REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université de M'sila
Faculté des Mathématiques et de l'Informatique
Département d'informatique

جامعة المسيلة
كلية الرياضيات والإعلام الآلي
قسم الإعلام الآلي

Chapter 2:
The simple sequential algorithm

Algorithms and data structure

Presented by: Dr. Benazi Makhlouf
Academic year : 2023/2024

Chapter contents :

1. Introduction

When the computer was invented, it operated using vacuum tubes, and programming was done by entering zeros (0) and ones (1). With the invention of the transistor and integrated circuits, computers became smaller in size and their capabilities increased. High-level programming languages were created to make programming tasks easier.

There are three main ways to represent an algorithm:

1. **Pseudocode**: This involves describing the algorithm using languages that are close to natural language.
2. **Flowchart**: This involves representing the algorithm using arrows and geometric shapes to describe the steps and decisions.
3. **Programming languages**: This involves writing the algorithm directly in a specific programming language.

2. Concept of language

Language

It is the way in which a computer understands human commands. It consists of characters, vocabulary, grammatical rules, and meanings.

Programming language:

It is an intermediate language between human language and machine language. It provides us with a framework to develop programs that a computer can execute.

It allows us to describe data structures and the operations to be performed.

Types of programming languages:

- **Interpreted**: If the program is read and executed instruction by instruction by an interpreter. Examples include Matlab and web languages.
- **Compiled**: If the program is fully translated into machine language by a compiler before execution. An example is the C language.

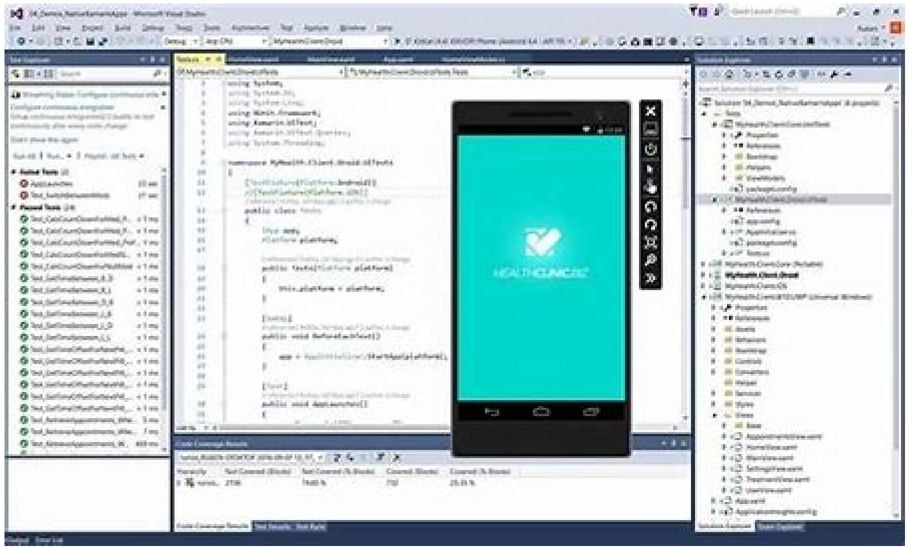
Integrated Development Environment (IDE) 1/2

It is a set of programs that provide all the necessary tools for the application development process, which includes designing, developing, testing, debugging, and deploying applications.

- **Code Editor:** It is a tool used to write and edit source code. It provides features such as syntax highlighting, code completion, and code formatting to facilitate the development process.
- **Project Manager:** It is a tool that helps organize and manage the files and resources related to a software project. It allows developers to structure their code, manage dependencies, and build the project.

Integrated Development Environment (IDE) 2/2

- **Debugger**: It is a tool used to identify and fix errors, or bugs, in a program. It allows developers to track the execution of their code, set breakpoints, and inspect variables to find and resolve issues.
- **Shortcuts** for compiling and running the program: These are keyboard shortcuts or menu options provided by the IDE to quickly compile and execute the program without manually navigating through multiple steps.



example : Dev-C++, Embarcadero, visual studio...

Comments

These are texts that are ignored during compilation and are not part of the program.

There are two types:

- Single-line comments: They start with // (double slashes) and end at the end of the line.
- Multi-line comments: They start with /* (slash asterisk) and end with / (asterisk slash).

Example:

```
// Single-line comment
/* Multi-line
comment */
```

reserved words

These are words that have a predefined meaning in the programming language and cannot be used to identify new elements.

