BIL105E
Introduction to Scientific and Engineering Computing (C)

Purpose of the Course

- Learning programming concepts and algorithms
- Learning the C language
- Learning computing techniques such as numerical analysis, sorting, statistics, etc.

Course Web Page

www.ninova.itu.edu.tr

- Lecture slides
- Homework assignments
- Example C programs
- Old exam questions
- Announcements

Grading

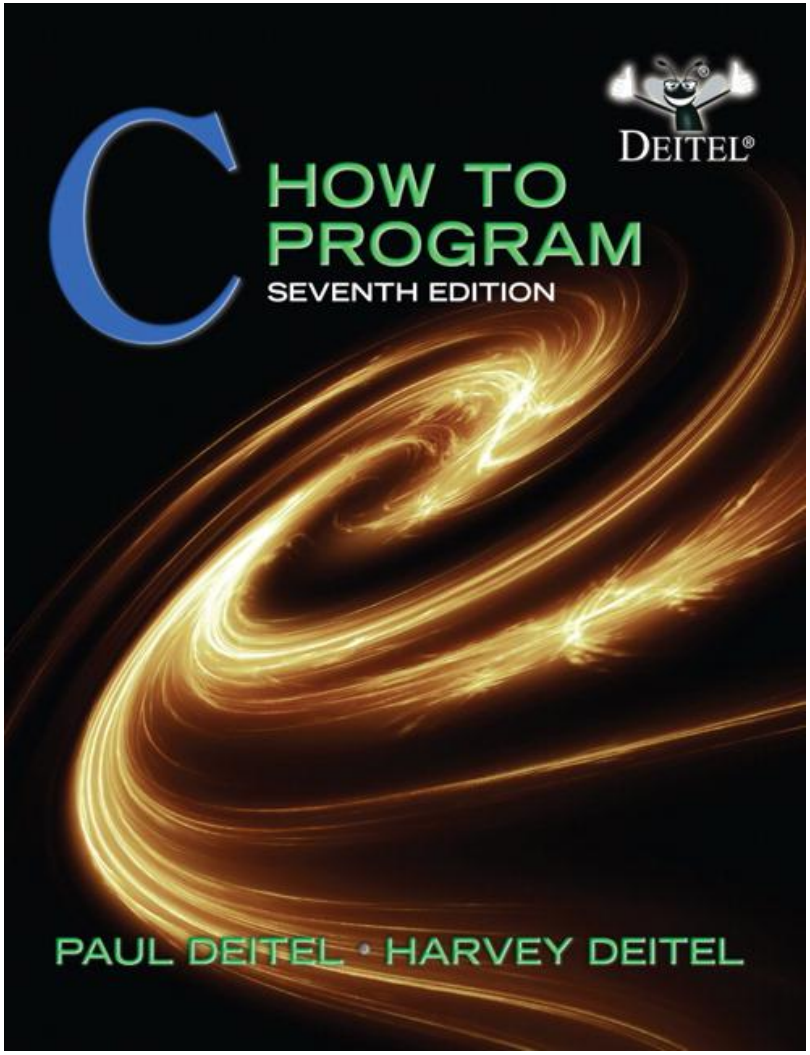
Homeworks	(4)	15%
Midterm exam	(1)	35%
Final exam		50%

- Homeworks must be submitted at Ninova only.
- Late submissions thru email are not accepted.

- Attendance (yoklama) (%70) is required both in lectures and in labs.

- Homeworks and exams are common in all BIL105E course sections.

Text Book



C How to Program,
(7th edition),
Harvey M. Deitel,
Paul J. Deitel,
Prentice Hall, 2012

Course Plan

DEITEL CHAPTER	TOPICS
–	Writing and Compiling a C Program; Algorithms / Flowcharts
1 , 2	Introduction to C Language
3	Structured Program Development in C
4	C Program Control
5	C Functions
6	C Arrays
7	C Pointers
8	C Characters and Strings
9	C Formatted Input/Output
10	C Structures, Typedefs, and Enumerations
11	C File Processing
12	Data Structures
13 , 14	Preprocessor; Other C topics

Compilers

- **Orwell Dev-C++ (version 5.4.1)**
  - Download from the internet

- **Microsoft Visual Studio 2012**
  - Download from Microsoft web site, or
  - Download from İTÜ repository :
    start->run-> \\yazilim.cc.itu.edu.tr

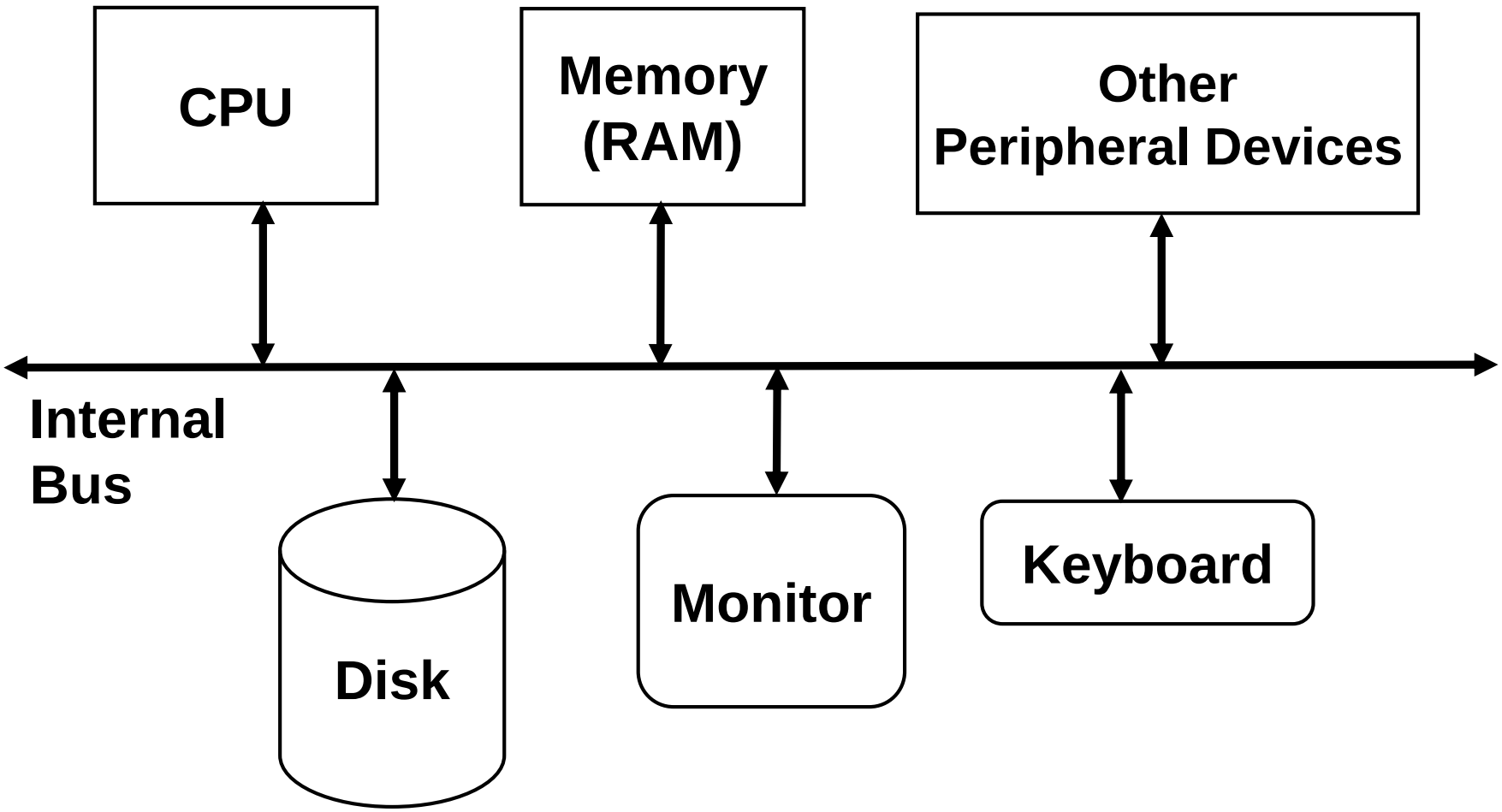
- **GNU C**
  - Available at the computer lab

Lab Sessions

- Lab sessions are given by course assistant.

- In labs, you will learn the followings:
  - How to edit, compile, and link a C program.
  - How to find and fix compiler errors.
  - How to test and debug.
  - Working on various exercices.

