Chapter 3

Data Types, Arithmetic Operators and Basic I/O

Objectives:

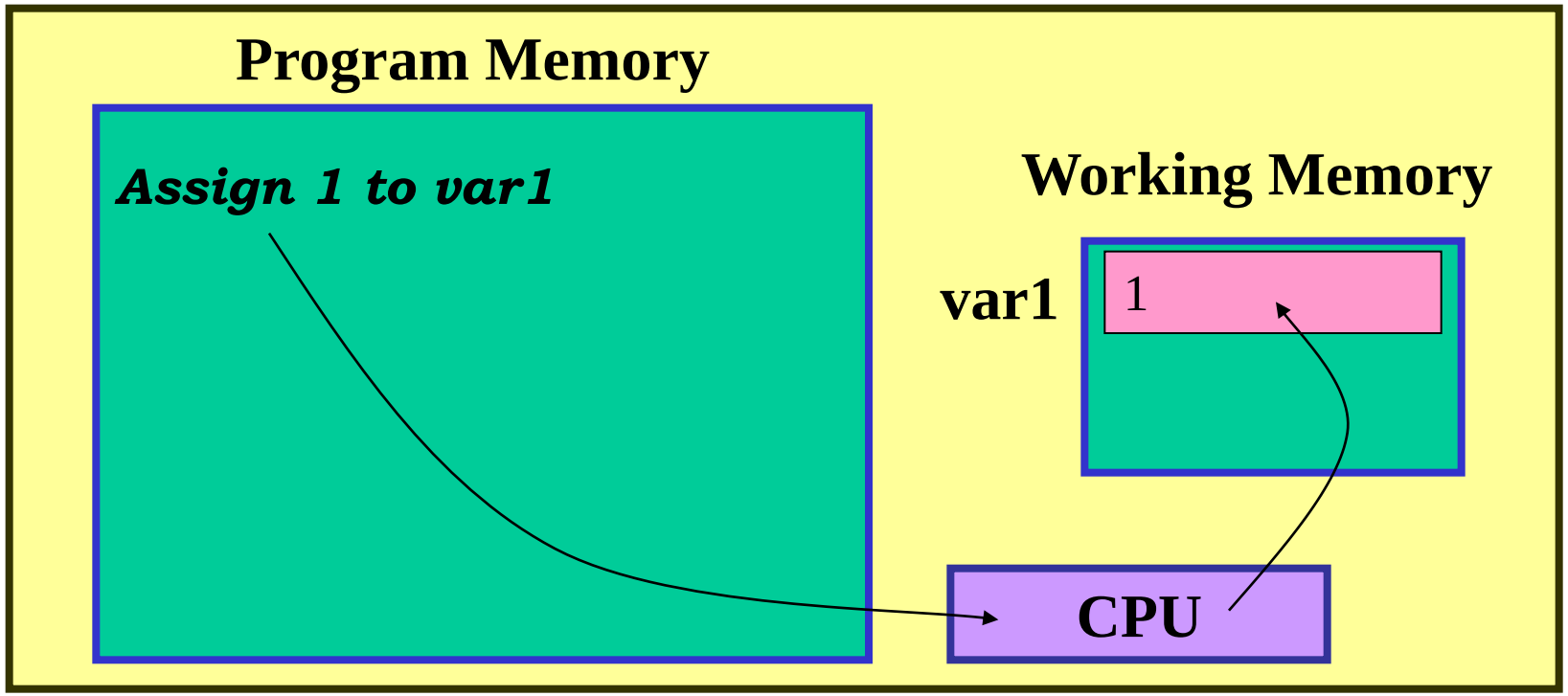
- Variables
- Data Types in C
- Arithmetic operations
- Input functions
- Output functions

3.1 The Variable

- The basic object in any computer language is the ***variable***
- A variable corresponds to **a location** in memory and has **an address**.
  - The type of variable defines how much memory it takes to store a value of its type, e.g. it takes less memory to store an integer value than a real value
- The variable name corresponds to the address of the variable (when a program is compiled, a name is translated to an address for use by the CPU to access the contents of the variable)

The Assignment Operation

- Values are assigned to variables, which means that a value is stored in the memory location reserved for the variable.
- The assignment operation is destructive, i.e. anything store in the variable is lost when a new value is assigned to the variable
- Consider the following pseudo-code: the CPU executes the instruction (in reality a number of CPU instructions) and stores the value 1 into the variable



