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.

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
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

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
      ```
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
    - \( z = a + b \)

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.
Numbers and Number Systems

Introduction

Real World Data → Input device → Computer Data

Dear Mom: → Keyboard → 10110010...

Digital camera → 10110010...
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
  \[(0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)\]
  \[A = 10, \ B = 11\]
  \[\ldots F = 15\]

Converting with fraction, examples

- Binary to Decimal
  Ex: \((1101.1)_2\) \[1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} = (13.5)_{10}\]

- Octal to Decimal
  Ex: \((673)_8\) \[6*8^2 + 7*8^1 + 3*8^0 = (443)_{10}\]

- Hexadecimal to Decimal
  Ex: \((A9C)_{16}\) \[A*16^2 + 9*16^1 + C*16^0 =\]
  \[10*16^2 + 9*16^1 + 12*16^0 = (2788)_{10}\]
Converting, example

- Binary to Octal
  Ex: \((011011.101100)_2 \rightarrow (33.54)_8\)

- Binary to Hexadecimal
  Ex: \((1111011.1101000)_2 \rightarrow (FB.D8)_{16}\)

- Hexadecimal to Binary
  Ex. \((A3.B)_{16} \rightarrow (10100011.1011)_2\)

Converting, example

- Decimal to Binary
  EX: \((12.3)_{10} \rightarrow (1100.01001)_2\)

  \[
  \begin{align*}
  12 / 2 &= 6 \ (\text{Remainder 0 (right)}) \\
  6 / 2 &= 3 \ (\text{Remainder 0}) \\
  3 / 2 &= 1 \ (\text{Remainder 1}) \\
  1 / 2 &= 0 \ (\text{Remainder 1 (left)})
  \end{align*}
  \]

  \[
  \begin{align*}
  0.3 \times 2 &= 0.6 \ (\text{left}) \\
  0.6 \times 2 &= 1.2 \\
  0.2 \times 2 &= 0.4 \\
  0.4 \times 2 &= 0.8 \\
  0.8 \times 2 &= 1.6 \ (\text{right})
  \end{align*}
  \]

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

• One byte of data can be used to store integers up to.

\[
\begin{array}{ccccccc}
0 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[
= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 127_{10}
\]

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
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

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).
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

Character Code: ASCII and Unicode

• ASCII and Unicode are two computer ‘languages’ for naming letters
  – The ASCII name for ‘a’ is ‘61’
  – Unicode
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 \(2^7 = 128\)

ASCII Reference Table
Control, Numeric, Alphabetic, Punctuations Codes

<table>
<thead>
<tr>
<th>MSD</th>
<th>LSD</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>DLE</td>
<td>SP</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>&quot;</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>ETB</td>
<td>7</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>10</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>11</td>
<td>K</td>
<td>{</td>
<td>k</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>12</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>13</td>
<td>M</td>
<td>}</td>
<td>m</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>SO</td>
<td>RS</td>
<td>&gt;</td>
<td>14</td>
<td>N</td>
<td>^</td>
<td>n</td>
<td>^</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>15</td>
<td>O</td>
<td>_</td>
<td>o</td>
<td>DEL</td>
<td></td>
</tr>
</tbody>
</table>
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.

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.
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.

Other Programming Languages

• Java
• FORTRAN
• COBOL
• Pascal
• .NET platform
• Visual Basic .NET
• Visual C++
• C#
  – Based on C++ and Java
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 ends 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.

Seven Steps to write a good program.

1. Define the program objectives
   - What you want the program to do?

2. Design the program
   - What should the user interface be like? How should the program be organized?
   - Who will the target user be? How much time do you have to complete the program?

3. Write the code

4. Compile

5. Run the program

6. Test and debug the program
   - It runs but it may run incorrectly.

7. Maintain and modify the program
Program Design-step 2

• Find a suitable algorithm to solve the problem.

• Draw the flowchart of that algorithm.

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.
Flowchart

Instruction category
• Starting program

Geometric figure
• start

Input
• input

Output
• output

Program design...

Flowchart

Instruction category
• Process

Geometric figure
• Z = x + y

Conditional instruction

End program
• End

Condition?
No
Yes
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.
- 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.

Program design...