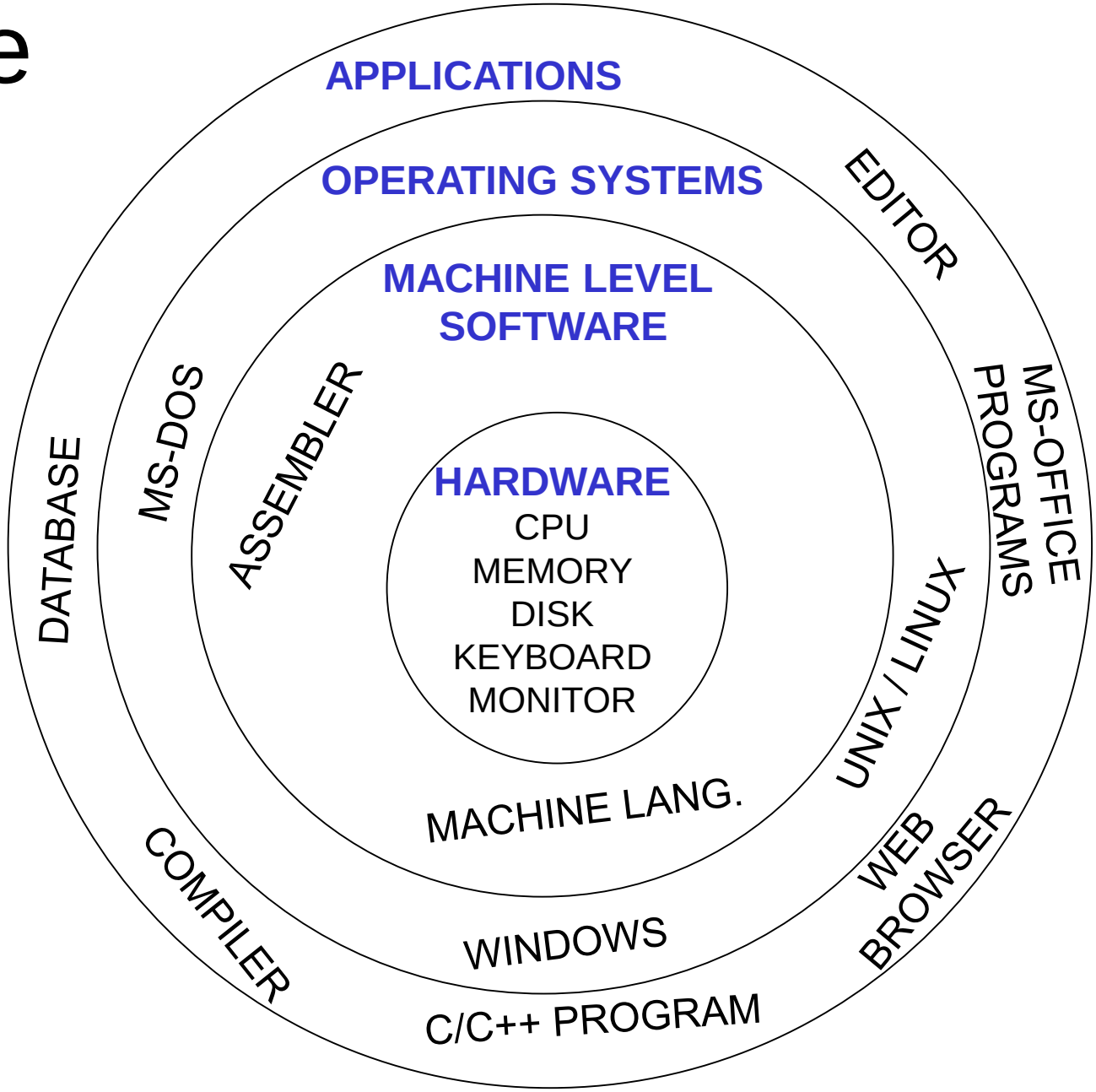
Computer Hardware



Computer Hardware

1. Input units
   - Obtains information from input devices (keyboard, mouse)
2. Output units
   - Outputs information (to screen, to printer, to control other devices)
3. Memory units (RAM, ROM)
   - Rapid access, low capacity, volatile, stores input information
4. Central processing unit (CPU)
   - Control Unit : Supervises and coordinates the other sections of the computer
   - Arithmetic and logic unit (ALU) : Performs arithmetic calculations and logic decisions
5. Secondary storage units (Hard disk, CD, DVD, Flush, etc)
   - Cheap, long-term (non-volatile), high-capacity storage
   - Stores data files, data bases, documents, and inactive programs

Software Layers



Machine Code

- Executable program files (binary) are in the machine code format which computer understands.
- Executable format differs between
  - Hardware
  - Operating systems

- Programmers do not write directly in machine code, instead they write source code in a *high-level language* like C, Java.

- Compiler tools are used to convert source code to machine code.

Comparison of Languages

High-level (C)

```
Total = 5 + 4;
```

Low-level (Intel Assembly)

```
LDA 5
STA Num1
LDA 4
ADD Num1
STA Total
END
```

Machine Code

```
1000111000
1000000111
0001110001
1000101010
0001110101
0010101010
```

- One instruction of high-level language generates many instructions of assembly and machine code.

A Simple C Program

```c
/* Take a number, multiply it by 10, and display result */
#include <stdio.h>
#include <stdlib.h>

int main()
{
int number, result;

printf ("Enter a number \n");
scanf ("%d", &number);
result = number *10;
printf ("The number multiplied by 10 equals %d \n", result);
system("pause");
}
```

Example screen output

```
Enter a number
23
The number multiplied by 10 equals 230

Press any key to continue ...
```

A Simple C Program

```
/* Take a number, multiply it by 10, and display result */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int number, result;
    printf (“Enter a number \n");
    scanf ("%d", &number);
    result = number *10;
    printf ("The number multiplied by 10 equals %d \n", result);

    system("pause");
}
```

Comments are written between /* and */

A Simple C Program

```
/* Take a number, multiply it by 10, and display result */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int number, result;
    printf (" Enter a number \n");
    scanf ("%d", &number);
    result = number *10;
    printf ("The number multiplied by 10 equals %d \n", result);

    system("pause");
}
```

The C pre-processor replaces this directives with the contents of the header files from the standard C library.

- printf() and scanf() are defined in <stdio.h>
- system() is defined in <stdlib.h>

A Simple C Program

```
/* Take a number, multiply it by 10, and display result */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int number, result;
    printf (" Enter a number \n");
    scanf ("%d", &number);
    result = number *10;
    printf ("The number multiplied by 10 equals %d \n", result);

    system("pause");
}
```

Every C program must have one **main** function.

Here ***int*** is type of value that returned to Operating System.

A Simple C Program

```
/* Take a number, multiply it by 10, and display result */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int number, result;
    printf (" Enter a number \n");
    scanf ("%d", &number);
    result = number *10;
    printf ("The number multiplied by 10 equals %d \n", result);

    system("pause");
}
```

Each variable must be explicitly defined as a specific data type.

A Simple C Program

```
/* Take a number, multiply it by 10, and display result */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int number, result;
    printf (" Enter a number \n");
    scanf ("%d", &number);
    result = number *10;
    printf ("The number multiplied by 10 equals %d \n", result);

    system("pause");
}
```

The <stdio.h> library defines the **printf()** function for displaying output.

A Simple C Program

```
/* Take a number, multiply it by 10, and display result */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int number, result;
    printf (" Enter a number \n");
    scanf ("%d", &number);
    result = number *10;
    printf ("The number multiplied by 10 equals %d \n", result);

    system("pause");
}
```

\n is the newline character

A Simple C Program

```
/* Take a number, multiply it by 10, and display result */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int number, result;
    printf (" Enter a number \n");
    scanf ("%d", &number);
    result = number *10;
    printf ("The number multiplied by 10 equals %d \n", result);

    system("pause");
}
```

The stdio library defines the **scanf()** function for capturing input.

A Simple C Program

```
/* Take a number, multiply it by 10, and display result */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int number, result;
    printf (" Enter a number \n");
    scanf ("%d", &number);
    result = number *10;
    printf ("The number multiplied by 10 equals %d \n", result);

    system("pause");
}
```

%d tells scanf() to interpret the input as a decimal value

A Simple C Program

```
/* Take a number, multiply it by 10, and display result */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int number, result;
    printf (" Enter a number \n");
    scanf ("%d", &number);
    result = number *10;
    printf ("The number multiplied by 10 equals %d \n", result);

    system("pause");
}
```

The **=** operator is used for assignment.

The ***** operator is used for multiplication.

A Simple C Program

```
/* Take a number, multiply it by 10, and display result */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int number, result;
    printf (" Enter a number \n");
    scanf ("%d", &number);
    result = number *10;
    printf ("The number multiplied by 10 equals %d \n", result);

    system("pause");
}
```

%d tells printf() to treat the value of the result variable as a decimal number.

A Simple C Program

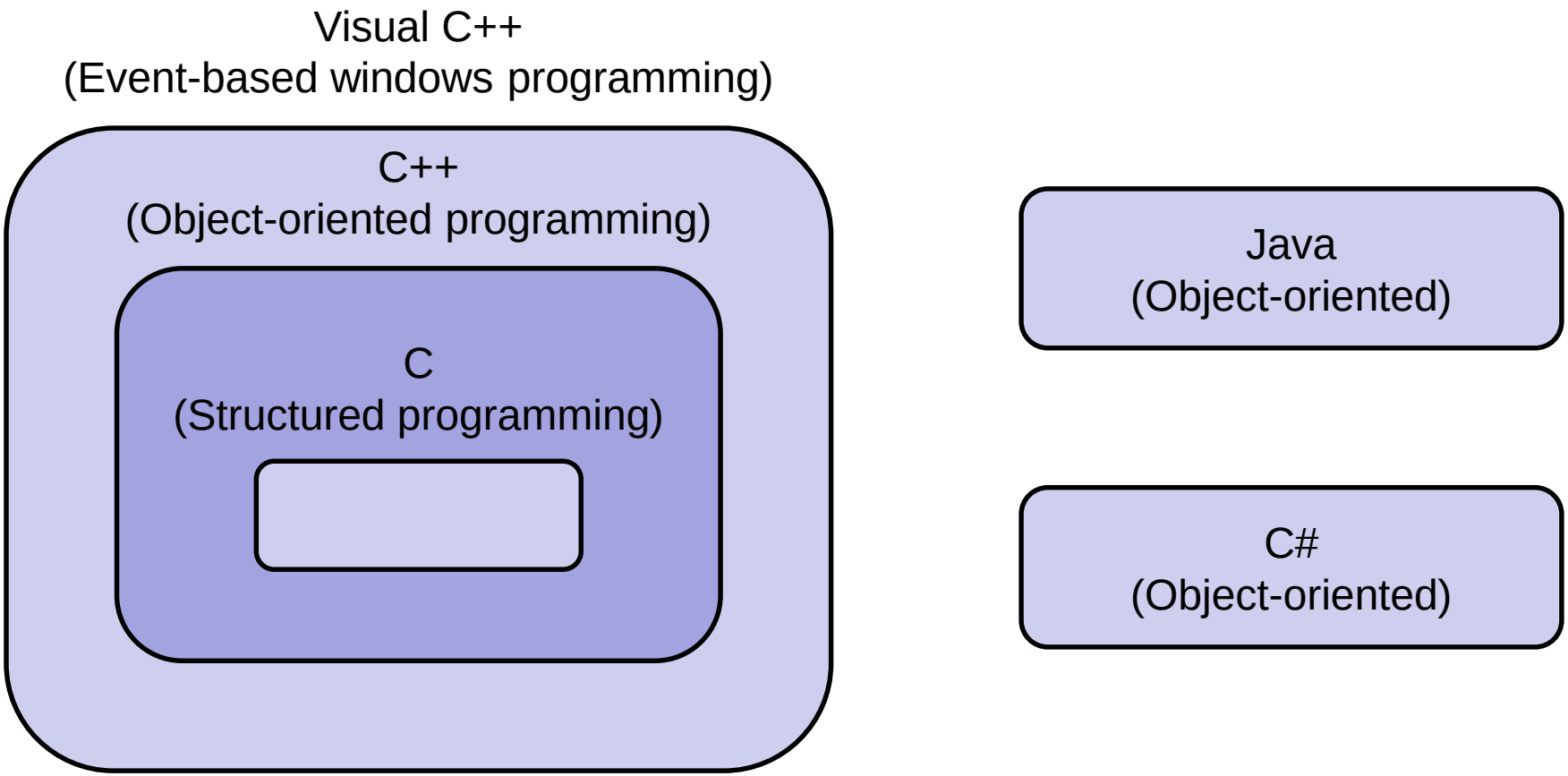
```
/* Take a number, multiply it by 10, and display result */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int number, result;
    printf (" Enter a number \n");
    scanf ("%d", &number);
    result = number *10;
    printf ("The number multiplied by 10 equals %d \n", result);

    system("pause");
}
```

- system("pause") causes the program to display a message and wait until user hits a key.
- It prevents the command line window from closing.

