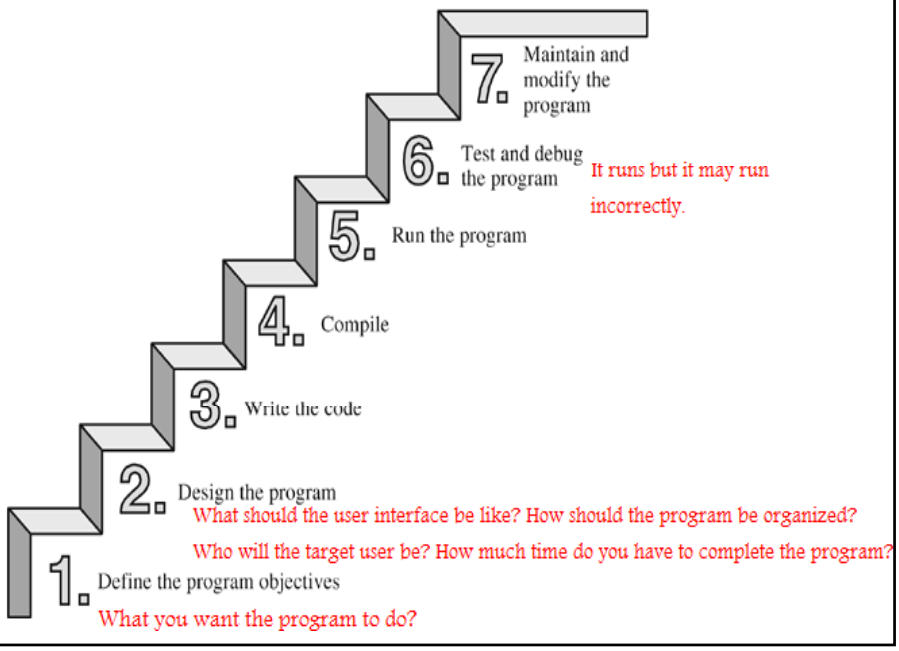
26

Before you begin

- When you write a program in the C language, you store what you write in a text file called a source code file.
- The name of the file end in .cpp
- The part of the name before the period is called the base name, and the part after the period is called the extension.

27

Seven Steps to write a good program.



Program Design-step 2

- Find a suitable algorithm to solve the problem.
- Draw the flowchart of that algorithm.

What does it mean?

29

Program design…

Flowchart

- Flowchart is the step-by-step solution of a problem, using suitably geometric figures connected by flow lines for the purpose of designing.
- Such that, program instructions are categorized into different categories, and each category has different geometric figure.
- Each instruction is presented by the geometric figure of its category.

30

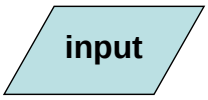
Program design…

Flowchart

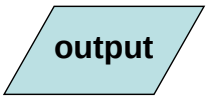
Instruction category — Geometric figure

- Starting program

- Input



- Output



31

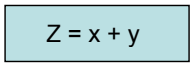
Program design…

Flowchart

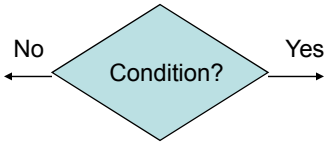
Instruction category

- Process

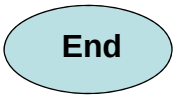
- Geometric figure



- Conditional instruction



- End program



32

Program design…

Flowchart example 1

Draw the flowchart for a program that reads two numbers and prints the sum of these two numbers?

- The program must read two numbers.
- But where can the program store these numbers?
  - The answer is, in two memory locations have been previously allocated by a process called Declaration.

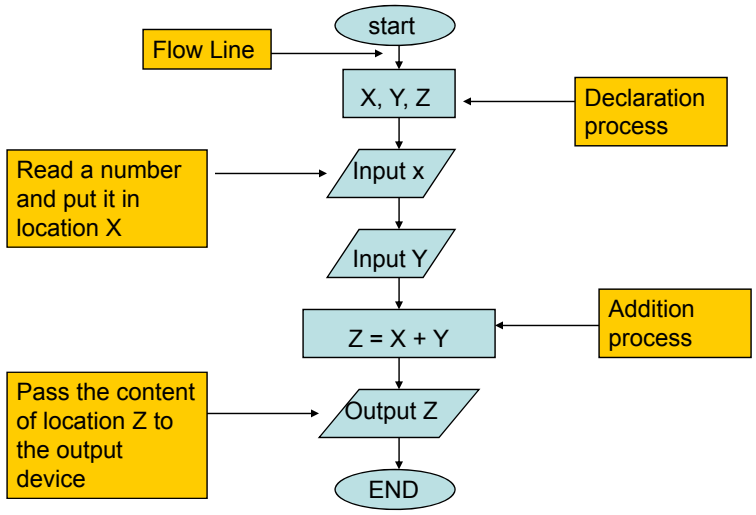
locate a location in memory and assign it a name for later use

- The sum can be calculated and stored in another location that has been declared.
- Now the program can print the result by passing it to the output device.

33

Program design…

Flowchart example 1

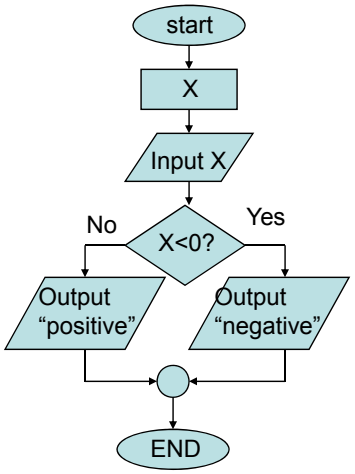


34

Program design…

Flowchart example 2:

Draw the flowchart for a program that reads a number, and determines whether the number is positive or negative.



Why did we put the quotations around the words positive and negative?

When we need to print a word or a sentence as it is, we must put it between quotations.

35

Program design…

Flowchart example 3

Draw the flowchart for a program that evaluates the average for N input numbers

- How many memory locations do we need?
- Do we need N memory locations?
- Take in consideration that N is an input number decided by the user. So, the value of N is user dependent.

36

Flowchart example 3

- From the equation Avg = Sum / N , we can decide that we need 3 locations to start with:
- Avg
- Sum
- N

Flowchart example 3

Since we can't allocate N locations, we need a location x to use while reading the N numbers.