3.2 Data Types in C

- The basic "objects" in C are variables and symbolic constants.
- The content of a variable can be changed by a program but a symbolic constant cannot be changed.
- All variables and symbolic constants must be declared and defined before usage in a program.
  - Note that declaration of variables is not required when developing pseudo-code

Names of Variables and Symbolic Constants

- All letters `a` to `z` and `A` to `Z`, all numbers `0` to `9` and the underscore _ can be used in the name of a variable or symbolic constant.
- C is case sensitive so `a` is not recognized as being the same letter as `A`.

  E.g.: `STUDENT_1` is not the same name as `student_1`

- A variable name cannot begin with a number and names that begin with _ are often in bad programming style unless they are reserved for a specific set of variables.

- C programmers usually follow the convention that variable names are written in lowercase letters while symbolic constants are written in UPPERCASE letters.

- Reserved words cannot be used as variable names.

  E.g.: `if`, `else`, `while`... are reserved words since they are used in C commands so these words cannot be used as variable names.

- **Use descriptive variable names! Don't be lazy!**

  E.g.: use `student_1` instead of `s1`

Variables

- Variables must be declared via declarations.
  - Declarations are usually placed after the first `{` of `main` (or of a function definition) and before the first command.
  - A declaration contains at least the variable name and its type.

- The type of a variable defines the nature or kind of data that can be stored in the variable.

- A variable consists of a space in memory where the program can store and retrieve data.

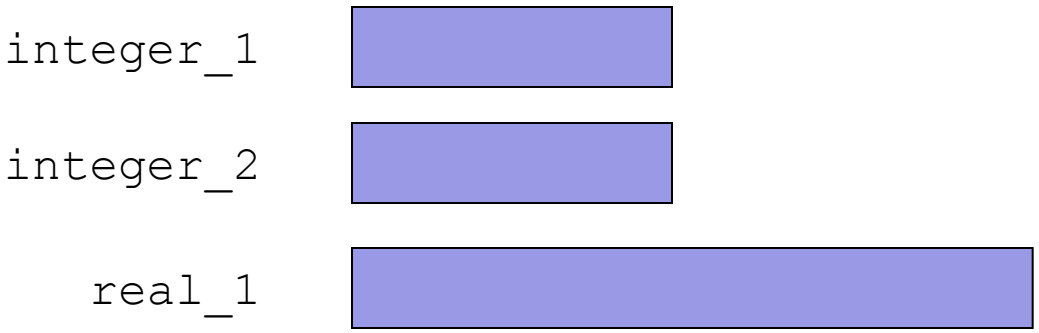
- There are 4 basic data types in C:

`char`	stores one character: `a` to `z`, `A` to `Z`, `!`, `$`,...
`int`	stores an integer: `1`, `101`, `-1462`,...
`float`	stores a real number: `0.5`, `-122.34`,...
`double`	stores a real number in double precision.

- Here are a few examples of variable declarations in C:

```
int integer_1, integer_2;
float real_1;
```

  - A declaration begins with the type followed by the variable names separated by commas `,` and ends with a semi-colon `;`
  - These declarations specify that the variables `integer_1` and `integer_2` are of type `int` and thus can store integer numbers, while `real_1` is a variable of type `float` that can store a real number. Space in memory will be allocated to these variables by the compiler:



Symbolic Constants

- "Magic numbers" sprinkled throughout a program is generally considered to be an example of bad programming style.

- If a "magic number" must be used in the program, then it is best to associate the number to a symbolic constant that has a descriptive name and then to use the constant in the program.

- Symbolic constants are defined using the pre-processor directive `define`E.g.:

```
#define NBR_OF_STUDENTS 455
#define PI 3.141593
```

Define NBR_OF_STUDENTS as 455

Define PI as 3.141593

  - Pre-processor directives do not require a `;` at the end.

- The pre-processor performs a textual substitution of the symbolic constant name with the associated quantity (directly to the right) everywhere in the code before compilation.
  - ➔ No memory space is used for a symbolic constant. They are not variables.
  - ➔ A program cannot change the value associated with a symbolic constant.
  - ➔ Be careful with the `;`

    E.g.: `#define MAX 100;` ***Define MAX as 100***

    will result in a substitution of `MAX` by `100;` everywhere in the program!