Source file suffixes

Suffix	Meaning
.c	C program file
.h	Header file for including
.cpp .cc	C++ program file

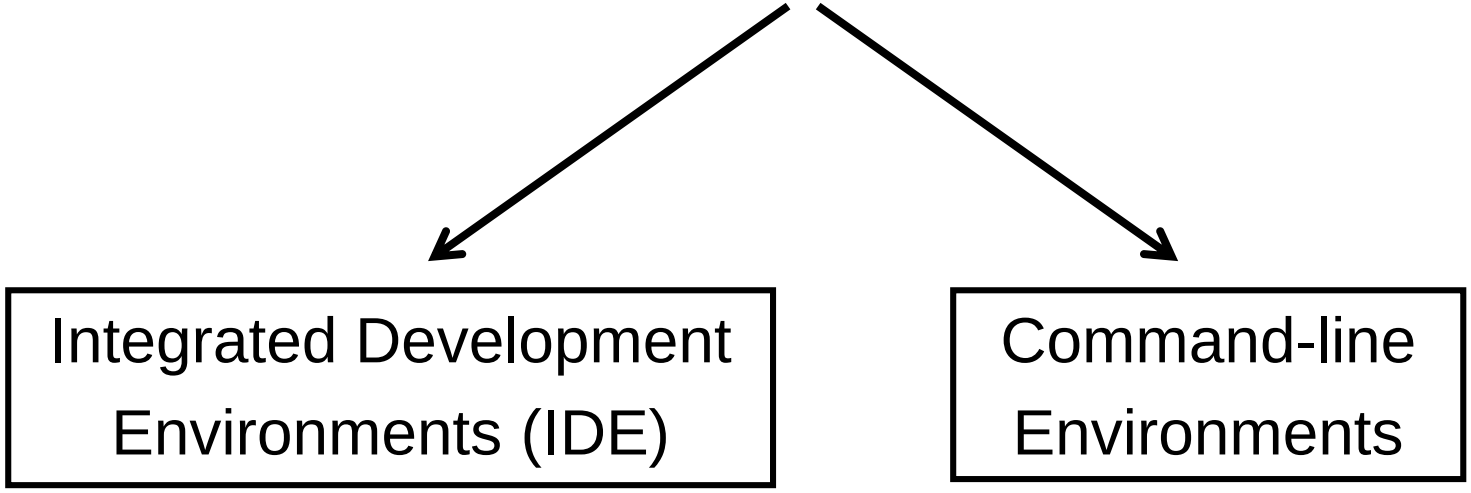
Programming langues with C-like syntax



Programming Environments

- All programming environments require some text editing capability, a C compiler and linker, and a way to run the executable code.

- There are two different types programming environments:



Integrated Development Environments (IDE)

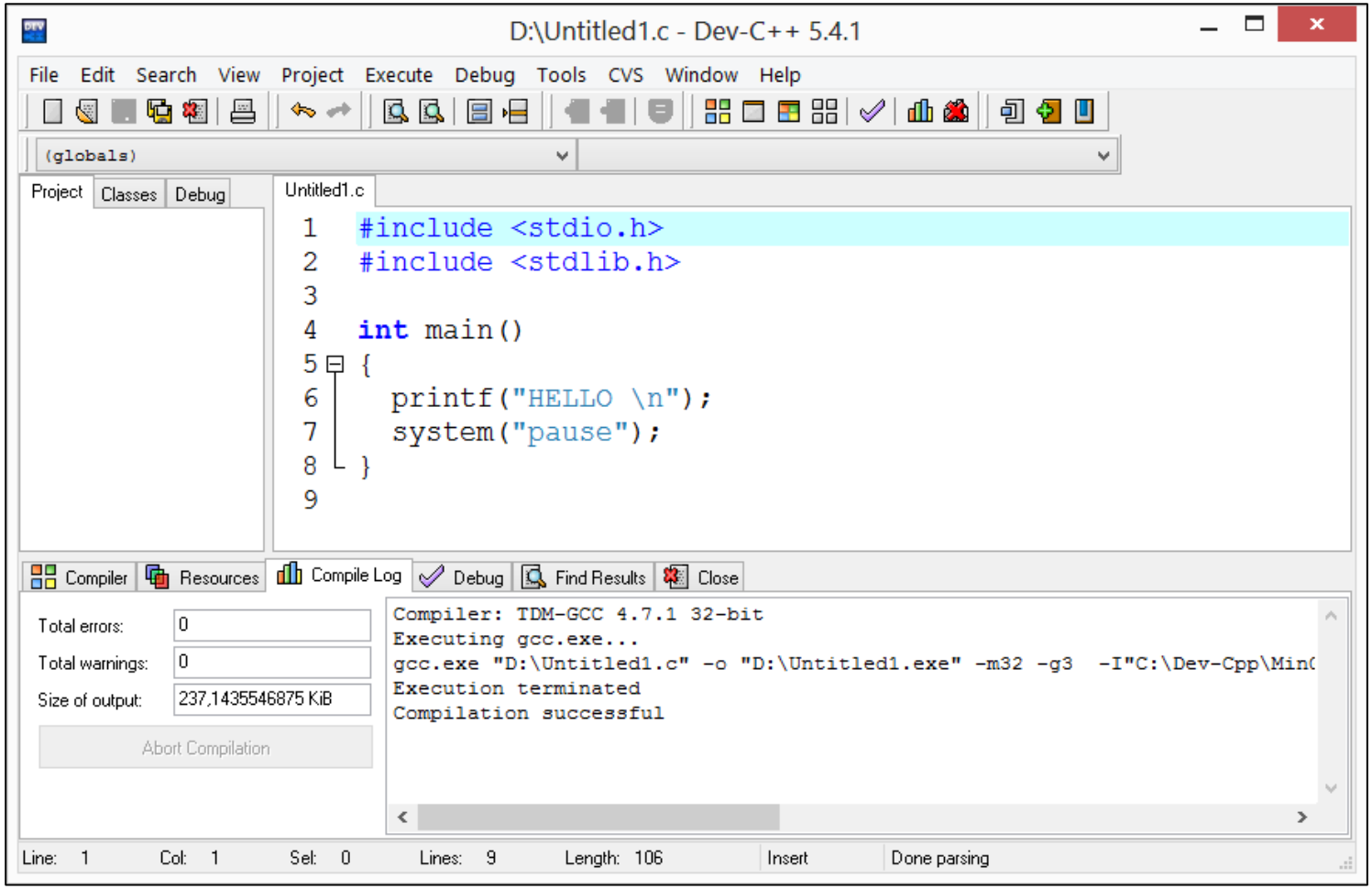
- An Integrated Development Environment is a graphical user interface program that integrates all components such as *text editor, compiler, linker, debugger, and project management.*
- Optionally, a visual interface builder may be included.

- Some of the IDE tools in Windows:

Lang. / IDE	C/C++	Visual C++	Visual Basic	C#.NET	PHP	Java
Dev-C++	√					
Microsoft Visual Studio	√	√	√	√		
NetBeans	√				√	√
Eclipse						√

IDE Example: Dev-C++

1) Choose from menu: File -> Save As... (or Ctrl+Alt+S keys)
   Enter "hello.c" as file name, then click Save button.
2) Choose from menu: Execute -> Compile & Run (or F11 key)



Command-line Environments

- A command-line environment is just a collection of commands that can be typed in to edit files, compile source code, and run programs.
- This is the simplest way to develop programs.

- Examples of command-line tools:

	Linux	Windows
Text Editor	vi	notepad, wordpad, etc.
Compiler	GNU C	Dev-C++ , Visual Studio

Command-line Example: Notepad and gcc

hello.c - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

```
// Örnek program
#include <stdio.h>

int main()
{
  printf("HELLO WORLD\n");
}
```

Yönetici: C:\Windows\system32\cmd.exe

```
C:\>gcc -o hello.exe hello.c

C:\>hello.exe
HELLO WORLD


C:\>_
```

Compiling and Executing a C Program (in command-line)

- First, write your program with a text editor like Notepad and save it under “Documents” folder. (Example: **hello.c**)

- Open a command-line window:
  - Click Windows **Start** button, then select **Run**
  - Type **“cmd”**, and hit Enter

- In the command-line window, type the followings:
  - “cd Documents”
  - **“gcc hello.c”**

- By default, an executable file named as **“a.exe”** is created unless specified otherwise. To execute your program, type ***a*** and hit Enter.