And repeat the following statements for each number:-

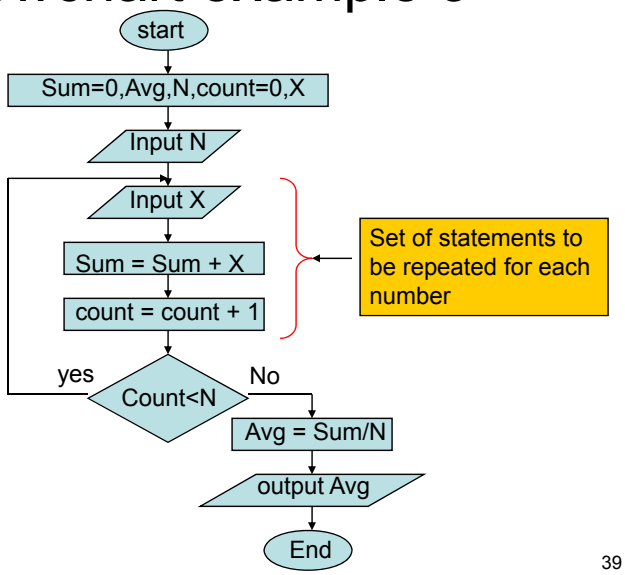
- Read the number and put it in location x.
- Add the content of x to the content of sum.

- How do we know that we have repeated the previous two statements for N times?

- We need a counter that already initialized to zero, and increment the counter for each number.

Program design…

Flowchart example 3



39

Before you begin – cont.

- Bellow is a C program that prints the sentence “My First Program” on the screen.

```
#include <stdio.h>

int main(void)

{

    printf("My First Program.\n");


    return 0;

}
```

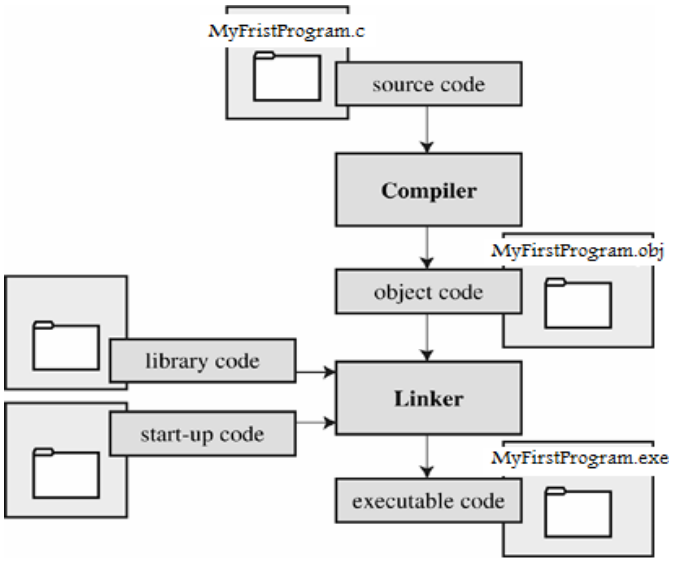
40

Before you begin – cont.

- After writing the source code in a text file and saving it in a proper extension, you need one of the programs that convert your source code file to an executable file (a file containing machine language code).
- These programs do this in two steps:
  - Compiling: converts your source code to an intermediate code (object file).
  - Linking: combines the intermediate code (object file) with other code to produce the executable file.
- This is good because you can compile individual modules separately and then use the linker to combine the compiled modules later.

41

Before you begin – cont.



42

Program writing

- We will use Microsoft Visual Studio 6.0.

It can be used for:-

- Program writing.
- Program compilation.

43

Program writing…

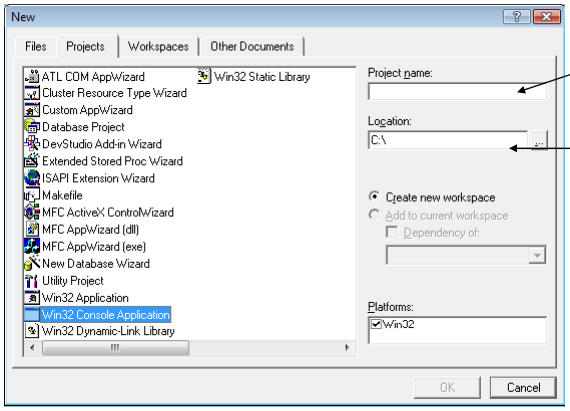
How to start writing a program?

- Go to all programs menu from desktop.
- Select Microsoft Visual Studio 6.0.
- Select Microsoft Visual C++ 6.0
- From file menu select new.
- Select the projects tap.
- Select the win32 console application

44

Program writing…

How to start writing a program?



The project name

Where do you want to store your project?

After this point always press the ok bottom

45

Program writing…

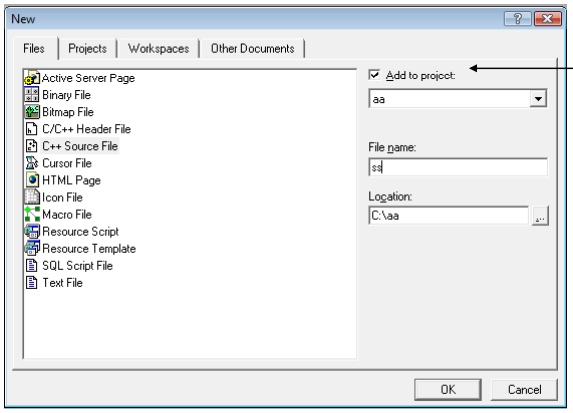
How to start writing a program?

- From the file menu select new.
- Select the file tap.
- Select the c++ source file option.

46

Program writing…

How to start writing a program?



Don't forget to tick this, in order to specify to which project does your file belong.

Any project may contain many files. But only one can contain the main function.

The extension of any C++ source file is cpp

47

Program writing…

Example #1

write a C++ program that prints a hello word on the screen?

48

Typical C++ Development Environment

- Input/output
  - `cin`
    - Standard input stream
    - Normally inputs from keyboard
  - `cout`
    - Standard output stream
    - Normally outputs to computer screen

49

Input/Output

- I/O objects **cin, cout**
- Defined in the C++ library called <iostream.h>
- Must have these lines (called pre-processor directives) :
  - #include <iostream.h>

50

Output

- Use ">>"
- What can be outputted?
  - Any data can be outputted to display screen
    - Variables
    - Constants
    - Literals
    - Expressions (which can include all of above)
  - cout << x << " games played.";
- Cascading: multiple values in one cout
  cout<<x<<y<<z;

51

Input Using cin

- cin for input
- Differences:
  - ">>"
  - Object name "cin" used instead of "cout"
  - No literals allowed for cin (cin>>3; // error)
    - Must input "to a variable“
    - cin >> num;
  - Waits on-screen for keyboard entry
  - Value entered at keyboard is "assigned" to num

52

Program writing…

Example #1

```
#include<iostream.h>
void main()
{
cout<<“Hello”;
}
```

The main function is part of every C++ program. And is called the main block.