Example:

In algorithmic language, "algorithm", "begin", "end", "and", "if".

In C language, "**if**" and "**while**".

Identifier

It is the name given by the programmer to any element of the algorithm that they want to create. Examples include algorithm name, variable, type, constant, function, etc.

Rules for identifiers:

1. Can consist of A-Z, a-z, 0-9, or _
2. Must be a single word (no spaces)
3. Should not contain special characters like ;:!/.?§…
4. Must not start with a digit
5. Must not be a reserved word
6. Must be unique, meaning no two elements can have the same name

Note: It is recommended to use meaningful names.

Examples:

Valid identifiers: x, pi, Mat_info, isEmpty, n5, _begin, _0a

Invalid identifiers: α, ج, π, élève, 3a, Mat info, begin (algo), if (C)

3. The structure of an algorithm

An algorithm consists of data and instructions, and it is very similar to a cooking recipe. It consists of a header, declarations, and instructions, and it takes the following form:

```
Algorithm name
  Declaration
Begin
  Instructions
End
```

- **Header:** Composed of the word "Algorithm" followed by the name that explains the problem to be solved. It must be a valid identifier.
- **Declarations**: Reserved memory space for data (constants and variables).
- **Instructions**: A set of commands that will be executed to solve the problem. It starts with "Begin" and ends with "End".

Data:

- Values
- Constants
- Variables
- Data types

Values

- Numbers like: 2, -7, 3.12, 6.2e-7
- Characters always enclosed in single quotation marks, such as:‘k’, ‘1’, ‘?’, ‘ ‘, ‘\n’ , ‘خ’
- Strings always enclosed in double quotation marks, such as: "azety", "السلام عليكم" , "k" , "1" , ""
- true, false

Data types

Type: It refers to the domain to which the data belongs, such as numbers, text, images, audio, or video.

The type determines how the bits are translated and the memory size to reserve, i.e., the number of bits, and the allowed operations.

There are 5 basic types in algorithms:

1. **Integer**: -5, 0, 1, 13
2. **Real**: -7, 0, 1., 3.14, 2.7e03
3. **Boolean**: It only contains two values: true or false.
4. **Character**: It includes all the icons on the keyboard: digits, letters in all languages, and printed (visual) and non-printed symbols: 'a', 'M', '1', '+', ',', 'س'.
5. **String**: A collection of characters, with a length of 0 or more, always enclosed in double quotation marks: "computer science", "Good luck\n", "1", "3.14".

constant

A value (numeric or symbolic) that has a name and cannot be changed during program execution.

Declaration:

```
        Const Identifier = value
```

- **Const** or **Constant**: These are two reserved words.
- Identifier: It is the name given to the constant.

Example:

```
        Const
                Pi = 3.1415926
                DEP = "قسم الاعلام الالي"
```

Advantages of constants:

- Compact code: Replace a long phrase with a short word. Use "Pi" instead of 3.1415926.
- Avoid errors: Use a meaningful name. For example, use "PI" instead of 3.1415926.
- Simplify maintenance: The value needs to be modified in only one place.

Variables

A memory location used to store data. It has a name, a type, and a value.

- **Name**: An identifier used by the programmer to manipulate the variable. For example, weight.
- **Type**: The data type of the variable. For example, integer, real, character, string.
- **Value**: The data stored in the variable, which can change during program execution. For example, 5, 'v', "azrty".

Variable Declaration:

```
      Var Identifier : Type
```

- **Var** or **Variable**: These are two reserved words used for declaring variables.
- Identifier: The name given to the variable.
- Type: The data type of the variable.

Examples:

```
      Var
            age : integer
            x, y, z: real
```

5. Instructions

- Basic Operations:
- Expression
- Statement or Instruction and Block of Instructions:
- Assignment Instruction:
- Input Instruction: read()
- Output Instruction: write()

Basic operations

Arithmetic operations

Op	C	comment	example
- +	- +	sign	+3 -7 -a
- +	- +	The two operands are integers, the result is an integer. one is real, result is a real number.	5+3.0
*	*	For the multiplication, like addition and subtraction.	5*3 integer
/	/	for real division. The result is always a real number	5/3 ou 5.0/3 in C
mod	%	To calculate the remainder of the division. Both operands are integers The result is always an integer.	5 mod 3 ou 5%3 in C
div	/	To calculate the quotient (integer division). Like the remainder	5 div 3 or 5/3 in C
^		To calculate the power, in C we use the pow() function, the result is a real number	5^2 or pow(5,2) in C
√		To calculate the root, in C we use the sqrt() function, the result is a real number	√5 or sqrt(5) in C