- In Unix, default executable file name is **“a.out”** which you can run it by typing **“./a.out “**

Other C compiler options

• If you are using the Orwell Dev-C++ (version 5.4.1) compiler, and your Windows Operating System is 32-bit, then you should use the **-m32** option, otherwise compiler will generate 64-bit executable by default.

gcc -m32 hello.c

• If you want to give a different name to the executable, use the ***-o*** flag.

gcc -m32 -o hello.exe hello.c

• If your program uses any mathematical functions such as sqrt(x), etc., then you should also use the ***-lm*** flag, so that the math library is linked.

gcc -m32 -lm -o myprog.exe myprog.c

Other C compiler options

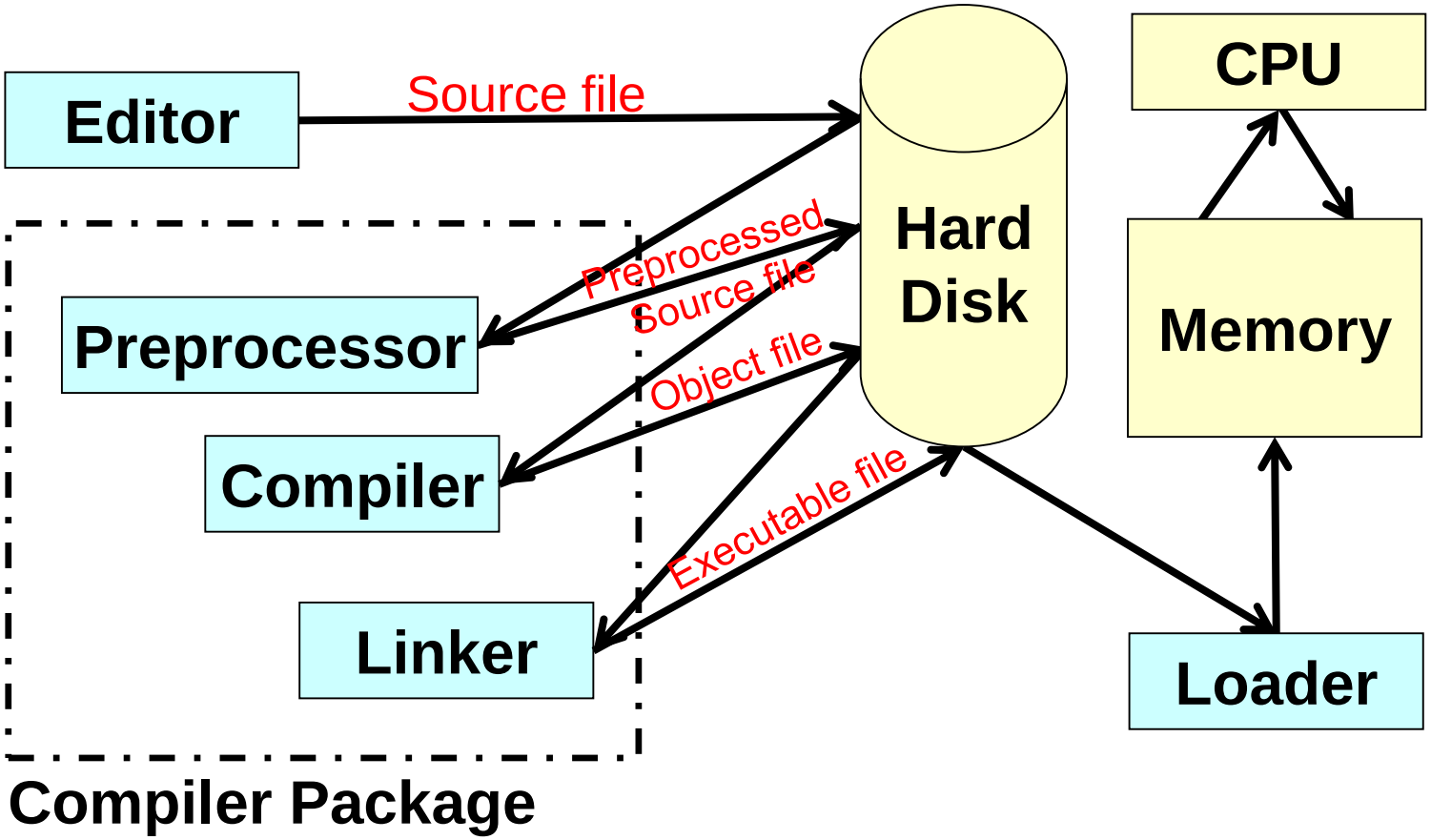
- If you have multiple source codes, then you can use the following example.

gcc -o myprog.exe part1.c part2.c

- If your source file is a C++ file (with cpp type), then you should use the following example.

g++ -m32 -o hello.exe hello.cpp

Role of Compiler Package and Loader



Role of Compiler Package and Loader

- Programmer writes the program in the editor and stores on disk.

- Preprocessing is to carry out predefined manipulations (defined by directives) on source code and to store on disk.

- Compiler converts preprocessed source code to object code and stores on disk.

- Linker links the object code with relevant libraries and creates executable and stores on disk.

- Loader loads the executable into memory.

- CPU reads instructions and data to execute.

- **Compiler:**
  - Compiler checks your source code finds all **syntax errors** such as mistyping
  - It translates the code all at once and produces “object code” as a temporary file

- **Linker:**
  - The Linker uses the “object code” to generate the “executable code”
  - Executable code is a binary file which means you can just double-click it and it runs
  - When a function is called, linker locates it in the C library and inserts it into object program
  - If function name is misspelled, the linker will produce an error because it will not be able to find function in the library

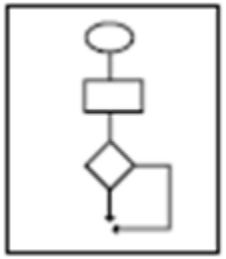
Program Development Phases

Programming Phases

Step 1

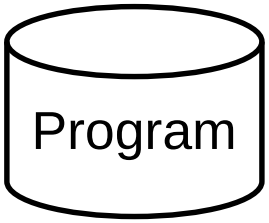
Input and Output Definition

Step 2



Logic Definition (Algorithm)

Step 3



Coding

Step 4

Testing and Debugging

Programming Phases

- **1. <u>Defining</u>** the problem

- **2. <u>Designing</u>** the program on “paper”:
  - Write the algorithm (pseudocode or flow chart)

- **3. <u>Coding</u>:** Writing the program source code in C language
  - Use an editor
  - Compile and link source code by a compiler
    - Compiler can detect syntax errors

- **4. <u>Testing (Running) and Debugging</u>:**
  - Test the program with input data and see whether the actual ouputs are the same as the expected outputs
  - Debugging is finding and fixing the logic errors in source code

Phase 1: Define the Problem

- In this phase, you write a specification of the problem.

- Specification defines the followings:
  - <u>input data</u>
  - <u>processings</u>
  - <u>output data</u>

- This phase requires cooperative work with the programmer and the problem owner (stakeholder).

- Other phases are only for the programmer.

Phase 2: Design the Program

- Program design is done from the specification.

- Identify the main components of a program and how they work together.
  - Identify the main goal of a program
  - Then, break it down into sub goals
  - Keep refining it until the program is designed

- **<u>Algorithms</u>** are written during design. The following choices can be used for algorithm design.

Flow Chart **Pseudo Code**

Phase 3: Coding the Program

- After designing the program, it is coded in a programming language such as C.
- Coding is also known as “implementation”.
- A compiler software must be used to compile the program source code and link the executable code.

Phase 4: Testing and Debugging the Program

- You should run your program and see how it works.

- Testing is the final phase before releasing the program
  - Various input data values should be used
  - Observed output values should be compared to the Expected output values

- In testing phase, you should find any **logic errors** such as wrong calculation, or using wrong input data.

- Errors are called “bugs” informally. The process of finding bugs is called “debugging”.

Example:
Calculate Grade of a Student

Phase 1: Define the Problem

- **PURPOSE:** A student's passing grade will be calculated and printed on the screen.

- **INPUTS:** Inputs are the numeric values of first midterm exam, second midterm exam, and final exam.

- **OUTPUT:** Output is the numeric value of Grade.

- **PROCESSING:** Grade should be calculated with the following weights:

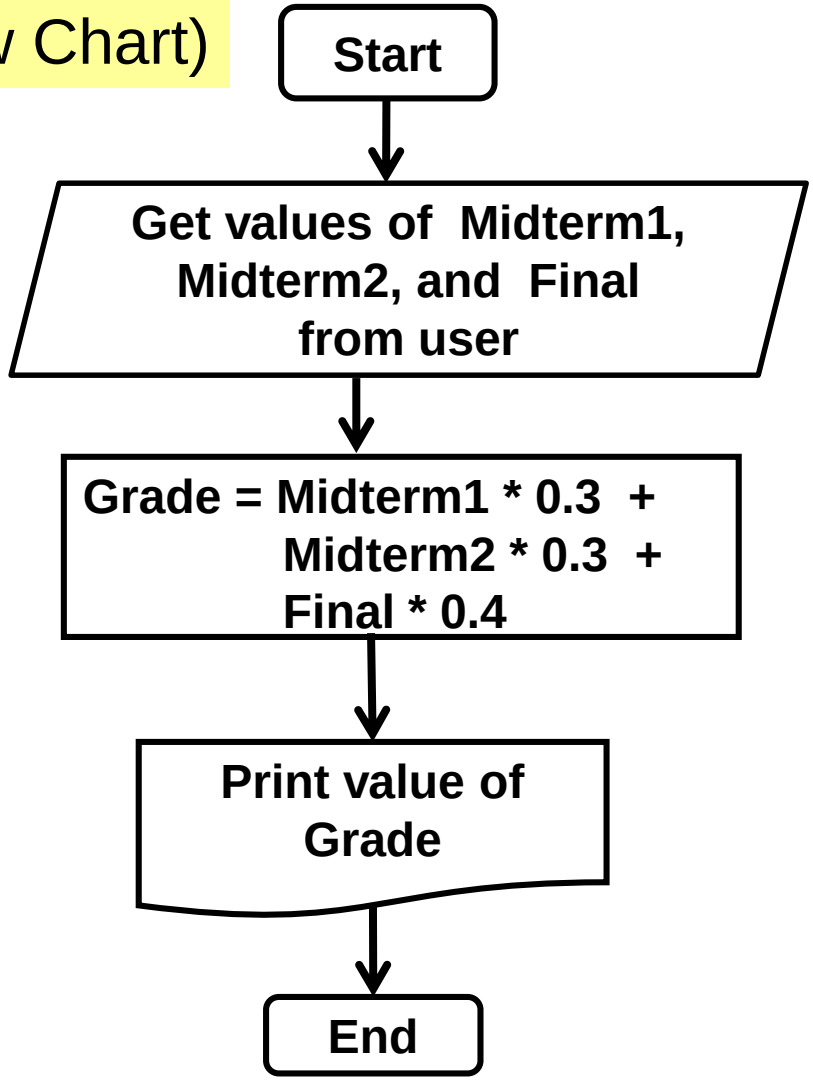
  30% of first midterm exam

  30% of second midterm exam

  40% of final exam

Phase 2: Design the Program

(Flow Chart)