This file must be included for any program that outputs data to the screen or inputs data from the keyboard.

C++ programs begin executing at the first statement after the left brace of the main and finish executing at the right brace of the main if there is no return statement.

53

Program writing…

Example #1

```
#include<iostream.h>
int main()
{
cout<<“Hello”;
return 0;
}
```

The main function is either of type int and contains the return statement or of type void and has no return statement. return 0 indicates that the program ended successfully.

There is must be a semicolon (;) at the end of each C++ statement

54

Program writing…

Escape sequence

- In some cases you want to break a sentence into many lines on the screen.
- For example, you want to print the “hello” word something like this:
  he
  llo
  Which means that the hello word has been broken into two lines.

55

Program writing…

Escape sequence

C++ statement contains escape sequence	output
cout<<“hello\n word”;	hello word
cout<<“hello\t word”;	Hello word
cout<<“hello\r word”;	word
cout<<“hello\a”;	hello You will hear a beep
cout<<“hello\\word”;	hello\word
cout<<“hello\” word”;	hello “ word
cout<<“hello”<<endl<<“word”;	hello word

56

Program writing…

Single line and multi-lines comments

- If we want that the compiler to ignore a certain line, we must put // before that line.
- If we want that the compiler to ignore a group of lines, we must put /* before the beginning of the first line. And we must put */ after the end of the last line.
- Commenting a program is useful for program readability

57

Data types,
Variables and Constant
Variable Declaration
Assignment Statement
Reading and writing variables
First C program

58

Data Types : 1- Integer

- An integer type is a number without a fractional part.
- Designed to hold whole numbers
- Can be `signed` or `unsigned`:
  - `12  -6  +3`
- Available in different sizes (number of bytes): `short int`, `int`, and `long int`
- Size of `short int` ≤ size of `int` ≤ size of `long int`

59

Declaration of Integer Variables

- Variables: locations in memory where values can be stored.
- Declarations tell the compiler what variable names will be used and what type of data each can handle (store).
- Variables of integer type can be defined
  - On separate lines:

```
int length;
int width;
unsigned int area;
```

  - On the same line:

```
int length, width;
unsigned int area;
```

60

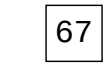
Data Type: 2- character

- Used to hold characters like ‘d’
- Numeric value of character is stored in memory:

CODE:

```
char letter;
letter = 'C';
```

MEMORY:



61

Declaration of character Variables

- Variables of character type can be defined:
  - On separate lines:

    ```
    char x;
    ```

  - On the same line:

    ```
    char x, y;
    ```

62

Data Types: 3- Floating-Point

- A floating-point type is a number with a fractional part
- Designed to hold real numbers

  `12.45` `-3.8`
- All numbers are signed
- Available in different sizes (number of bytes): `float`, `double`, and `long double`
- Size of `float` ≤ size of `double`
  ≤ size of `long double`

63

Declaration of floating point Variables

- Variables of floating point type can be defined:
  - On separate lines:

```
double x;
float y;
long double z;
```

  - On the same line:

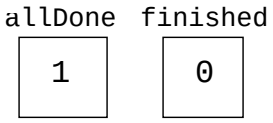
```
double x, y;
float y , e;
long double z , r;
```

64

Data Types: 4- The `bool`

- Represents values that are `true` or `false`
- `bool` variables are stored as small integers
- `false` is represented by 0, `true` by 1:
- bool declarations:

```
bool allDone = true;
bool finished = false;
```



65

Data Type: 5-void

The void type has no values and no operations.

66

Sizes of Data types

The function **sizeof**(***data type***) can be used to see the size

Type	Size in Bytes
`char`	1
`short`	2
`int`	4
`long`	4
`float`	4
`double`	8
`bool`	1

67

Program writing…

Variable naming

Variable name must start with

- Letter

or

- Underscore

or

- $

But, the rest of the variable name may contain letters, underscores, dollar signs or digits

-Variable name cannot start with a digit

-Cannot use C++ key words

68

Valid and Invalid Identifiers/Variable

Variable	VALID?	REASON IF INVALID
`totalSales`	Yes	
`total_Sales`	Yes	
`total.Sales`	No	Cannot contain `.`
`4thQtrSales`	No	Cannot begin with digit
`totalSale$`	No	Cannot contain $

69

Program writing…

Variable declaration

- You can declare the variable anywhere you want in your program.
- You must declare the variable before you can use it.
- You can declare each variable in one declaration statement.
- Also you can declare multiple variables of same type in one declaration statement.

70

Program writing…

Declaration statement structure

- Single variable declaration statement.
  data_type variable_name;
  Example : int x;

- Multiple variables declaration statement
  data_type var1,var2,var3,…….varn;
  Example : int x,y,z;

- You can give the initial value to the variable (initialize the variable) while declaring it
  Example: int x=6;

71

Program writing…

Example #2

Write a c++ program that reads two integer numbers and prints the result.

- The program must tell the user when to enter each number by print a message likes “enter the first number now please”.
- The program must print the result in this manner :
  the result = 12233……

72

Program writing…

Example #2

```
#include <iostream.h>
void main(){
int num1,num2, result;
cout<<“enter the first number plz”<<endl;
cin>> num1;
cout<<“enter the second number plz”<<endl;
cin>>num2
result = num1+num2;
cout<<“the result  =\t”<<result<<endl;
}
```

declaration

Input the first number and put it in location num1

Pass the content of location result to the output device

73

Program writing…

Example #3

What is wrong in this code (part of a program)?

```
int x=15;
X=10;
Cout<<x<<endl;
```

X is not declared, X is not the same as x

Cout is not the same as cout

So, C++ is case sensitive language

74

Keywords and Identifier

Some Keywords in C and C++ :

asm	**double**	new	**switch**
auto	**else**	operator	template
break	**enum**	private	this
case	**extern**	protected	throw
catch	**float**	public	try
char	**for**	**register**	**typedef**
class	friend	**return**	**union**
const	**goto**	**short**	**unsigned**
continue	**if**	**signed**	virtual
default	inline	**sizeof**	**void**
delete	**int**	**static**	**volatile**
do	**long**	**struct**	**while**

75

Declarations

- Constants and variables must be declared before they can be used.

A constant declaration:

- specifies the type, the name and the value of the constant.
- any attempt to alter the value of a variable defined as constant results in an error message by the compiler

A variable declaration:

- specifies the type, the name and possibly the initial value of the variable.