3.3 Evaluating arithmetic expressions

- Familiar arithmetic expressions can be evaluated by a computer.
- But remember the definition of a variable in a computer.
- The CPU evaluates an expression to obtain a value
  - For example: x + y
  - The CPU gets the value stored in the variable x, then gets the value store in the variable y and adds the two values
  - What is done with the result of the addition?
- What does the following algebraic expression mean:

  z = x + y
- What does the following pseudo-code mean:

  Assign x + y to z

  (note the use of the word Assign instead of =, why?)

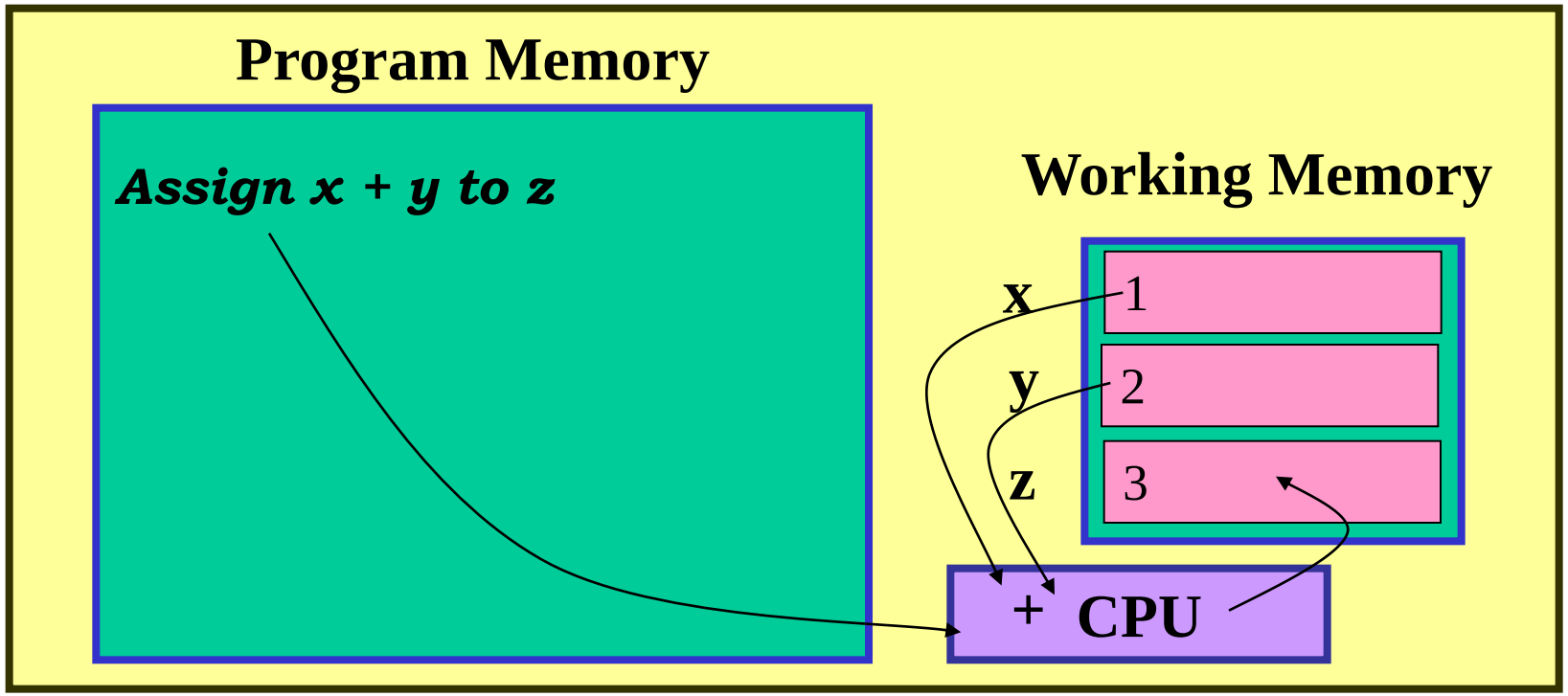
Evaluating expressions (continued)

- Examine how the CPU executes the pseudo-code shown below
- How do you think the CPU executes the following pseudo-code:

  Assign 1+(x+y)/2 to z

  Assign 1+x to x

- The computer does **one thing** at a time, but very fast



3.4 Arithmetic Operators and Arithmetic Expressions in C

- The = operator is the assignment operator: it assigns a quantity to a variable.
  - It is a binary operator; it requires 2 objects, one placed on either side.

