PROBLEM SOLVING AND C PROGRAMMING

Fourth Year Students
(Computer Science Section)

Dr. MAHMOUD A. SOLIMAN
The Art and Science of Problem Solving

1. Combination of Art and Science
   - Art is transformation of problem (Algebra)
   - Analyze the problem (input and output)
   - realistic assumptions

Science is the knowledge of equations
   - developed a series of steps
The Art and Science of Problem Solving

• The Engineering and Scientific Methods:
  Engineers and scientists are problem solver
  They must follow these problem analysis steps

  1) Recognize and understand the problem
  2) Accumulate facts (size, voltage, costs, ...)
  3) Select the appropriate theory or principle they apply
  4) Make necessary assumptions (Simplification of the problem.)
  5) Solve the problem
  6) Verify and check results (mathematically correct)
The Art and Science of Problem Solving
• Software Engineers (must follow)

1) Requirements specification (determine what a program to do)
2) Analysis (input and output, equations needed)
3) Design (way to solve the problem, algorithm)
4) Implementation (C program)
5) Verification and Testing (verify solution is correct)
The Art and Science of Problem Solving

• Understanding The Problem:

Case Study: Finding the area and circumference of a circle

PROBLEM: Take the radius of a circle and compute and print its area and circumference

ANALYSIS: One Input (radius)- Two Output (area, circumference) we know the relation between radius and “area and circumference”
The Art and Science of Problem Solving

• Understanding The Problem:

DATA REQUIREMENTS

1. Problem constant  \( \pi \)  3.14159
2. Problem input  radius  /* radius of a circle */
3. Problem output  area  /* area of a circle */
                   circum  /* circumference of a circle */

Relevant Formulas:

area of a circle = \( \pi r^2 \)
circumference = 2 \( \pi r \)
The Art and Science of Problem Solving

Understanding The Problem:

DESIGN: list the steps necessary to solve the problem (algorithm)

**Initial Algorithm:**
1. Get circle radius
2. Calculate area
3. Calculate circumference
4. Display area and circumference

**Complete Algorithm:**
1. Get circle radius
2. Calculate area
   2.1 Assign the product of PI and radius squared to area
3. Find circumference
   3.1 Assign the product of two times PI and radius to circum
4. Display area and circumference

Dr. Mahmoud A. Soliman
The Art and Science of Problem Solving

• Understanding The Problem:

IMPLEMENTATION:

we must write the algorithm as a C program that contains all the information needed for a complete machine-language translation
The Art and Science of Problem Solving

/*
 * Calculate and displays the area and circumference of a circle
 */

#include <stdio.h> /* Directive */
define PI 3.14159 /* Directive */

int main(void) {
    double radius, /* input – radius of a circle */
            area,   /* output – area of a circle */
            circum; /* output – circumference of a circle */
The Art and Science of Problem Solving

• /* Get the circle radius */
  printf(“Enter radius>”);
  scanf(“%f”, &radius);

  /* Calculate the area */
  area = PI * radius *radius;

  /* Calculate the circumference */
  circum = 2 * PI * radius;

  /* Display the area and circumference */
  printf(“The area is %fn”, area);
  printf(“The circumference is %fn, circum);

  return (0);

}
The Art and Science of Problem Solving

• Enter radius> 5.0
  The area is 78.539750
  The circumference is 31.415900

• The program consists of two parts
  1) The preprocessor directive (begin with #) and modify the text before it is compiled
  2) The main function
The Art and Science of Problem Solving

• **TESTING:** We should always examine program results carefully to make sure that they make sense

  We must try more than one test of the program to verify that the program is correct
Example

• /* Display the user’s nickname and the current year
  * in a welcoming message. */

```c
#include <stdio.h>  /* printf, scanf definitions */

int main(void)
{
    char letter_1, letter_2, letter_3;  /* three letters */
    int year;

    printf("Enter a 3-letter nickname and press return>”);
    scanf("%c%c%c", &letter_1, &letter_2, &letter_3);
```
Example