- When you declare a constant or a variable, the compiler:
  1. Reserves a memory location in which to store the value of the constant or variable.
  2. Associates the name of the constant or variable with the memory location.

76

Constant Variables

(C++ style constants)

•Example:

const double PI = 3.14159;

const double PI ; // Error must have value

(old C style constants)

• constants can also be specified using the preprocessor directive #define example:

#define PI 3.14159

the preprocessor replaces the identifier PI by the text 3.14159 throughout the program

• the major drawback of #define is that the data type of the constant is not specified

77

Constant declarations

- Constants are used to store values that never change during the program execution.
- Using constants makes programs more readable and maintainable.

Syntax:

```
const <type> <identifier> = <expression>;
```

Examples:

```
const double x = 7.8;

const double r = x * 2;
```

78

Variable declarations,cont.

- A variable has a type and it can contain only values of that type.
- For example, a variable of the type `int` can only hold integer values.
- Variables are not automatically initialized. For example, after declaration

```
int sum;
```

the value of the variable **sum** can be anything (garbage).

79

Character data

- A variable or a constant of char type can hold an ASCII character.
- When initializing a constant or a variable of char type, or when changing the value of a variable of char type, the value is enclosed in single quotation marks.

Examples:

```
const char star = '*';
char  letter, one = '1';
```

80

Variable Assignments and Initialization

Assignment:

- Uses the = operator
- Has a single variable on the left side and a value (constant, variable, or expression) on the right side
- Copies the value on the right into the variable on the left:

```
item = 12;
```

81

Variable Assignments and Initialization

- Initialize a variable: assign it a value when it is defined:

```
int length = 12;
```

- Can initialize some or all variables:

```
int length = 12, width = 5, area;
```

82

Arithmetic Operators

- Used for performing numeric calculations
- C++ has unary, binary, and trinary operators:
  - unary (1 operand) `-5`
  - binary (2 operands) `13 - 7`
  - ternary (3 operands) `exp1 ? exp2 : exp3`

83

Binary Arithmetic Operators

SYMBOL	OPERATION	EXAMPLE	VALUE OF `ans`
+	addition	`ans = 7 + 3;`	10
-	subtraction	`ans = 7 - 3;`	4
*	multiplication	`ans = 7 * 3;`	21
/	division	`ans = 3 / 7;`	0
%	Modulus-returns remainder	`ans = 7 % 3;`	1

84

Arithmetic Operations

• multiplication, summation, subtraction, division

```
int i = 1/3;      // 0
float x = 1.0/3;  // 0.3333
int j = 7 % 3;   // 1
```

• prefix and postfix-increment operator ++

```
int i=3;
int j=7;
cout << 10 * i++;  // outputs 30, i has value 4 afterwards
cout <<  10 * ++j; // outputs 80, j has value 8 afterwards
```

• arithmetic assignment operators

```
float x=6.0;
x+=3.5;
```

• is equivalent to x=x+3.5;

85

/ Operator

- / (division) operator performs integer division if both operands are integers

```
cout << 13 / 5;    // displays 2
cout << 91 / 7;    // displays 13
```

- If either operand is floating point, the result is floating point

```
cout << 13 / 5.0;  // displays 2.6
cout << 91.0 / 7;  // displays 13.0
```

86

% Operator

- % (modulus) operator computes the remainder resulting from integer division

```
cout << 13 % 5;   // displays 3
```

- % requires integers for both operands

```
cout << 13 % 5.0; // error
```

87

Program writing…

Integer division and float division

- int / int :the result is integer
- float /float : the result is float
- int/float: the result is float
- float/int: the result is float

Example: 6/4 is 1
but the result of 6.0/4 is 1.5

88

Program writing…

Casting

- Treat a variable of type1 as if it is of type2.

  (**type2**)*varibale of type1*
- It is mostly used when we don't need an integer division to truncate the remainder.

Example: int x=5,y=6;
float z;
z=(float)y/x;

Treat y as float so the division is float division and z=1.2.

If we remove the casting, the division becomes integer division and z=1;

89

Program writing…

ASCII values

- Each character can be treated as integer or as character.
- Each character is a one byte integer.
- Each character has its ASCII value.
  'a'…'z' -> 97…122
  'A'…'Z' -> 65…90

Example: cout<<(char)65; will print A
cout<<(int)'a'; will print 97

90

Character

- **Given:**

'A' = 65 in decimal = 41 hex

'A' = 97 in decimal = 61 in hex

- Given char A= 'F'; cout<< (char) ('A'+3);
- Answer: 'D'
- Given char A= 'F'; cout<< ('A'+3);
- Answer: 68
- Given char A= 'B'; cout<< (char)(A+3);
- Answer: 'E'
- Given char A= 'B'; cout<< (A+3);
- Answer: 69

91

Program writing…

Example #4

Write a C++ program that reads a small letter character and prints the upper case of that character.

92

Example #4

```
#include<iostream.h>
void main(){
char x;
cin>>x;
cout<<(char)(x-32)<<endl;
}
```

32 is the difference between the uppercase and the smaller case ranges

When a character exists in a mathematical equation, its ASCII value is automatically substituted

93

Program writing…

Assignment operators

- Addition assignment operator

  c = c + 3 can be written as c += 3
- Other operators can be written in the same manner

c = c *3	c*=3
c = c – 3	c-=3
c = c / 3	c/=3
c = c % 3	c%=3

94

Program writing…

Increment and decrement operators

- Increment operator is used to increment the variable by one in the form like

  x++ or ++x

  This is equivalent to x+=1 and x=x+1

- Decrement operator is used to decrement the variable by one in the form like

  x-- or --x

95

Program writing…

Preincrement and postincrement

- Preincrement: variable changed before used in expression. ++x
- Postincrement: variable changed after used in expression. x++

96

Program writing…

Preincrement and postincrement

- What is the output of the following code?

```
int x = 3;
cout<<x++<<endl;
cout<<++x<<endl;
```

3, because x is incremented after the first print operation

5 ,because after the first print operation x is 4

And now x is incremented before the second print operation to become 5 and it is printed as 5

97

Post/Pre-Increment in Action

- Post-Increment in Expressions:
- ```
  int n = 2;
  int y;
  y = 2 * (n++);
  cout << y << endl;
  cout << n << endl;
  ```
  - This code segment produces the output:
    4
    3
- Pre-Increment in Expressions:
  ```
  int n = 2,
  int y;
  y = 2 * (++n);
  cout << y << endl;
  cout << n << endl;
  ```
  - This code segment produces the output:
    6
    3