(Pseudo Code)

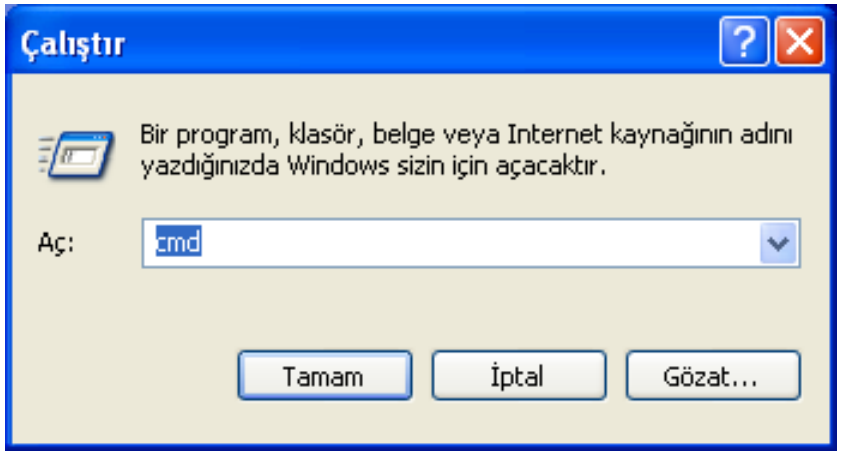
1. Get values of Midterm1, Midterm2, and Final from user
2. Grade ← Midterm1 * 0.3 + Midterm2 * 0.3 + Final * 0.4
3. Print value of Grade on screen
4. End

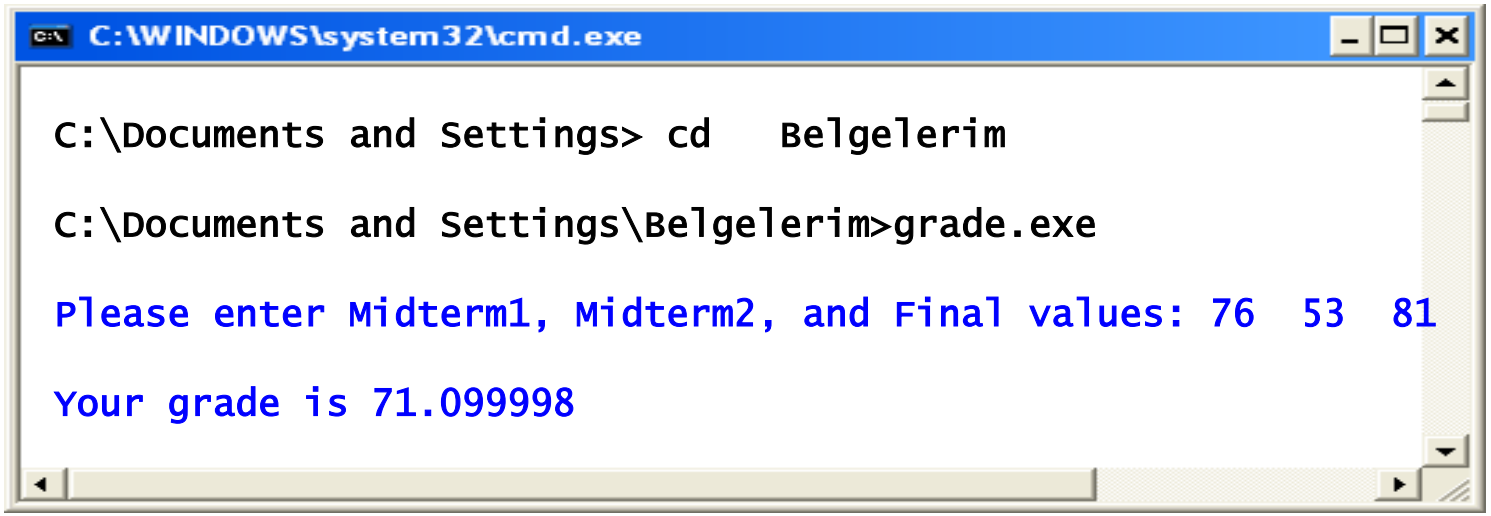
Phase 3: Coding and Compiling the Program

```
#include <stdio.h>

int main()
{
    int Midterm1, Midterm2, Final;
    float Grade;

    printf("Please enter Midterm1, Midterm2, and Final values: ");
    scanf("%d %d %d", &Midterm1,  &Midterm2,  &Final);

    Grade = (Midterm1 * 0.3) + (Midterm2 * 0.3) + (Final * 0.4);
    printf("Your grade is %f \n\n", Grade);
}
```

Phase 4: Running the Program (command-line)

- Click Windows **Start**, then select **Run**.
- Type **"cmd"** and hit Enter key.
- A Windows comand-line screen will appear as below.





ALGORITHMS

(FLOWCHART and PSEUDOCODE)

Algorithms

- An algorithm is a set of steps for carrying out a task, which is a step-by-step solution to the given problem.
- An algorithm is the design (modelling) of a program before coding with C language.

- The following choices can be used to design an algorithm:

- **Flow Chart**
  - Composed of standard shapes and symbols
  - Outlines the solution steps to the problem and flow of control

- **Pseudo Code**
  - An alternative method to flow chart
  - Constrained form of English (or Turkish) is used to outline the steps involved in a problem solution

Pseudo Code

- Pseudo code
  - Not a real programming language, so not prepared for compilation
  - Contains both some structural features of a programming language and a speaking language such as English
  - Easily understandable by non-programmers
  - Aims at defining a solution to a problem

- Rules for writing a pseudo code:
  - Use simple and short sentences (avoid compound sentences)
  - Each sentence is to indicate only one action

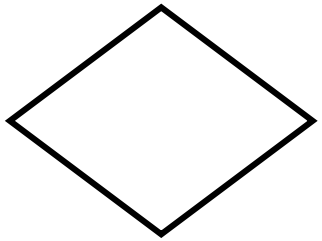
Flow Chart

- A flowchart is a diagrammatic representation that illustrates the sequence of operations to be performed to get the solution of a problem.
- Flowcharts are drawn in the first stages of program development.
- Flowcharts facilitate communication between programmers and also non-programmer people.
- Flowcharts play a vital role in the programming of a problem and are quite helpful in understanding the logic of complicated and lengthy problems.
- Once the flowchart is drawn, it becomes easy to write the program (i.e. coding) in any high level language such as C.

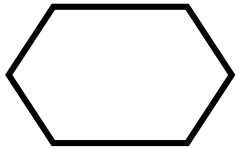
Standard Flow Chart Symbols

Start/End

Process
(Action)



Decision

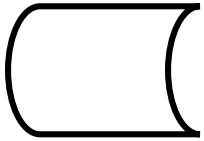


Loop

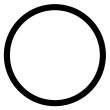
Function call
(Procedure,
Subroutine)

Output
(Monitor)

Input
(Keyboard)



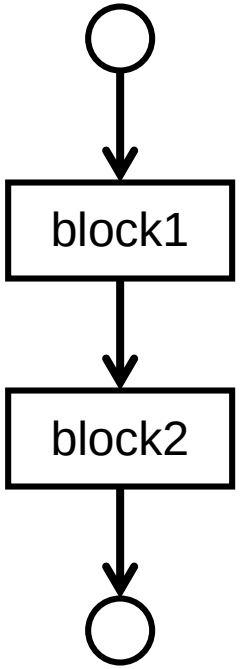
Data Storage
(Hard Disk)