    E.g.: integer_1 = 1;

    real_1 = 10.0;
  - The assignment operation is destructive. Any value previously stored in the variable will be erased and the new value will be assigned to the variable.
  - It is possible to make an assignment in a variable declaration.

    E.g.: int integer_1 = 1;

    defines integer_1 as being a variable of type int and stores the value 1 in it.

- The following binary arithmetic operators exist in C:

Operation	C Binary Operator	Typical C Arithmetic Sub-expression
addition	+	a + b
subtraction	-	a - b
multiplication	*	a * b
division	/	a / b
modulus	%	a % b

- The − operator can also be a unary operator when used to negate a variable.
- A standard C math function must be used for exponentiation: a^b
- Parentheses `()` are used to group sub-expressions in order to
  - make expressions more readable, and
  - change the order of precedence of the operators.

    E.g.: `e = (a + b) * (c + d);`

Rules of Operator Precedence

- C evaluates arithmetic expressions according to the following rules of operator precedence.

1. Contents of parentheses are evaluated first.
   - If there are many "levels" of parentheses, then the innermost pair is evaluated first; the next innermost pair is evaluated second…
   - If there are many pairs of parentheses on the same level then they are evaluated left to right.
2. Negation (unary) is evaluated next.

3 Multiplications, divisions and moduli are evaluated next.
   – If there are many then they are evaluated from left to right.

4 Additions and subtractions are evaluated last.
   – If there are many, they are evaluated left to right.

- Examples:

```
e = a * (b * (c + d));
```

3 2 1

```
e = (a + b) * (c + d);
```

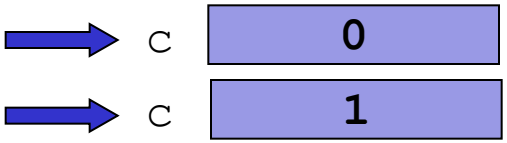
1 3 2

```
e = p * r % q + w / x - y;
```

1 2 4 3 5

The Division of Integers

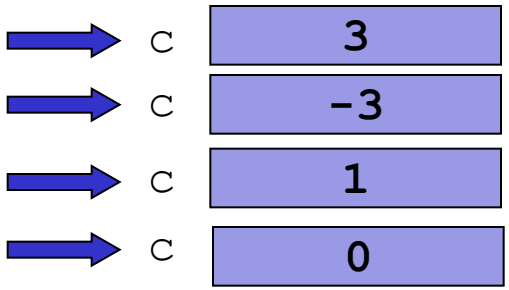
- Suppose that `a`, `b` and `c` are of type `int`.

E.g.:
```
a = 3;
b = 4;
c = a / b;
c = b / a;
```


  - The division of integers returns the quotient.

The Modulus

- Operation that computes the remainder of a division of integers. Suppose that `a`, `b` and `c` are of type `int`.

E.g.:
```
a = 3;
b = 4;
c = a % b;
c = -a % -b;
c = b % a;
c = b % b;
```


Sign of remainder is sign of numerator.

  - The modulus operates on integers only.

Mixed Type Arithmetic Expressions

- An expression that contains variables of many types will be converted in memory by the C compiler according to the implicit rule of promotion.

- The promotion hierarchy of our 4 basic data types is as follows:

```
double   <-  highest type
float
int
char     <-  lowest type
```

  - The hierarchy is organized according to the amount of information that can be stored in the data type.
    - A double variable can contain the same amount of information as all of the "lower" data types.

- The implicit rule of promotion followed by the C compiler when evaluating mixed type expressions is:
  - All variables in an expression are promoted to the highest type (in temporary memory) before evaluation of the expression.

- Consider the following examples of mixed type expressions.