98
  ```

Post/Pre-Decrement in Action

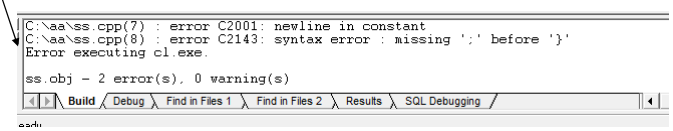
- Post-Decrement in Expressions:
- ```
  int    n = 2;
  int y;
  y = 2 * (n--);
  cout << y << endl;
  cout << n << endl;
  ```
  - This code segment produces the output:
    4
    1
- Pre-Decrement in Expressions:
  ```
  int    n = 2;
  int y;
  y = 2 * (--n);
  cout << y << endl;
  cout << n << endl;
  ```
  - This code segment produces the output:
    2
    1

99

How to find compilation errors

- In the window bellow the page of program editing.
- Go to the error name using right slide bare
- When you double click on the error, a notification arrow will point to the line that contains the error.
- Solve the first error first then the second and so on.

The window looks like this if there are some errors



100
  ```

Scope of the variable

- The area where a variable is declared and can be accessed is referred to as the scope of the variable.
- You can create and define your own scope.
- Each scope starts with left bracket { and end with right bracket }, this scope is called block.

101

example

what is wrong in the following program?

```
#include<iostream.h>
void main(){
int x=5;
    {int y=9;
     cout<<x<<endl;
    }
 cout<<y<<endl;
}
```

Outer scope

inner scope

Any variable declared inside the outer scope can be accessed from the inner scope. But that declared inside the inner cannot be accessed from the outer.

y is defined inside the inner scope not inside the outer scope , ERROR

102

Equality and Relational operators

•The result of a comparison is either true or false, where 0 is false and any value unequal to 0 is true

Example:

```
int x=44;
int y=12;
(x == y)   // false
(x >= y)   // true
(x != y)   // true
```

Relational operators	
x is less than y	x<y
x is less than or equal y	x<=y
x is greater than y	x>y
x is greater than or equal y	x>=y
Equality operators	
x is equal to y	x==y
x is not equal to y	x!=y

103

Conditional operator

- Three arguments: condition, value if true, value if false.

Example1:

```
cout<<(grade>=60?”pass”:”fail”);
```

Example2:

```
x=(z>0?z:y);
```

Only one statement can be exist in each argument

Example3:

```
(x<5?cout<<“hello\n”:cout<<“hi\n”);
```

104

Logical Operators

!→ (not)

Ex: a != b is true if a and b are not equal

&&→ (and)

Ex: 5<6 && 7>4 is true, but

5>6 && 7>4 is not true (i.e., false)

|| →(or)

Ex: 5>6 || 7>4 is true

5<6 || 7<4 is also true

105

Logical Operators

- Truth Tables

p	q	!p	p && q	p || q
True	True	False	True	True
True	False	False	False	True
False	True	True	False	True
False	False	True	False	False

106

Logical operations

- && Logical AND

True if both conditions are true

```
if((x<60)&&(x<35))  cout<<“35”<<endl;
```

- || logical OR

True if either conditions are true

```
if((x>100)||(x<0)) cout<<“x is not a grade\n”;
```

107

Logical operations

- ! Logical NOT.

  Returns true when its condition is false and vice versa.

```
if(x!=y)cout<<“x,y are not equal\n”;
```

This statement is equivalent to

```
if(!(x==y))  cout<<“x,y are not equal\n”;
```

108

Logical operations

- && operations evaluated first

  Example: what is the output of the following code?

```
bool x=true, y=false;
cout<<(x||x&&y)<<endl;
cout<<(x&&y||0)<<endl;
```

1

0

&& operations evaluated before || operations.

Any number exists in any logical expression is treated as true, except zero is treated as false.

109

Example

- What is the output of the following code?

```
int x=5;

if(x=6)     cout<<++x<<endl;

if(x++==7)     cout<<x<<endl;
```

This condition is always true because it is an assignment operation

7

Test then increment

8

110

Precedence and Associatively of the operators

Operator

Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Arithmetic Operations

What is the value of x in this expression:

- float x = (int)(5/2.0 + (float)(3/2) + (float) 7/2)

  Value of x is : (int) (2.5 + 1.0 + 3.0) =6.0
- Int x = 3/2.0 + (float)(5/2)+(float)(7/2)

  Value of x is : 1.5 + 2.0 + 3.0 =6
- Int x= 3.0+(3/6)+(3.0/2)+(float)(4/8)

  Value of x is : 3.0 + 0 + 1.5 + 0= 4

112

Arithmetic Operations

- int x= 3.0+(3/6)+(3.0/2)+(float)(4.0/8)

  Value of x is : 3.0 + 0 + 1.5 + 0.5= 5
- float x= 3.0+(3/6)+(3.0/2)+(float)(4.0/8)

  Value of x is : 3.0 + 0 + 1.5 + 0.5= 5.0

3/6 = 0	int / int= int
3.0/6.0 = 0.5	float/float= float
3.0/6 = 0.5	float/int = float
3/6.0 = 0.5	int/float = float

113

Assignment Operators

Example of assignment operators:

```
int  a = 4, b = 2, c = 36 ;
 a += b ;              /* This adds b to a, a=??   */
[ Answer:  a = a + b, so a = 4 + 2  or  a = 6  ]
 c /= a + b ;        /* What is value of c now  ?? */
[ Answer:  c = c / (a + b), and  a = 6  now,
  so  c = 36 / (6 + 2),  so  c = 36 / 8  or  c = 4  ]
```

114

Explicit type conversion

- Programmer specifies conversion with cast operator ( )
- More typical use
- Forcing a Type Change

• Example:

```
int x=1, y=2;
double result1 = x/y;     //  0.0
double result2 = double(x)/y; //  0.5
double result3 = x/double(y); //  0.5
double result4 = double(x)/double(y);// 0.5
double result5 = double(x/y);        //  0.0
int result6 = int(result4*100);  // 50
```

115

Arithmetic Operations

• What is the value of these expressions:

2+3/4*6+2 = 4

Start from left to right → determine precedence

3/4=0

0*6=0

2+0+2=4

116

Arithmetic Operations

- 3.0+4/7-(double)4/(5%3) = 1.0

Start from left to right → determine precedence
4/7 = 0
3.0+0 = 3.0
3.0 - (double)4/(5%3)
5%3 = 2
3.0 – (double) 4/2
3.0 – 2.0 =1.0

117

Assignment Conversions

- Example :

```
int m, n;
double xx;
m = 7;
n = 2.5;   // 2.5 converted to 2 and assigned to n
xx = m/n;  //7/2=3 converted to 3.0 and assigned to xx
n = xx+m/2;
Start from left to right with higher precedence
  // m/2=3 : integer division
  // xx+m/2 : double addition because xx is double
  // convert result of m/2 to double (i.e. 3.0)
  // xx+m/2=6.0
  // convert result of xx+m/2 to int (i.e. 6)
  //     because n is int
```

- Example :

```
a = (b = ( c= (d = (e = 4))));
    ←
Start from right to left
```

118

Exponentiation and Square root Operations

Exponentiation is not written as x^2

-C/C++ does not have an exponentiation operator. You can use the math function pow (a, b) which raises a to the b power.