Basic operations

Relational operators comparison

Operation	C	comment
>, >=, <, <=		the same in C
=	==	In C, "==" twice = is read as equal, while "=" is used for assignment and read as receive.
≠	!=	The "!=" operator is read as "not equal to"

Observation:

The result of a comparison operation is always of type boolean, which means it can only have two possible values: true or false.

Basic operations

Logical operators

Opération		C	comment
not	negation	!	true if the operand is false. and false if true.
and		&&	true if both operands are true, otherwise false
or		||	False if both operands are false, otherwise true
xor	or exclusive		true if one is true and the other is false, otherwise false
=	equivalence	==	true if are equal.

Basic operations

String operators

+ is used to concatenate two character strings.

For example: "good" + " + " luck gives "goodluck”

priority

priority	the operation
0	()
1	+ and - signs, not.
2	* / mod div
3	- +
4	>, >=, <, <=
5	≠ , =
6	and
7	or

if priority is equal, priority is given to the left-hand operation

Expression

An expression is a combination of variables, constants, operators, and functions that evaluates to a single value.

For example: Assume a=2, b=3 and ok=true

expression	result	expression	result	expression	result
5	5	a+3	5	ok	vrai
a	2	"In"+"fo"	Info	a*(b-7)>8 **and** ok	faux

Instruction

An instruction is a single command or action that is executed by the computer. It can be a basic operation, a control structure, or a function call.

block of instructions also known as a code block or compound statement

- a set of instructions that begins with the word **begin** and ends with **End**, or begins with a reserved word that defines the start as **if** and ends with the word **end** + the start word as **end if**. It allows for the execution of multiple instructions as a single unit.

- In C, a block of instructions,, is enclosed between curly braces {}.

example :

algorithm	C
`Begin`	**`{`**
`Instruction 1`	`Instruction 1 ;`
…	…
`Instruction n`	`Instruction n ;`
`End`	**`}`**

Assignment

is the process by which we store a value in a variable.

Syntax :

variable ← expression

- ← is read as "assigns" or "gets“ or receives. The arrow always points towards the variable
- the value of the expression and the variable must be of the same type, or of compatible types.
- Before a variable can be used, it must be declared and initialized.
- To obtain the value of a variable or constant, you simply need to write its name.

Example

`a←5`	a receives 5
`b←a*2`	b receives 10
`a←0`	a receives 0
`b←b-1`	b receives 9
`c←'b'`	c receives the letter b
`d←b>a`	d receives true
`s←"name"`	s receives the word name

a	b
5	?
5	10
0	10
0	9

Input/Output statements

To interact with the user, the developer has two instructions:

- read()
- write()

Input : Read()

is used to retrieve a value entered by the user, typically from the keyboard, and assign it to the variable within the parentheses. It is commonly used for inputting data.

Syntax :

```
        Read(variable)
```

- read() can only be used with variables.
- When the "Read()" instruction is encountered, the execution pauses until the user enters the data. The input process is completed by pressing the Enter key.
- Several variables can be entered at once, separated by a comma ",".
- Always before the ‘Read()' instruction, the ‘Write()' instruction is used to explain to the user what is being asked to enter.

Example

```
        Read(name)                              Read(a, b)
```

Output : Write()

It displays on the screen everything we put inside its brackets. It is always used to print results.

Syntax :

```
        Write(expression)
```

or

```
        Write("message")
```

- expression, calculated to obtain a single value, which will be displayed on the screen.
- "message": is any text to be displayed as is on the screen. It is not calculated. It's in any language. It must be enclosed in double quotation marks, which are not displayed.
- Several values and texts can be displayed at once, separated by ",".

Example :

```
a←5
Write(a+3)                              Write("a+3")
Write(" square of ", a, " is ", a*a)    Write("b=",a)
```

6. Building a simple algorithm

- The header: Algorithm + name
- The declaration: variables and constants
- Instructions: The instruction part generally consists of three basic steps:
    1. The first step, "Inputs": the data needed for the execution is entered using the "Read()" instruction.
    2. The second step, "Processing": It contains a set of instructions required to solve the problem, using the assignment instruction.
    3. The third step, "Outputs": the results are presented using the "Write()" instruction.