Flowchart example 1

1. Start
2. Declaration
3. Input x
4. Input Y
5. Z = X + Y
6. Output Z
7. End
Flowchart example 2:

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

```
Program design...

start

X

Input X

No

X<0?

Yes

Output "positive"

Output "negative"

END
```

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.

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.
Flowchart example 3

- From the equation $\text{Avg} = \frac{\text{Sum}}{N}$, we can decide that we need 3 locations to start with:
  - Avg
  - Sum
  - N

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.
Before you begin – cont.

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

```c
#include <stdio.h>

int main(void)
{
    printf("My First Program.\n");

    return 0;
}
```
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.
Program writing

• We will use Microsoft Visual Studio 6.0.

It can be used for:
• Program writing.
• Program compilation.

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
How to start writing a program?

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

Example #1

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

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

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>`
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;

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
Example #1

```cpp
#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.

Example #1

```cpp
#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.
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.

<table>
<thead>
<tr>
<th>C++ statement contains escape sequence</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cout&lt;&lt;“hello\n word”;</code></td>
<td>hello word</td>
</tr>
<tr>
<td><code>cout&lt;&lt;“hello\t word”;</code></td>
<td>Hello word</td>
</tr>
<tr>
<td><code>cout&lt;&lt;“hello\r word”;</code></td>
<td>word</td>
</tr>
<tr>
<td><code>cout&lt;&lt;“hello\a”;</code></td>
<td>hello You will hear a beep</td>
</tr>
<tr>
<td><code>cout&lt;&lt;“hello\word”;</code></td>
<td>hello\word</td>
</tr>
<tr>
<td><code>cout&lt;&lt;“hello\” word”;</code></td>
<td>hello “ word</td>
</tr>
<tr>
<td><code>cout&lt;&lt;“hello”&lt;&lt;endl&lt;&lt;“word”;</code></td>
<td>hello word</td>
</tr>
</tbody>
</table>
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

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

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;
Data Type: 2- character

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

CODE:
```c
char letter;
letter = 'C';
```

MEMORY:
```
letter 67
```
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

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;
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:

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

Data Type: 5-void

The void type has no values and no operations.
Sizes of Data types

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Size in Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>bool</td>
<td>1</td>
</tr>
</tbody>
</table>

Variable naming

Variable name must start with
- Letter
- Underscore
- $!

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
Valid and Invalid Identifiers/Variable

<table>
<thead>
<tr>
<th>Variable</th>
<th>VALID?</th>
<th>REASON IF INVALID</th>
</tr>
</thead>
<tbody>
<tr>
<td>totalSales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>total_Sales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>total.Sales</td>
<td>No</td>
<td>Cannot contain .</td>
</tr>
<tr>
<td>4thQtrSales</td>
<td>No</td>
<td>Cannot begin with digit</td>
</tr>
<tr>
<td>totalSale$</td>
<td>No</td>
<td>Cannot contain $</td>
</tr>
</tbody>
</table>

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.
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;
  ```

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……
  ```
Example #2

```cpp
#include <iostream.h>
void main(){
    int num1, num2, result;
    cin >> num1;  // Input the first number and put it in location num1
    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;
}
```

Example #3

What is wrong in this code (part of a program)?
```
int x = 15;
X = 10;  // X is not declared, X is not the same as x
Cout << x << endl;  // Cout is not the same as cout
```

So, C++ is case sensitive language
Keywords and Identifier

Some Keywords in C and C++:

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

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.
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

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;
```
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).

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 = 'l';
```
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;
  ```

**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;
```
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) \(\text{exp1 ? exp2 : exp3}\)