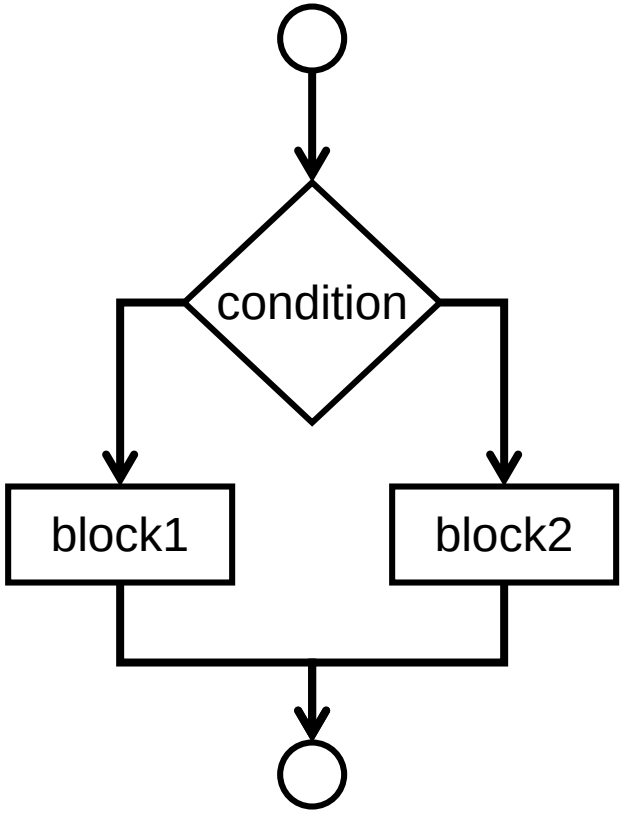
Page connection

Block Structures

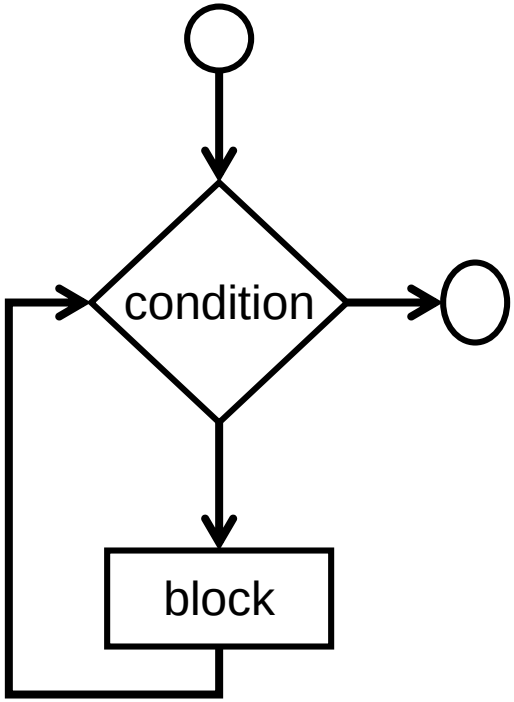
Sequence



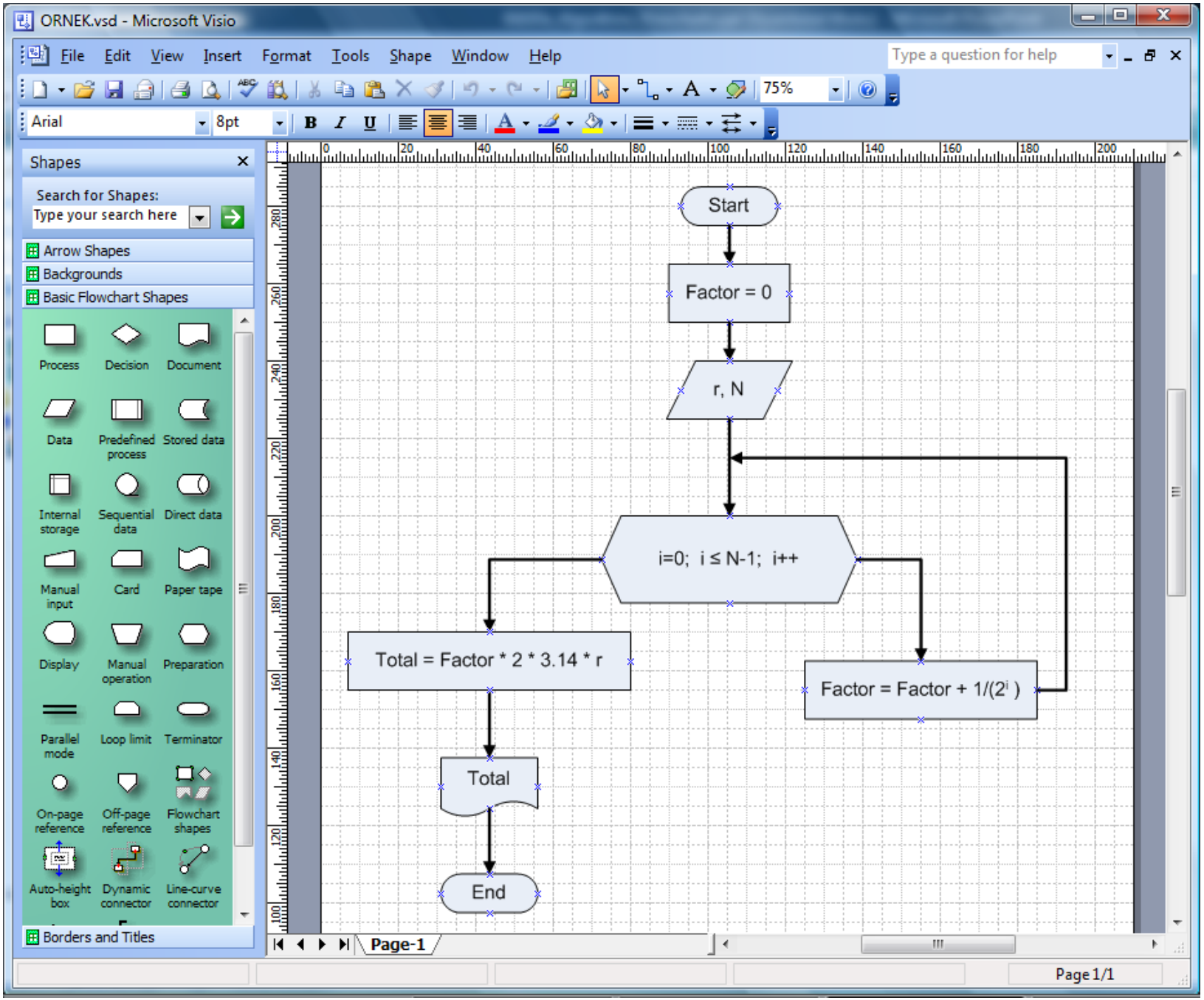
Selection



Repetition (loop)



Example Tool: Microsoft Visio



Example:
Calculating Factorial

Phase 1: Define the Problem

- **PURPOSE:** Write a program which calculates the factorial of a number.

- **INPUT:** An integer number N.

- **OUTPUT:** Factorial of the N.

- **PROCESSING:** Factorial is computed as the following.