E.g.:

```
int a, b, c;
float x, y, z;
a = 3;
b = 4;
x = 1.0;
y = 2.5;
```

No conversions	`c = a + b;`	→	`c`	7
No conversions	`z = x + y;`	→	`z`	3.5
1 conv. from `int` to `float`	`z = a + b;`	→	`z`	7.0
1 conv. from `int` to `float`	`z = x + a;`	→	`z`	4.0
1 conv. from `float` to `int`	`c = x + y;`	→	`c`	3
2 conversions	`c = x + a;`	→	`c`	4
1 conv. from `int` to `float`	`z = a / b;`	→	`z`	0.0

Loss of information!

- Care must be taken when storing a real number into an `int`. The real number will be truncated and only the integer portion will be stored.

- Careful with the division of integers; in the last example, `a / b` is evaluated as an integer division, the result `0` is promoted to a float `0.0` and stored in `z`.

The Cast Operator

- The unary `cast` operators can be used to force the type conversion of a variable.

  E.g.: (same variable definitions as in the previous example)

  ```
  z = (float) a / b;
  ```
   `z` **0.75**

  - `(float) a` forces the conversion of `a` to type `float`,
  - according to the implicit rule of promotion, `b` is converted to type `float`,
  - the division is then evaluated as a division of real numbers and the result is assigned to `z` which is of type `float`.

- A unary operator operates on one argument only: the argument directly to its right.

- A cast operator exists for each data type in C: `(double)`, `(float)`, `(int)`, `(char)`.

Summary of Hierarchy and Associativity of Arithmetic Operators

`()`	left to right
`- (type)`	right to left
`* / %`	left to right
`+ -`	left to right
`=`	right to left

3.5 Basic Input and Output

Basic Output

- Recall that a computer program interacts with the operating system
- To output a message to a screen, some call to the operating system is required.
- Shall represent such a call simply with the word ***Print*** as in

  Print "Welcome to the U. of Ottawa"

  – Note that the string to be printed is enclosed in quotes.
- To print the value of a variable use the following form of pseudo-code

  Print "The value of x is", x

  – If x contains the value 3, then the message "The value of x is 3" will be printed.

  – What message does the following print, given that x contains 4 and y contains 5:

  Print "The value of x is ", x, " and y is ", y

Basic Input

- To read a value from the keyboard, some other call to the operating system is required.

- Shall represent such a call simply with the word ***Read*** as in

  Read a value into var

  – The value read from the keyboard is assigned to the variable *var*.

3.6 Basic Input and Output in C

Basic Output

- `printf` can also be used to print out numbers and the content of variables to the screen:

```
E.g.:  printf("%d\n", 100);
       printf("%f\n", 101.3);
       printf("%d\n", integer_1);
       printf("%f\n", real_1);
```

- The `%d` is the conversion specifier used to print out an integer and the `%f` is the specifier used to print out a real number.
  - The conversion specifier informs `printf` of the type of the data to be printed.
  - A conversion specifier always begins with a `%`
  - The specifier must be enclosed in a pair of quotes " and "
  - As we shall see, there are many more conversion specifiers.
- More than one "object" can be written out using a single `printf`:

```
E.g.:  printf("The sum is %f\n", sum);
```

  - The arguments to printf are seperated by a comma `,`
- A `%` can be printed out as part of a message.
  - Use two percent symbols `%%` to print one out.

Basic Input

- Data can be entered from the keyboard using the standard C function `scanf`.

E.g.:
```
scanf("%d", &integer_1);
scanf("%f", &real_1);  // real_1 is a float
scanf("%lf", &real_2);  // real_2 is a double
scanf("%d%f", &integer_1, &real_1);
```

- `scanf` requires a conversion specifier corresponding to the type of data to be read.

- An ampersand `&` must be placed before the variable name into which the data will be read.

Standard Input and Output Header File

- **Information** related to the standard functions `printf` and `scanf` is contained in the standard input/output header file `stdio.h`.

- The pre-processor directive `#include <stdio.h>` placed before `main()` forces the compiler to include this information in the compilation process.

Example 1

Develop a program that requests three integers from the user and prints them in the reverse order, as well as the sum of the three integers.

- Start with pseudo-code.
- Then provide the C program.

3.7 Counters

- Counters are often used in loops to keep track of the number of times that a certain task is done. A counter in C is usually a variable of type `int` that we increment or decrement:

E.g.:
```
int ctr = 0;
ctr = ctr + 1;      Increment ctr
ctr = ctr - 1;      Decrement ctr
```

- Recall how the computer interprets the above instructions.