---

Binary Arithmetic Operators

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>OPERATION</th>
<th>EXAMPLE</th>
<th>VALUE OF ans</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>ans = 7 + 3;</td>
<td>10</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>ans = 7 - 3;</td>
<td>4</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>ans = 7 * 3;</td>
<td>21</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>ans = 3 / 7;</td>
<td>0</td>
</tr>
<tr>
<td>%</td>
<td>Modulus-return remainder</td>
<td>ans = 7 % 3;</td>
<td>1</td>
</tr>
</tbody>
</table>
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;
```

/ 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
  ```
% 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
  ```

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
Casting

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

\[(\text{type2})\text{variable of type1}\]

- It is mostly used when we don’t need an integer division to truncate the remainder.

Example: int x=5,y=6;

\[
\text{float z;}
\]

\[
z=(\text{float})y/x;
\]

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;

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
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

Example #4

Write a C++ program that reads a small letter character and prints the upper case of that character.
Example #4

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

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

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

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
Increment and decrement operators

- Increment operator is used to increment the variable by one in the form like:
  \[ x++ \text{ or } ++x \]
  
  This is equivalent to \(x+=1\) and \(x=x+1\)

- Decrement operator is used to decrement the variable by one in the form like:
  \[ x-- \text{ or } --x \]

Preincrement and postincrement

- Preincrement: variable changed before used in expression. \(++x\)

- Postincrement: variable changed after used in expression. \(x++\)
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

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
Post/Pre-Decrement in Action

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

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

How to find compilation errors

• In the window below 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.
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.

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;
}
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:

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

<table>
<thead>
<tr>
<th>Relational operators</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x is less than y</td>
<td>x&lt;y</td>
</tr>
<tr>
<td>x is less than or equal y</td>
<td>x&lt;=y</td>
</tr>
<tr>
<td>x is greater than y</td>
<td>x&gt;y</td>
</tr>
<tr>
<td>x is greater than or equal y</td>
<td>x&gt;=y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Equality operators</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x is equal to y</td>
<td>x==y</td>
</tr>
<tr>
<td>x is not equal to y</td>
<td>x!=y</td>
</tr>
</tbody>
</table>

Conditional operator

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

Example1:
```cpp```
cout<<(grade>=60?"pass":"fail");
```
Example2:
```cpp```
x=(z>0?z:y);
```
Example3:
```cpp```
(x<5?cout<<"hello\n":cout<<"hi\n");
```cpp
```
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

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 |
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";

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";
Logical operations

• && operations evaluated first

Example: what is the output of the following code?

bool x=true, y=false;
cout<<(x||x&&y)<<endl; 1
cout<<(x&&y||0)<<endl; 0

&& operations evaluated before || operations.
Any number exists in any logical expression is treated as true, except zero is treated as false.

Example

• What is the output of the following code?

int x=5;
if(x=6)    cout<<++x<<endl; 7
This condition is always true because it is an assignment operation
if(x++==7)    cout<<x<<endl; 8
Test then increment
Precedence and Associatively of the operators

Operator

<table>
<thead>
<tr>
<th>Operators</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>postfix</td>
<td>expr11 expr</td>
</tr>
<tr>
<td>unary</td>
<td>++expr --expr +expr -expr ~ !</td>
</tr>
<tr>
<td>multiplicative</td>
<td>* / %</td>
</tr>
<tr>
<td>additive</td>
<td>+ -</td>
</tr>
<tr>
<td>shift</td>
<td>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</td>
</tr>
<tr>
<td>relational</td>
<td>&lt; &gt; &lt;= &gt;= instanceof</td>
</tr>
<tr>
<td>equality</td>
<td>== !=</td>
</tr>
<tr>
<td>bitwise AND</td>
<td>&amp;</td>
</tr>
<tr>
<td>bitwise exclusive OR</td>
<td>^</td>
</tr>
<tr>
<td>bitwise inclusive OR</td>
<td></td>
</tr>
<tr>
<td>logical AND</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>logical OR</td>
<td></td>
</tr>
<tr>
<td>ternary</td>
<td>?:</td>
</tr>
<tr>
<td>assignment</td>
<td>+= -= *= /= %= &amp;= ^=</td>
</tr>
</tbody>
</table>

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
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

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 ]
```
Explicit type conversion

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

• Example:

```java
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
```

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
**Arithmetic Operations**

- \( 3.0 + 4/7 - (\text{double})4/(5 \mod 3) = 1.0 \)

Start from left to right → determine precedence

\[
4/7 = 0 \\
3.0 + 0 = 3.0 \\
3.0 - (\text{double})4/(5 \mod 3) \\
5 \mod 3 = 2 \\
3.0 - (\text{double})4/2 \\
3.0 - 2.0 = 1.0
\]

**Assignment Conversions**

- **Example:**
  ```c
  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:**
  ```c
  a = (b = (c = (d = (e = 4))));
  Start from right to left
  ```
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 \texttt{pow (a, b)} which raises \texttt{a} to the \texttt{b} power.
Example:
\begin{verbatim}
int a= 2, b=5;
cout<<pow(a,b); // result $2^5 = 32$
\end{verbatim}