  N ! = 1 * 2 * 3 * 4 * ….. * N

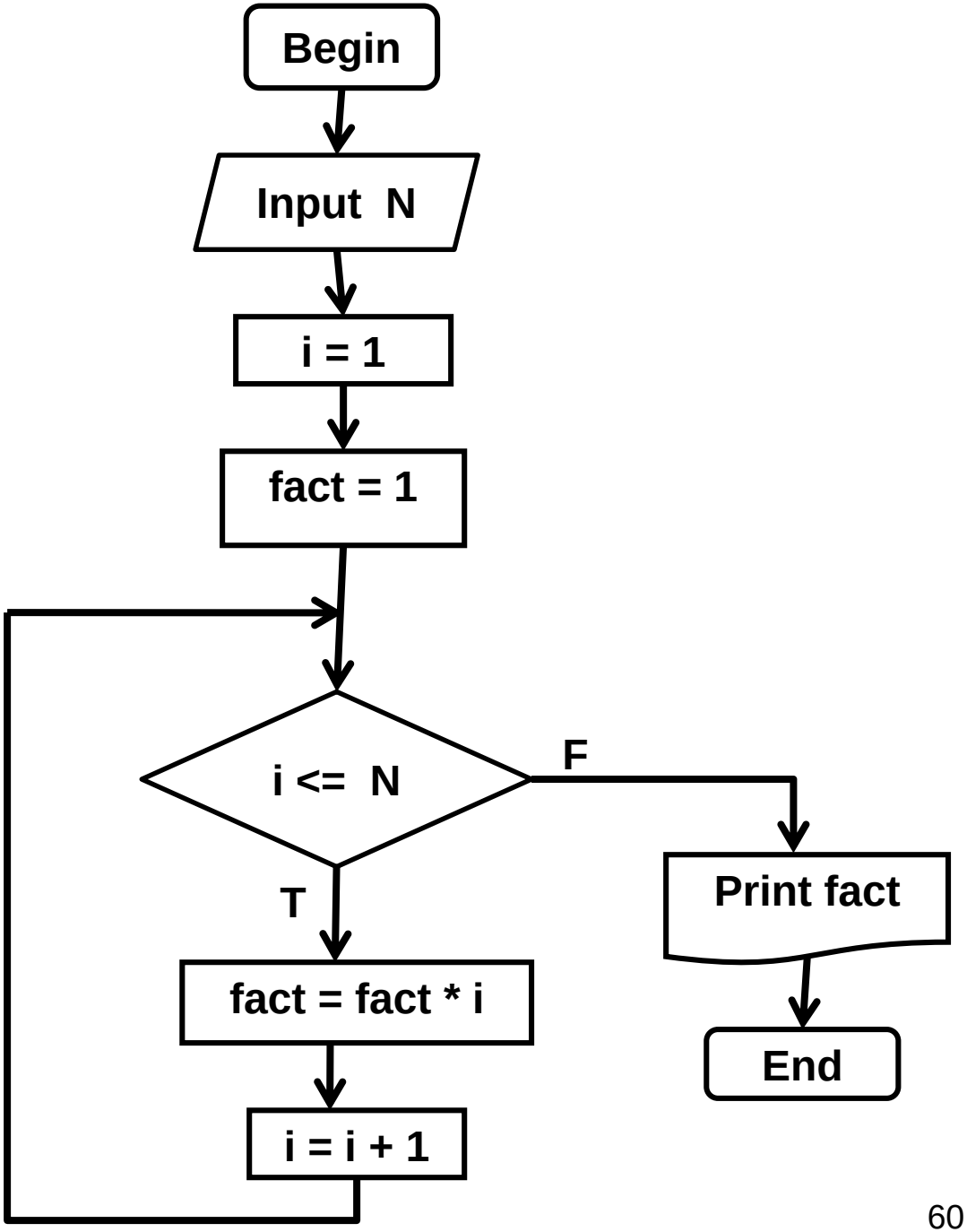
Phase 2: Design the Program

<u>Variables:</u>

N : Limiting number

i : Loop counter

fact : Factorial



Phase 3: Coding the Program

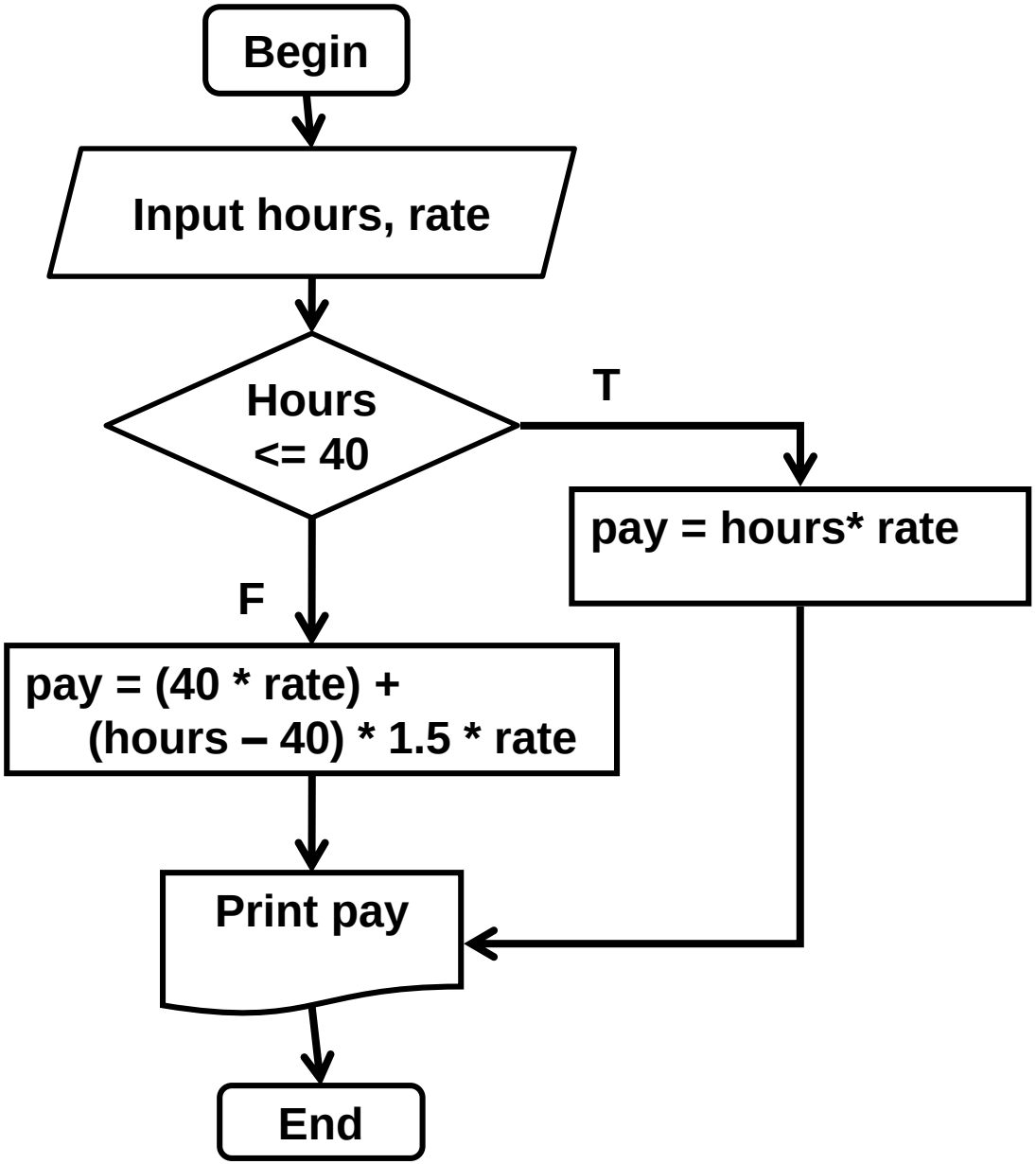
```
/* Example program to calculate factorial */
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int N, i;
  double fact;
  printf("Enter N value : ");
  scanf("%d", &N);
  i = 1;
  fact = 1;
  while (i <= N)   {
      fact = fact * i;
      i = i + 1;
  }
  printf("Factorial : %.lf \n\n", fact);
  system("PAUSE");
}
```

Phase 4: Compiling and Running the Program

```
C:\>gcc  -o  factorial.exe  factorial.c

C:\>factorial

Enter N value : 3
Factorial : 6
```

EXAMPLES of ALGORITHMS

Example1: Calculate Payment (Selection)




```
Begin
  input hours, rate
  if hours ≤ 40 then
     pay = hours * rate
  else
    pay = (40 * rate) +
     (hours – 40) * 1.5 * rate
    print pay
End
```

Example2: Computing Roots

- Find the roots of the second degree equation:

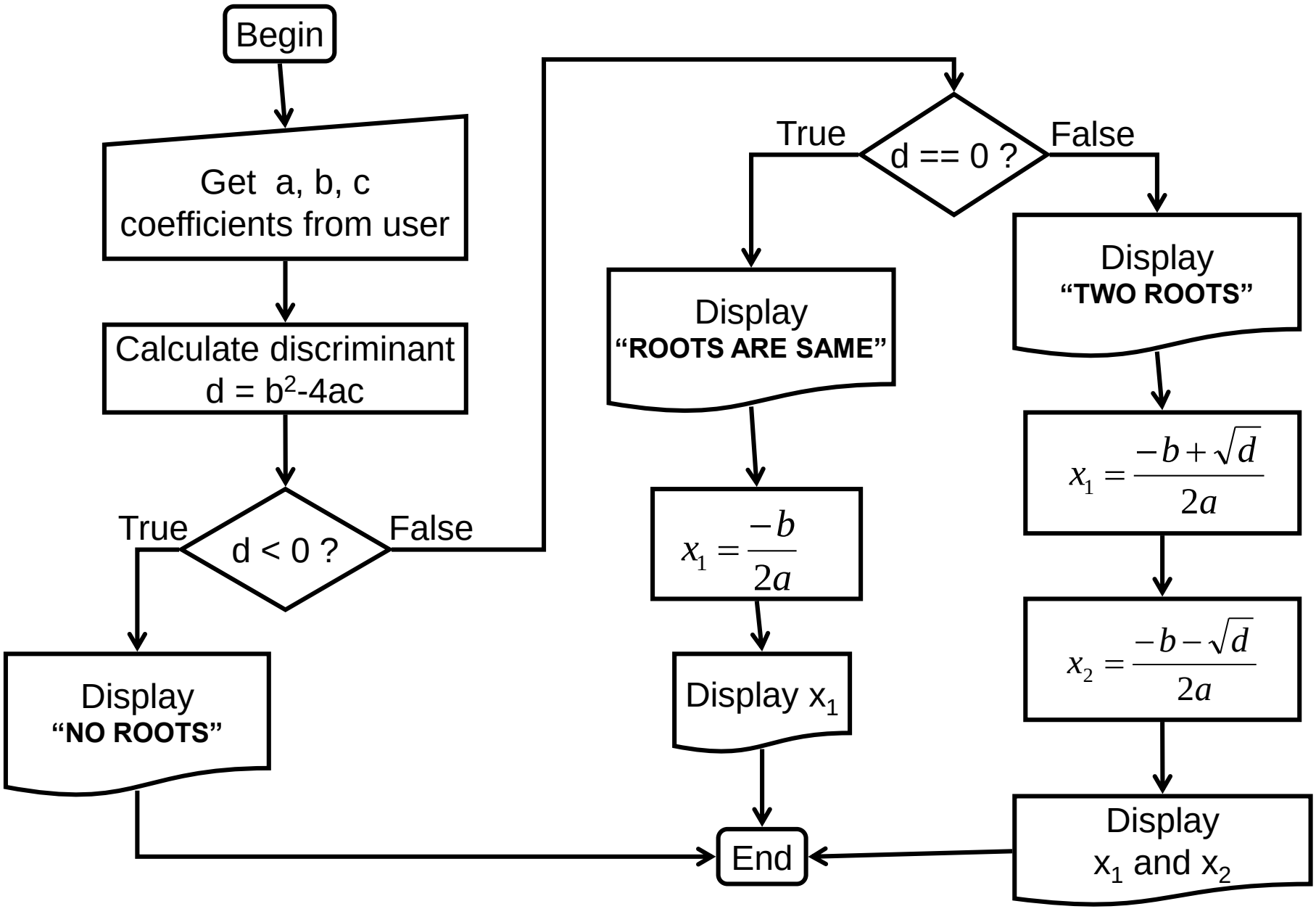
  $ax^2 + bx + c = 0$

$$X_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$X_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

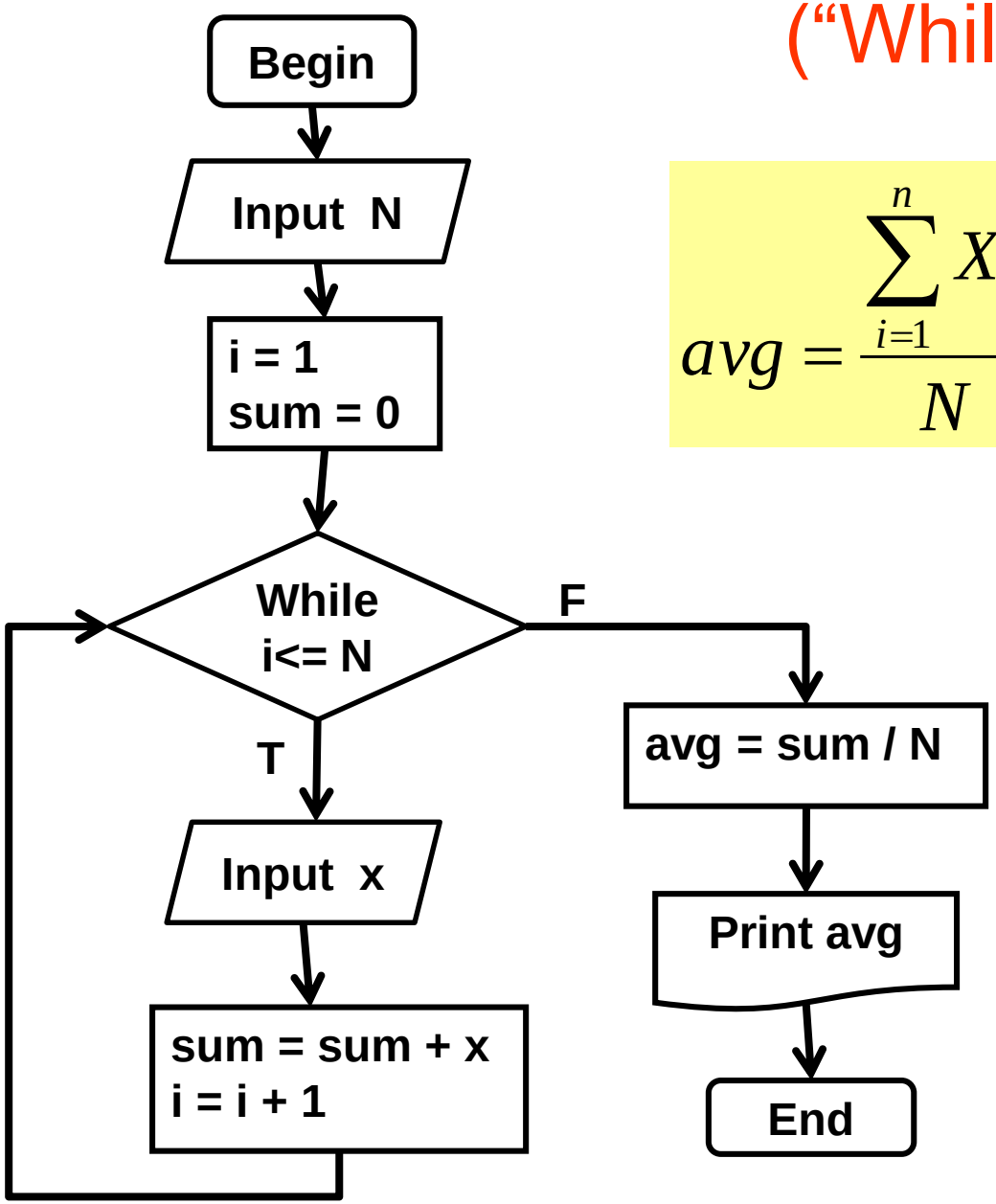
- Compute discriminant: **$d = b^2 - 4ac$**
  - if negative: “No real roots"
  - if zero: “Roots are same"
  - if positive: “Two real roots"