Example 1:

Write an algorithm to calculate the area of a circle.

```
Algorithm area_cercle
Const
     P=3.14
Var
   r, a: integer //r is the radius and a is the area
Begin
   Write(" Enter radius ")
   Read(r)
   a←p*r*r
   Write("The area of the circle is:" , a)
End
```

Example 2:

Write an algorithm that calculates the average for ADS1.

```
Algorithm avg_ADS1
Var
   exm, td, tp, avg : real
Début
 Write("Enter the exam grade, the tutorial grade, and
       the practical work grade.")
 Read(exm, td, tp)
 avg ←(exm*3+td+tp)/5
 Write("La moyenne est:" , avg)
Fin
```

Running an algorithm

It aims to find out the value of each variable after each instruction.

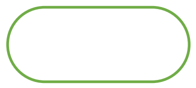
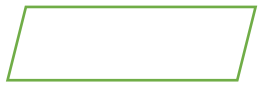
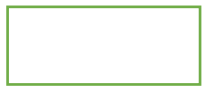
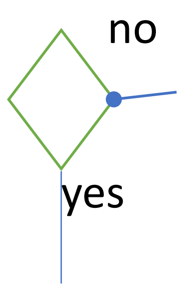
After each assignment or reading, the value of the variable changes.

Example

Algorithm mirror	Execution		
Var			
a, b, c:integer	a	b	c
Begin			
a←357	357	?	?
c←0	357	?	0
b←a **mod** 10	357	7	0
c←c*10+b	357	7	7
a←a **div** 10	35	7	7
b←a **mod** 10	35	5	7
c←c*10+b	35	5	75
a←a **div** 10	3	5	75
b←a **mod** 10	3	3	75
c←c*10+b	3	3	753
End			

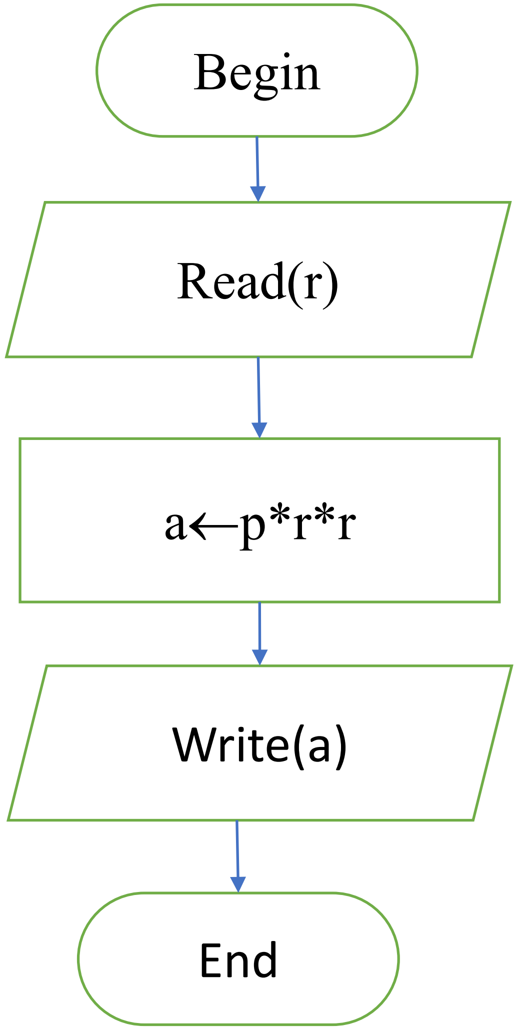
7. Flowchart

An algorigram, also known as a **flowchart**, is a visual representation of an algorithm using different shapes and arrows to illustrate the sequence of operations. It provides a clear and structured view of the algorithm's logical flow, making it easier to understand and analyze.

	Begin, end, interruption
	Input - Output
	Treatment as assignment
	Choice with condition

Example :

the algorithm that calculates the area of a circle



8. Translation into C language

- "C" is a high-level imperative language that is fully compiled. It is one of the most widely used programming languages in the world and is considered the parent language of many other programming languages.
- "C" is case-sensitive, meaning it distinguishes between uppercase and lowercase letters. For example, <main> is not the same as <Main>, <MAIN>, or <mAin>. Therefore, it is recommended to always write in lowercase.
- All simple statements (declarations, assignments, input/output, and return statements) in "C" must end with a semicolon ";".