Square Root is not written as $\sqrt{x}$
\begin{verbatim}
double sq= sqrt(36);
cout<<sq<"n";
\end{verbatim}

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

Control Statements
control structures

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

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
if Selection Statement

• Selection statements (Cont.)
  – Translation into C++
    ```cpp
    if ( grade >= 60 )
    cout << "Passed";
    ```
  – Any expression can be used as the condition

if Selection Statement

• Logical AND (&&) Operator
  – Consider the following if statement
    ```cpp
    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
i f  Selection Statement

• Logical OR ( | | ) Operator
  – Consider the following i f  statement
    
    ```
    if ( ( semesterAverage >= 90 ) || ( finalExam >= 90 )
      cout << "Student grade is A" << endl;
    ```
  – Combined condition is t r u e
    • If either or both of the simple conditions are t r u e
  – Combined condition is f a l s e
    • If both of the simple conditions are f a l s e

i f  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 t r u e (since 4 is non-zero)
    • Bonus given in every case
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
  
  ```cpp
  if ( grade >= 60 )
  cout << "Passed";
  else
  cout << "Failed";
  ```

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

- Ternary conditional operator (?:)
  - Three arguments (condition, value if true, value if false)

- Code could be written:
  ```cpp
  cout <<( grade >= 60 ? "Passed" : "Failed" );
  ```

  Condition  Value if true  Value if false
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”

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";
if...else statements (Cont.)

• Nested if...else statements (Cont.)
  – Written In C++

```cpp
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";
```

• else problem
  – Compiler associates else with the immediately preceding if
  – Example

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

– Compiler interprets as

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

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
if Statement

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

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

Example:
write a C++ program in order to test whether 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.
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";
```

Output is:
- x is greater than zero

Example

What is wrong in the following code?
```
int x=10;
if(x==10)cout<<"x=10\n";
cout<<"hi\n";
else cout<<"x is not equal to 10\n";
```

You can't put any statement between the if block and the else statement.
The \textit{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

\textbf{The \textit{switch} Statement}

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

```
switch (expression) {
  case value1:
    statement1;
    break;
  case value2:
    statement2;
    break;
  case value3:
    statement3;
    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.";
}

** **** Menu ****

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

Choose either 1, 2, 3 or 4:
The `switch` Statement

```cpp
#include<iostream>

int 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";
    }
}
```
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.

Example

```cpp
#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
}
```
Loops

- for
- while
- do-while

for Repetition Statement

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

```
for (int counter = 1; counter <= 10; counter++ )
```
for Repetition Statement

Not Valid: X

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

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
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

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 )
for Repetition Statement (Cont.)

• Using a comma-separated list of expressions

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

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

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

```plaintext
#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;
}
```
Example

What is the output of the following code?
```cpp
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

Example

What is the output of the following code?
```cpp
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
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.

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.
Example

```c++
#include<iostream.h>
void 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
  } //outer for
  cout<<endl;
} //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.

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
    ```c++
    int product = 3;
    while ( product <= 100 )
      product = 3 * product;
    ```
Example: Finds the total of 4 numbers using while loop

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

d o...w hile 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:
  ```cpp
do
  {
      cout<<x<<" ";
      x++;
  } while (x<10);
  ```
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?

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;
}
Solution using the do-while structure

```cpp
#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="<avg<<endl;
}
```

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

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
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

break

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

Output = -5
**continue**

Void main ()
{
    int k;
    for ( k= -5; k < 25; k= k+5)
    {
        cout << k;
        continue;
        cout << “ Good Morning” << endl;
    }
}

Output = -5
0
5
10
15
20

**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.
Example

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

Example

do 
{
    x = x + 5;
    y = x * 25;
    cout << y << endl;
    if ( x == 100) done = true;
} while (!done);
Hw # 3

• Write a program that computes the value of $e^x$ by using the formula:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots$$

Hw # 3

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