Example2: Computing Roots(Selection)



Example3: Average of N Numbers

("While" loop)

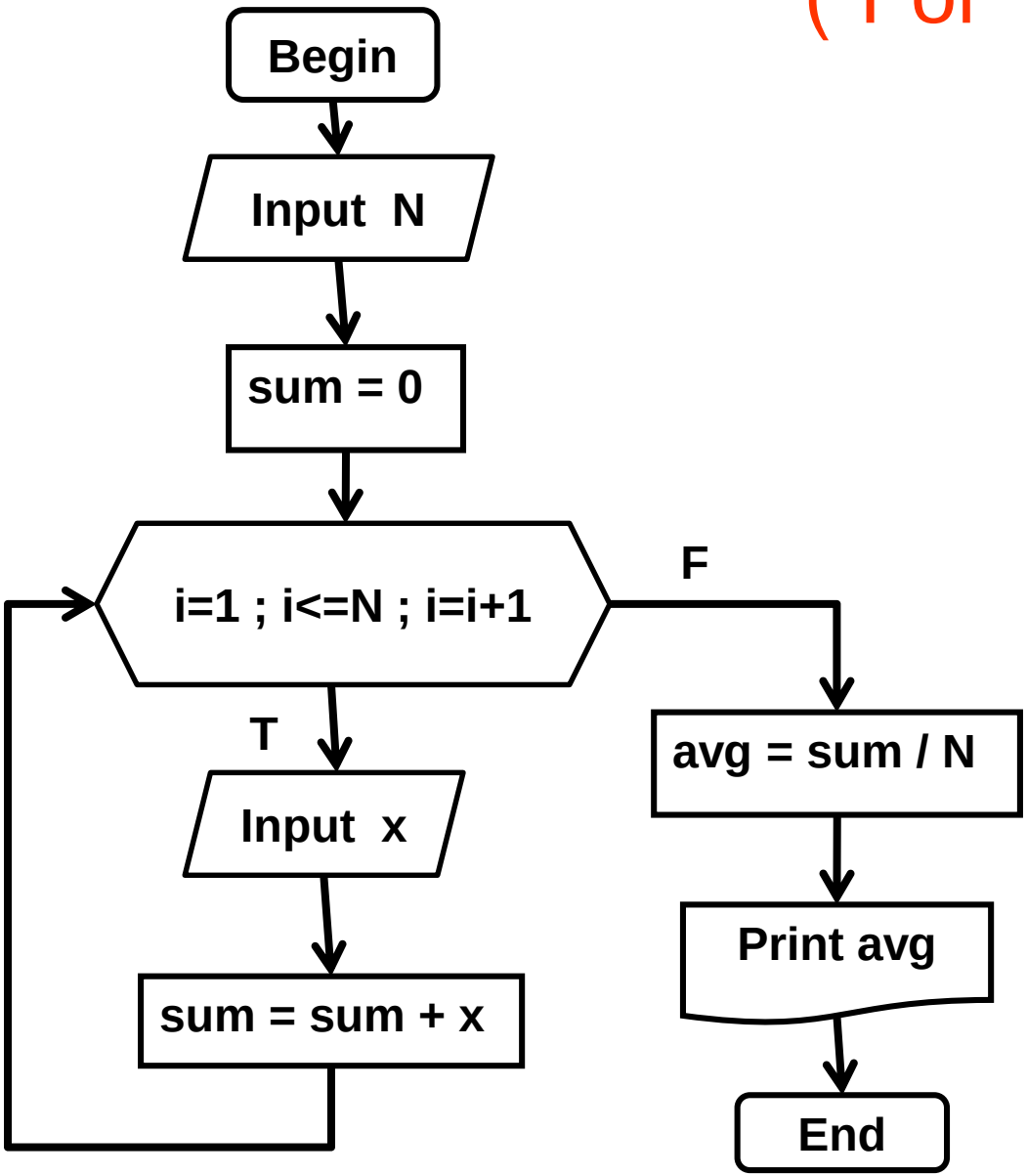


$$avg = \frac{\sum_{i=1}^{n} X_i}{N}$$

```
Begin
  input N
  i = 1
  sum = 0
  while i <= N
  Begin
      input x
      sum = sum + x
      i = i + 1
   End
  avg = sum / N
  print avg
End
```

Example4: Average of N Numbers

("For" loop)



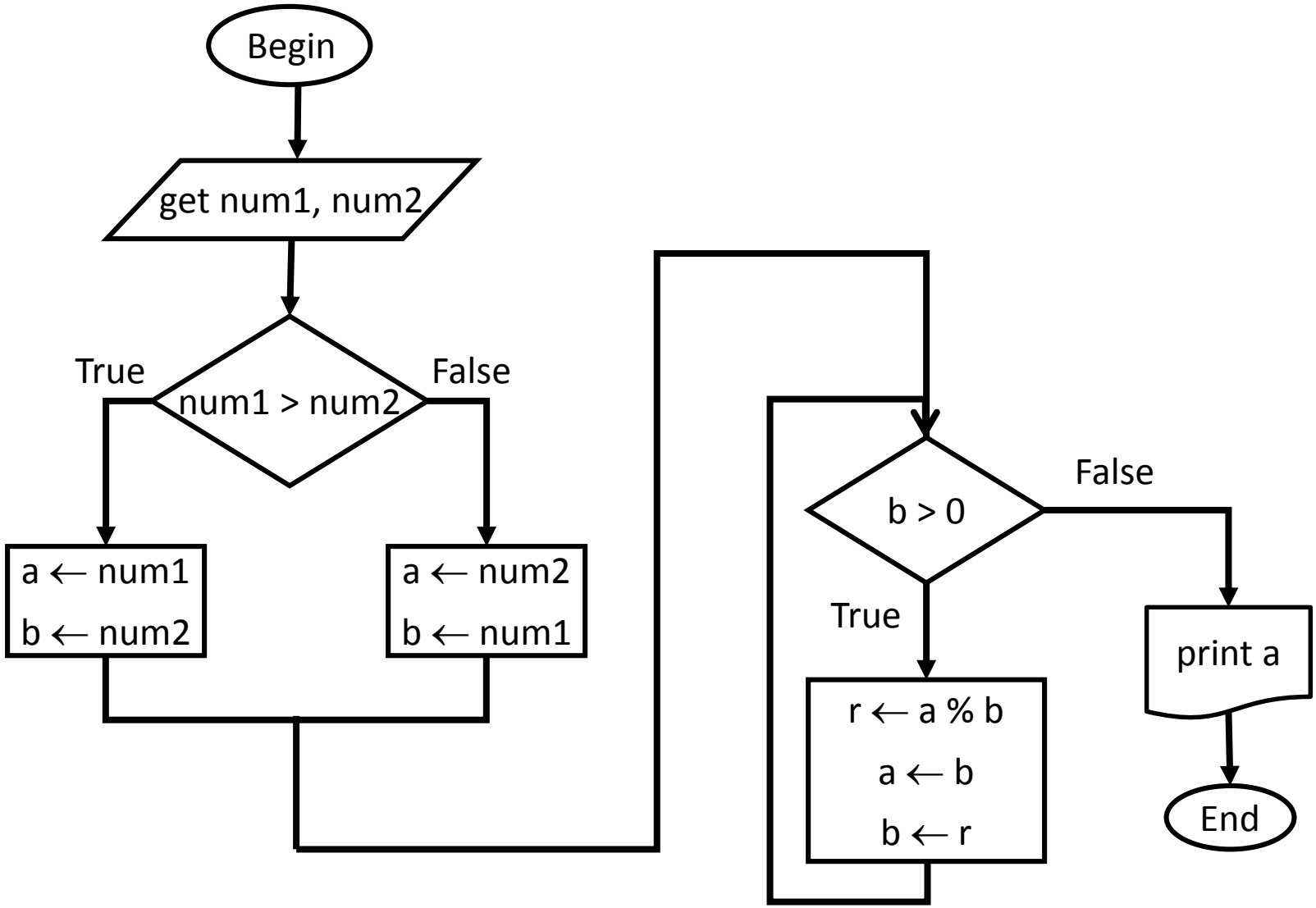

```
Begin
 Input N
 sum = 0
  for i = 1 to N step 1
  Begin
      input x
      sum = sum + x
  End
  avg = sum / N
  print avg
End
```

Example5: Greatest Common Divisor

- Problem: Find the Greatest Common Divisor (GCD) of two numbers.
- Sample numbers= 9702 and 945
- Answer (GCD) = 63

- **<u>Euclid algorithm:</u>**
  - Let a be the bigger number and b the smaller number
  - The gcd of a and b is the same as the gcd of b and a % b

Euclid algorithm flowchart



Running the Euclid algorithm

a	b	r
9702	945	252
945	252	189
252	189	63
189	63	0
63	0	

Answer