Example:

```
int a= 2, b=5;
cout<<pow(a,b);     //  result 2^5  = 32
```

Square Root is not written as √ x

```
double sq= sqrt(36);
cout<<sq<<"\n";
```

-You must put a #include <math.h> in your source code

119

Control Statements

120

control structures

- Three control structures
  - Sequence structure
    - Programs executed sequentially by default
  - Selection structures
    - `if`, `if…else`, `switch`
  - Repetition structures
    - `while`, `do…while`, `for`

121

`if` Selection Statement

- Selection statements
  - Pseudocode example
    - *If student's grade is greater than or equal to 60*
      Print "Passed"
      - If the condition is `true`
        - » The print statement executes then the program continues to next statement
      - If the condition is `false`
        - » The print statement ignored then the program continues

122

`if` Selection Statement

- Selection statements (Cont.)
  - Translation into C++

```
if ( grade >= 60 )
     cout << "Passed";
```

  - Any expression can be used as the condition

123

`if` Selection Statement

- Logical AND (&&) Operator
  - Consider the following `if` statement

```
if ( gender == 1 && age >= 65 )
   Females++;
```

  - Combined condition is `true`
    - If and only if both simple conditions are `true`
  - Combined condition is `false`
    - If either or both of the simple conditions are `false`

124

`if` Selection Statement

- Logical OR (| |) Operator
  - Consider the following `if` statement

```
if ( ( semesterAverage >= 90 ) || ( finalExam >= 90 )
      cout << “Student grade is A” << endl;
```

  - Combined condition is `true`
    - If either or both of the simple conditions are `true`
  - Combined condition is `false`
    - If both of the simple conditions are `false`

125

`if` Selection Statement

- Example

```
if ( payCode == 4 )
   cout << "You get a bonus!" << endl;
```

  - If `paycode` is 4, bonus is given

```
if ( payCode = 4 )
   cout << "You get a bonus!" << endl;
```

  - `paycode` is set to 4 (no matter what it was before)
  - Condition is `true` (since 4 is non-zero)
    - Bonus given in every case

126

`if…else` Double-Selection Statement

- `if`
  - Performs action if condition true
- `if…else`
  - Performs one action if condition is true, a different action if it is false
- Pseudocode
  - *If student's grade is greater than or equal to 60*
    print "Passed"
    Else
    print "Failed"
- C++ code

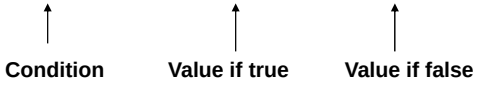
```
if ( grade >= 60 )
   cout << "Passed";
else
   cout << "Failed";
```

127

`if…else` Double-Selection Statement (Cont.)

- Ternary conditional operator (`?:`)
  - Three arguments (condition, value if `true`, value if `false`)
- Code could be written:

```
cout <<( grade >= 60 ? "Passed" : "Failed" );
```



128

`if…else` Double-Selection Statement (Cont.)

- Nested `if…else` statements
  - One inside another
  - Once a condition met, other statements are skipped
  - Example
    - *If student's grade is greater than or equal to 90*
      Print "A"
      Else
      If student's grade is greater than or equal to 80
      Print "B"
      Else
      If student's grade is greater than or equal to 70
      Print "C"
      Else
      If student's grade is greater than or equal to 60
      Print "D"
      Else
      Print "F"

129

`if…else` statements (Cont.)

- Nested `if…else` statements (Cont.)
  - Written In C++

```
if ( studentGrade >= 90 )
   cout << "A";
else
   if (studentGrade >= 80 )
      cout << "B";
   else
      if (studentGrade >= 70 )
         cout << "C";
      else
         if ( studentGrade >= 60 )
            cout << "D";
         else
            cout << "F";
```

130

if…else statements (Cont.)

- Nested if…else statements (Cont.)
  - Written In C++

```
if ( studentGrade >= 90 )
   cout << "A";
else if (studentGrade >= 80 )
   cout << "B";
else if (studentGrade >= 70 )
   cout << "C";
else if ( studentGrade >= 60 )
   cout << "D";
else
   cout << "F";
```

131

if…else statements (Cont.)

- else problem
  - Compiler associates else with the immediately preceding if
  - Example

```
if ( x > 5 )
   if ( y > 5 )
      cout << "x and y are > 5";
else
   cout << "x is <= 5";
```

  - Compiler interprets as

```
if ( x > 5 )
   if ( y > 5 )
      cout << "x and y are > 5";
   else
      cout << "x is <= 5";
```

132

`if…else` statements (Cont.)

- `else` problem (Cont.)
  - Rewrite with braces (`{}`)
    -
```
if ( x > 5 )
{
   if ( y > 5 )
      cout << "x and y are > 5";
}
else
   cout << "x is <= 5";
```
      - Braces indicate that the second `if` statement is in the body of the first and the `else` is associated with the first `if` statement

133

`if…else` statements (Cont.)

- Compound statement
  - Also called a block
    - Set of statements within a pair of braces
    - Used to include multiple statements in an `if` body
  - Example
    -
```
if ( studentGrade >= 60 )
   cout << "Passed.\n";
else
{
   cout << "Failed.\n";
   cout << "You must take this course again.\n";
}
```
  - Without braces,
```
cout << "You must take this course again.\n";
```
    always executes

134

if Statement

```
if (expression)
{
   statements;
 if (expression)
   {
      statements;
    }
}
```

Example:

```
if (u > v)
{
   a = 1;
   b = 2;
    if ( u > z)
    {
         x =11;
         y = 12;
    }
}
```

135

Example:
write a c++ program in order to test wither an input number is positive or negative.

```
#include<iostream.h>
void main(){
int x;
cin>>x;
if(x<0){//if
        cout<<“negative”;}//if
else{//else
cout<<“positive”;}//else
}
```

These comments are preferred for brackets tracking. In order to avoid forgetting to close any block.