The preprocessor

Before compiling the source code, the file is processed by a special software known as the preprocessor, which allows the inclusion of other files (libraries) and replacement of words with other phrases (macros).

Preprocessor directives always start with #.

#include: Used to include the content of a file in the current program code.

#define macro_name replacement_text Used to define a macro with a specific name and replacement text.

Example:

- To use I/O functions (scanf and printf), we include the stdio.h library.
- To use mathematical functions (sin, cos, exp, pow, sqrt, ...), we use the math.h library.

```
#include <stdio.h>
#define N 10
```

The preprocessor replaces all occurrences of the word N with 10.

The types

- Integers in the algorithm from -∞ to +∞

Types	Size in bytes	Size in bits	interval
char	1	8	-2^7 ,2^7-1
short	2	16	-2^{15} ,2^{15}-1
long	4	32	-2^{31} ,2^{31}-1
int	4	32	-2^{31} ,2^{31}-1

In algorithms, natural numbers range from 0 to +∞, and we commonly use integers to represent them. In C, we can use the <**unsigned**> keyword before a data type to express natural numbers without negative values.

Types	Size in bytes	Size in bits	interval
unsigned char	1	8	0 ,2^8-1
unsigned short	2	16	0 ,2^{16}-1
unsigned long	4	32	0 ,2^{32}-1
unsigned int	4	32	0 ,2^{32}-1

The types

Real numbers

Types	Size in bytes	precision
float	4	6
double	8	8
long double	10	8

- In C, there is no built-in Boolean type, but instead, we use int to represent **Boolean** values. In this representation, true is represented by the value 1, and false is represented by the value 0. Any number other than 0 is considered as true.
- The type used for characters in C is **char**.
- To represent strings in C, we use arrays of char (Chapter 5) **char[]**, or pointers (second semester) **char***.
- The **char** type is used for both **integers** and **characters**, where each character is associated with a numeric value.
- In this course, we use **char** for characters, **int** for integers and Booleans, and **float** for real numbers.

Declaration of variables and constants

Variables

Syntax :

```
Type variable;
```

example :

```
int age;
char grade;
float x, y, z;
Banane b;
```

Constants

Syntax :

const Type Identificateur=valeur;

ou

Type **const** Identificateur=valeur;

example :

```
int const N = 10;
const float PI = 3.1415926;
```

Note: Macros can be used to declare constants.

```
#define DEP "Département
informatique"
```

The assignment

We use = instead of ← and read it as receives and not equals.

```
variable = expression;
```

Example :

```
a=5;
b=a*2;
a=0;
d=b>a;
```

Declaration with initialization

```
float x, y=3, z;//y is initialized with 3
```

If the assignment appears more than once, priority is given to the right. Such as:

```
b=3 ;
a=b=5+3 ;
```

Assignment shortcuts in C.

expression	comment	example
v+=exp ; ⇔ v=v+(exp) ;	parentheses are important	x=2 ; x*=5+3 ; ⇔ x=x*(5+3) ; ≠ x=x*5+3 ; x becomes 16
v-=exp ; ⇔ v=v-(exp) ;		
v*=exp ; ⇔ v=v*(exp) ;		
v/=exp ; ⇔ v=v/(exp) ;		
v%=exp ; ⇔ v=v%(exp) ;		
v++ ; ⇒ v=v+1 ;	When you use the post-increment or post-decrement operators, the current value of the variable is used in the expression, and then the value of the variable is incremented or decremented by 1 after the operation is completed.	x=2 ; y=3+x++ ;⇔ y=3+x ;x=x+1 ; In other words, y becomes 5 and x 3
v-- ; ⇒ v=v-1 ;		
++v; ⇒ v=v+1 ;	When you use the pre-increment or pre-decrement operators, the value of the variable is incremented or decremented by 1 before the expression is evaluated, and then the updated value is used in the expression.	x=2 ; y=3+ ++x ;⇔ x=x+1 ; y=3+x ; In other words, y becomes 6 and x 3
--v ; ⇒ v=v-1 ;		

printf (print formatted)

The printf function, defined in the stdio.h library, is used to write formatted data to the screen.