- There exists in C, special operators for adding or subtracting 1 from a variable; these operators are called the increment operator `++` and the decrement operator `--`.
  - These operators will not be presented in this course – they present subtleties that can easily lead to bugs.
  - You may use them, but at your own risk.

3.8 More on Numerical Types

Type unsigned int

- A variable of type `int` can store a positive or a negative integer.
- A variable of type `unsigned int` can store a positive integer only.
  - An `unsigned int` is read using the `%u` conversion specifier in `scanf`.
  - An `unsigned int` is written using the `%u` conversion specifier in `printf`.

Real Numbers

- In general, a real number can be written in scientific notation as:

  mantissa $\times 10^{\text{exponent}}$ where: $1.0 \leq |\text{mantissa}| < 10.0$

- The number of digits in the mantissa determines the accuracy of a real number while the exponent determines its range .
  - The accuracy and range of a real number depend on the variable type and on the architecture of the computer (machine dependant).
- In C the letter `e` is used to separate the mantissa from the exponent of `10`.

E.g.:

Real number	Scientific notation	C notation
25.6	2.56×10^{1}	`2.56e1`
-0.004	-4.0×10^{-3}	`-4.0e-3`
1.5	1.5×10^{0}	`1.5e0`

Numerical Conversion Specifiers to be Used in printf and scanf

Type	printf	scanf
short	`%d`, `%i`, `%hd`, `%hi`	`%hd`, `%hi`
int	`%d`, `%i`	`%d`, `%i`
long	`%ld`, `%li`	`%ld`, `%li`
unsigned short	`%hu`	`%hu`
unsigned int	`%u`	`%u`
unsigned long	`%lu`	`%lu`
float	`%f`, `%e`, `%E`, `%g`, `%G`	`%f`, `%e`, `%E`, `%g`, `%G`
double	`%f`, `%e`, `%E`, `%g`, `%G`	`%lf`, `%le`, `%lE`, `%lg`, `%lG`
long double	`%Lf`, `%Le`, `%LE`, `%Lg`, `%LG`	`%Lf`, `%Le`, `%LE`, `%Lg`, `%LG`

- Careful with the conversion specifier for a `double` in a `printf`.
- The conversion specifier `%e` prints out the number in scientific notation: `1.0e0`
- The conversion specifier `%E` prints out the number in scientific notation: `1.0E0`
- The conversion specifier `%g` prints out the number in `%f` or in `%e`
- The conversion specifier `%G` prints out the number in `%f` or in `%E`

On the Range of Numerical Types

- The range of numbers that can be represented by each type depends on the computer system used since the latter fixes the amount memory (bytes) used to represent each numerical type.
  - The 3 different types of integer can represent 3 different ranges of positive and negative integers.
    - The ANSI C standard states that the range of values for an `int` must be at least the same as the range for a `short` and no larger than the range for a `long`.
  - The 3 different types of unsigned integer can represent 3 different ranges of positive integers.
  - The 3 different types of real have three different accuracies and can represent three different ranges of real number.

- The file `limits.h` contains the symbolic constants that define the range of all integer types in C. Typical values are given in this appendix.

- The file `float.h` contains the symbolic constants that define the range and accuracy of all real data types in C. Typical values are given in this appendix.

Example 2

- Problem Statement: Software is required to translate temperature from Fahrenheit to Centigrade.
- Gathering of Information and Input/Output Description
  - The equation for translating the temperature from Fahrenheit to Centigrade is

    $T_C = (5/9) * (T_F - 32)$
  - Input/output:

    Input consists of a temperature value in ^oF

    Output is a temperature value in ^oC
- Test Cases and Algorithm Design
  - Discuss possible test cases for this problem.
  - Develop an algorithm using pseudo-code for the software.
- Implementation
  - Develop a C program for the software.


Thank You!
Ευχαριστώ
MULTUMESC
ขอบคุณ
Vielen Dank
Merci
DMnvwd
شكراً
Teşekkürler
متشكرم
Gracias
Grazie
Bedankt
THANK YOU
Dankie
Obrigado!
Köszönettel
Hvala
Asante
شکریا
Díky
謝謝
GADDA GUEY
WAD MAHAD
SAN TAHAY
감사합니다
Urakoze