136

Example

What is the output of the following code?

```
int x=5,y=7;
If(x>0)cout<<“x is greater than zero\n”;

else if(y==7)cout<<“y=7\n”;
else cout<<“bye\n”;
```

Brackets are optional for blocks that contain only one statement

Else statement always belongs to the last if statement. And the block after else statement is executed only when the last if statement is tested and its condition is false.

Output is:

x is greater than zero

137

Example

What is wrong in the following code?

```
int x=10;
if(x==10)cout<<“x=10\n”;
cout<<“hi\n”;
```

You can't put any statement between the if block and the else statement.

```
else cout<<“x is not equal to 10\n”;
```

138

The *switch* Statement

- Similar to if statements
- Can list any number of branches
- Used in place of nested if statements
- Avoids confusion of deeply nested ifs

139

The *switch* Statement

- `switch` statement
  - Controlling expression
    - Expression in parentheses after keyword `switch`
  - `case` labels
    - Compared with the controlling expression
    - Statements following the matching `case` label are executed
      - Braces are not necessary around multiple statements in a `case` label
      - A `break` statements causes execution to proceed with the first statement after the `switch`
        - » Without a `break` statement, execution will fall through to the next `case` label

140

The *switch* Statement

- `switch` statement (Cont.)
  - `default` case
    - Executes if no matching case label is found
    - Is optional
      - If no match and no `default` case
        - » Control simply continues after the `switch`

The *switch* Statement

```
switch (expression) {
        case value1:
                statement1;
                break;
        case value2:
                statement2;
                break;

        case valuen:
                statementn;
                break;
        default:
                statement;
                }
```

Break is necessary to exit the switch block after the execution of any group of statements.

If there is no break statement, every statement after the value of the variable will be executed till a break statement is encountered or till the end of the switch block;

The *switch* Statement

```
switch (grade)
{
    case 'A':
        cout << "Grade is between 90 & 100";
        break;
    case 'B':
        cout << "Grade is between 80 & 89";
        break;
    case 'C':
        cout  << "Grade is between 70 & 79";
        break;
        case 'D':
        cout  << "Grade is between 60 & 69";
        break;
    case 'E':
        cout  << "Grade is between 0 & 59";
        break;
    default:
        cout << "You entered an invalid grade.";
}
```

143

The *switch* Statement

* * * * Menu * * * *

1. Nablus
2. Rammallah
3. Tolkarm
4. Jenien

Choose either 1, 2, 3 or 4:

144

The *switch* Statement

```
switch (choice)
{
   case 1:
     cout << “Nablus”;
   case 2:
     cout << “Rammallah”;
   case 3:
     cout << “Tolkarm”;
   case 4:
      cout << “Jenien”;
default:
      cout<<“ invalid choice”;
}
```

```
switch (choice)
{
   case 1:
     cout << “Nablus”;
       break;
   case 2:
     cout <“Rammallah”;
       break;
   case 3:
     cout << “Tolkarm”;
     break;
   case 4:
      cout << “Jenien”;
     break;
  default:
      cout<<“ invalid choice”;
}
```

145

The *switch* Statement

```
#include<iostream>
Void main ( )
{
        int value
        cout << “Enter 1- Palestine 2- Egypt 3- USA”;
        cin >> value;
        switch (value)
        {
          case 1: cout << “No of population is 5 million”; break;
          case 2: cout << “No. of  population is 70 million”; break;
          case 3: cout << “No. of population is 180 million”; break;
          default: cout<<“invalid choice”;
       }
    }
```

146

Example

Using the switch structure, write a c++ program that reads a grade as character; A,B,C,D,E

And prints:

“excellent” for A

“very good” for B

“good” for C

“fair” for D

“fail” for other characters.

147

Example

```
#include<iostream.h>
void main(){
char grade;
cin>>grade;
switch(grade){//switch
case ‘A’:cout<<“excellent\n”;break;
case ‘B’:cout<<“very good\n”;break;
case ‘C’:cout<<“good\n”;break;
case ‘D’:cout<<“fail\n”;break;
default: cout<<“fail\n”;
} //switch
}
```

148

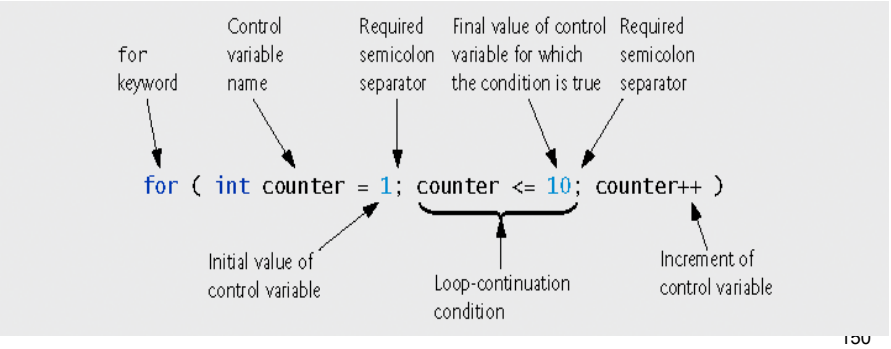
Loops

- for
- while
- do-while

149

`for` Repetition Statement

- `for` repetition statement
  - Specifies counter-controlled repetition details in a single line of code



150

for Repetition Statement

Not Valid: X

- for (j = 0, j < n, j = j + 3) // semicolons needed
- for (j = 0; j < n) // three parts needed

151

for Repetition Statement

Example 1:

```
j = 1;
sum = 0;
for ( ; j <= 3; j = j + 1){
      sum = sum + j;
     cout<<"sum = "<<sum<<"\n"; }
```

Output : 1, 3, 6

Example 2:

```
j = 1;
sum = 0;
for ( ; j <= 3; ){
     sum = sum + j;}
```

Output : infinite loop

Example 3:

```
j = 1;
sum = 0;
for ( ; ; )
{    sum = sum + j;  j++;
     cout << "\n" << sum;
}
```

Output : infinite loop

152

`for` Repetition Statement (Cont.)

- General form of the `for` statement
  - `for ( initialization; loopContinuationCondition; increment )`
    `statement;`
- Can usually be rewritten as:
  - *initialization*;
    `while ( loopContinuationCondition )`
    `{`
    statement;
    increment;
    `}`
- If the control variable is declared in the *initialization* expression
  - It will be unknown outside the `for` statement

153

`for` Repetition Statement (Cont.)