Syntax :

```
printf (format, expression_1,… , expression_n);
```

- **expression**: calculated to be displayed in <format> format.
- **`Format:`** is a text or string of characters that is displayed exactly as it is on the screen. However, it may contain special symbols like "%" to express the format of expressions and the “\" character as an escape symbol.

The format takes the following form:

```
%type_char
```

Where type_char can be:

%d	**d**ecimal (10)	%o	**o**ctal (8)	%x	he**x**adecimal (16)
%u	natural **u**nsigned	%i	**i**nt	%f	**f**loat
%c	**c**har	%s	**s**tring	%e	scientific format

Escape sequence

"Escape technique" or "Escape sequence" is a method used in programming and text processing to represent certain characters that have special meanings, such as the newline character "\n" or the tab character "\t", which cannot be directly represented in a string.

In C, the escape character is the backslash \, and we use it to add a new line '\n' or a tab (a large space) '\t'. and to display " we use ‘\“’, to print \ we use ’\\’ and to print % we use %%.

example :

```
printf("Hello");
```

Display Hello

```
int a=13;
printf("a=(%d)10\ta=(%o)8\ta=(%x)16\n",a ,a ,a);
```

```
a=(13)10   a=(15)8   a=(D)16
```

```
a=66;
printf("a=%i\ta=%f\ta=%c\ta=%c\n",a ,a ,a ,a+32);
```

```
a=66   a=0.000000   a=B   a=b
```

Because 66 is not necessarily 66 in float

And 66, if we think of it as a character, represents the letter B, while 66 + 32 = 98 represents the coding of the letter b in lower case.

```
float pi=3.1415926;
printf("%f\t%.4f\t%06.2f" ,pi ,pi ,pi);
```

```
3.141593      3.1416       003.14
```

scanf (scan formatted)

The scanf function, defined in the stdio.h library, is used to read formatted data from the keyboard.

Syntax :

```
scanf(format, &variable_1, … , &variable_n);
```

- variable: Represents the variable name. It is preceded by <&>, unless it is a string variable (pointer type).
- format: String representing the reading format.

    The **format** takes the following form:

    %type_char (same as in printf)

Example :

```
scanf("%s", name);

scanf("%d%f",&a, &b);
```

scanf problem with characters

When reading a character using scanf("%c"...), the user enters the first character, then presses the Enter key, which in turn produces the character '\n', which is not deleted from memory, and when the program encounters the scanf("%c", ...) instruction for the second time, it doesn't wait for what the user will enter, but instead assigns the character \n to the second variable.

To avoid this problem, in the second scanf we use a space ' ' after % like this: scanf("% c"...). Or we use the **getch()** function

Example :

```
        scanf("%c", &c1) ;
        scanf("% c", &c2) ;
```

scanf problem with strings.

The problem arises when we try to enter a string containing spaces in a variable.
for example:

```
scanf("%s%s", v1,v2) ;
```

It puts the first word in v1 and the second in v2

```
scanf("%s", v1) ;
```

When we enter the words math info and press Enter, the program assigns the first word to v1 and the second word will be lost.

To avoid this problem, we use the gets function defined in the string.h library.

```
#include <sting.h>

gets(v1);
```

Structure of a C program

```
#include <stdio.h>
   Declaration of constants, types and variables
int main()
{
   Declaration of constants and variables
   Instructions
   return 0;
}
```

An example of how to translate an algorithm into C

algorithm	C
Algorithm area_cercle	
Const P=3.14	**Const** float P=3.14;
Var r, a:integer	int r, a;
//r radius and a area	//r radius and a area
Begin	int main() {
Write("Enter radius")	printf("Enter radius\n");
Read(r)	scanf("%d", &r);
a←p*r*r	a=p*r*r;
Write("The area of the circle is:" , a)	printf("The area of the circle is: %d" , a);
End	}

example

Write a program to calculate the average for ADS1.

```
#include <stdio.h>
int main() {
   float exm, td, tp, avg ;
   printf(" Enter exam grade \n") ;
   scanf("%f", &exm) ;
   printf(" Enter TD grade \n") ;
   scanf("%f", &td) ;
   printf(" Enter TP grade \n") ;
   scanf("%f", &tp) ;
   avg = (exm * 3 + td + tp) / 5 ;
   printf(" The average is %.2f" , avg) ;
   return 0;
}
```

End Chapter 02