- `for` statement examples
  - Vary control variable from `1` to `100` in increments of `1`
    - `for ( int i = 1; i <= 100; i++ )`
  - Vary control variable from `100` to `1` in increments of `-1`
    - `for ( int i = 100; i >= 1; i-- )`
  - Vary control variable from `7` to `77` in steps of `7`
    - `for ( int i = 7; i <= 77; i += 7 )`
  - Vary control variable from `20` to `2` in steps of `-2`
    - `for ( int i = 20; i >= 2; i -= 2 )`
  - Vary control variable over the sequence: `2`, `5`, `8`, `11`, `14`, `17`, `20`
    - `for ( int i = 2; i <= 20; i += 3 )`
  - Vary control variable over the sequence: `99`, `88`, `77`, `66`, `55`, `44`, `33`, `22`, `11`, `0`
    - `for ( int i = 99; i >= 0; i -= 11 )`

154

`for` Repetition Statement (Cont.)

- Using a comma-separated list of expressions

```
for ( int number = 2;number <= 20;total += number, number += 2 ) ;

This is equal to :
for ( int number = 2;number <= 20;number += 2 ){
      total += number
}
```

155

Example: Write a c++ program that reads N input grades, and determines the average.

```
#include<iostream.h>
void main(){
float avg,x, sum=0;
int n;
cin>>n;
for(int i=1;i<=n;i++){//for
cin>>x;
sum+=x;
}//for
avg=sum/n;
cout<<“average=\t”<<avg<<endl;
}
```

156

Example

What is the output of the following code?

```
for(int i=0;i++<10;i++){
 cout<<i<<"\t";
}
```

1 3 5 7 9

Because in each repetition, the counter is incremented two times:

1 Just after the condition has been tested.

2 At the end of the repetition

157

Example

What is the output of the following code?

```
for(int i=1;i<=3;i++){//for
switch(i){//switch
case 1:cout<<"hi\t";
case 2:cout<<"hello\t";
case 3:cout<<"bye\t";
                }//switch
cout<<endl;
}//for
```

hi hello bye

hello bye

bye

It seems like that

"bye" is printed when the value of i is 1 or 2 or 3.

"hello" is printed when the value of i is 1 or 2

"hi" is printed only when the value of i is 1

158

Nested for structure

When there are two or more for structures inside each others, this called nested for structure.

The word nested can be used for any structure, when there are many blocks inside each others. Ex: for, if, while,…etc.

159

Example

Write a c++ program in order to print the numbers 1..100 ten numbers per line.

Such that, the first line contains the numbers 1..10, the second contains 11..20 and so on.

160

Example

```
#include<iostream.h>
void main(){//main
for(int i=0;i<10;i++){//outer for
   for(int j=1;j<=10;j++){//inner for
        cout<<10*i+j<<"\t";
        }//inner for
cout<<endl;
}//outer for
} //main
```

For each repetition of the outer loop, the inner loop is repeated 10 times

So, the outer loop is for line transition, and the inner loop is for printing the numbers.

161

`while` Repetition Statement

- Repetition statement
  - Action repeated while some condition remains true
  - Pseudocode
    - *While there are more items on my shopping list*
      Purchase next item and cross it off my list
  - `while` loop repeats until condition becomes `false`
  - Example
    - ```
      int product = 3;

      while ( product <= 100 )
         product = 3 * product;
      ```

162
      ```

Example: Finds the total of 4 numbers using `while loop`

```
int count =1;
int total = 0;
while (count <=4)
  {
  cout << "\nEnter a number: ";
       cin >> num;
       total = total + num;
       cout "The total is now " << total << endl;
       count++;
}
```

163

`do...while` Repetition Statement

- `do...while` statement
  - Similar to `while` statement
  - Tests loop-continuation after performing body of loop
    - Loop body always executes at least once
- C++ Code:

```
do
  {
     cout<<x<<" ";
          x++;
    } while (x<10) ;
```

164

Example

Write a C++ program that evaluates the average for unknown number of input numbers. such that, as long as the input number is positive, the program takes another number. But if the input number is negative, the program will stop and it will exit.

how can you use the while structure?

How can you use the do-while structure?

165

Solution using the while structure

```
#include<iostream.h>
void main(){
float avg,sum=0,count=0,x=0;
while(x>=0){//while
cin>>x;
sum+=x;
count++;
} //while
avg= sum/count;
cout<<“the result=\t”<<avg<<endl;
}
```

For this example, we must initialize x by a temporary value that makes the while condition to be true.

Otherwise the statements inside the loop can not be executed.

166

Solution using the do-while structure

```
#include<iostream.h>
void main(){
float avg,sum=0,count=0,x;
do{
cin>>x;
sum+=x;
count++;
} while(x>=0);
avg= sum/count;
cout<<“the result=\t”<<avg<<endl;
}
```

Whatever the value of x, the statements inside the loop will be executed at least for one time.

167

`break` and `continue` Statements

- `break` statement
  - Causes immediate exit from the loop
  - Used in `while`, `for`, `do…while` or `switch` statements
- `continue` statement
  - Skips remaining statements in loop body and start from the beginning of loop
    - `Used in for` loop, `while` , `do…while` loops

168

```
void main ( )
{
    int k;
    for ( k= -5; k < 25; k= k+5)
    {
          cout << k;
          cout << " Good Morning" << endl;
    }
}
```

Output = - 5 Good Morning
0 Good Morning
5 Good Morning
10 Good Morning
15 Good Morning
20 Good Morning

169

break

```
void main ( )
{
    int k;
    for ( k= -5; k < 25; k= k+5)
    {
          cout << k;
          break;
          cout << " Good Morning" << endl;
    }
}
```

Output = -5

170

continue

```
Void main ( )
{
    int k;
    for ( k= -5; k < 25; k= k+5)
    {
        cout << k;
        conitnue;
        cout << " Good Morning" << endl;
    }
}
```

Output =
```
- 5
0
5
10
15
20
```

171

Examples

```
int j =50;
while (j < 80)
    {
        j += 10;
        if (j == 70)
            break;
        cout << "j = " << j<< '\n';
}
cout << "We are out of the loop.\n";
```

Output

```
j = 60
We are out of the loop.
```

172

Example

```
do
    {
        cout << “Enter your age: “;
        cin >> age;
    if (age <=0)
            cout << “Invalid age.\n”;
    else
        cout << "DO SOMETHING\n";
    } while (age <=0);
```

173

Example

```
do  {
  x  =  x  + 5;
  y  =  x  *  25;
  cout << y  << endl;
  if ( x == 100) done = true;
  } while (!done);
```

174

Hw # 3

- **Write a program that computes the value of e^x by using the formula:?**

- **$e^x = 1 + x/1! + x_2/2 + x^3/3! + x^4/4!+\ldots..$**

175

Hw # 3

Write a program that reads three nonzero integers and determines and prints if they could be the sides of a right triangle ?

176