Printf("Enter the current year and press return> ");
scanf("%d", &year);
printf("Welcome, %c%c%c. %d is a great year to study C!
", letter_1, letter_2, letter_3, year);

return (0);
}

Enter a 3-letter nickname and press return> Bob
Enter the current year and press return> 2012
Welcome, Bob 2012 is a great year to study C!
C Program Outline

Preprocessor directives
main function prototype
{
  declarations
  statements
}

Dr. Mahmoud A. Soliman
C Program Outline

• Main Function Prototype

```c
int main(void)
```

This prototype marks the function where program execution begins
C Program Outline

• **Main Function Body**
  The curly braces \{ \} marks the beginning and end of the function body

  The declarations identify the memory cells used by the function

  The statements list instructions to manipulate data according to the program’s purpose
C Program Outline

# include Directive for defining Identifiers from Standard Libraries
SYNTAX: #include <standard header file>
EXAMPLES: # include <stdio.h>
          # include <math.h>

# define Directive for Creating Constant Macros
SYNTAX: # define NAME value
EXAMPLES: # define MILES_PER_KM 0.62137
          # define PI 3.141593
          # define MAX_LENGTH 100
Variables Declaration in C programs

- double radius /* input – radius of a circle */
  area /* output – area of a circle */
  circum /* output – circumference of a circle */
- char letter_1, letter_2, letter_3; /* three letters */
- int year;

The rules for forming user-defined identifiers are as follows:

1) An identifier cannot be begin with a digit
2) An identifier must consist only of letters, digits, or underscore
3) A C reserved word cannot be used as an identifier
4) An identifier defined in a C standard library should not be redefined
Variables Declaration in C Programs

• **Uppercase and Lowercase Letters**
  C programmer must take great care in the use of uppercase and lowercase letters because the C compiler considers such usage significant.

  *Rate, rate, and RATE*

  are viewed by the compiler as different identifier
Program Style

• Pick meaningful names for variable and constant macros so that your program is easy to read

`weight` would be a good name for a variable used to store the weight of an object

`newtons_per_sq_mm` rather than `newtonspersqmm`

Choose identifiers long enough to convey your meaning

`newtons_per_sq_mm` rather than `newtons_per_sq_millimeter`
Program Style

• If you mistype a name the identifier looks like the name of another variable (do not use the same variable names with upper- and lowercase and presence or absence of underscore for different variables)

$LARGE$ and $large$
$x\_coord$ and $xcoord$
### Reserved Words and Identifiers used in our examples

- **Reserved Words**
  - int, void
  - double, char
  - return

- **Identifiers from Standard libraries**
  - printf, scanf

- **User-Defined Identifiers**
  - PI, radius
  - letter_1, letter_2
  - letter_3, circum, year

---

Dr. Mahmoud A. Soliman
Syntax Display for Declarations

- SYNTAX: int variable_list;
  double variable_list;
  char variable_list;

EXAMPLES: int count,
  large;
  double x, y, z;
  char first_initial;
  char ans;
Executable Statements

• Program in Memory

Memory

Machine Language circle characteristic program

Memory
Executable Statements

• Arithmetic Operator
  +  Meaning  Addition
  _  Subtraction
  *  Multiplication
  /  Division
Executable Statements

• Input/Output Operations Using the scanf Function

```
#include<stdio.h>

scanf("%lf", &radius);  // %lf double
scanf("%d", &year);     // %d int
scanf("%c%c", &Letter_1, &Letter_2);  // %c char
```
Executable Statements

• Using the `printf` function

```c
#include<stdio.h>

printf("The area is %f\n", area);
printf("Welcome, %c%c%c\n", Letter_1, Letter_2, Letter_3);
```
Formatting Numbers in Program. Output

- Formatting Values of Type Int

```c
Printf("Results: %3d meters = %4d ft. %2d in.\n", meters, feet, inches);
```

Results: 21 meters = 68 ft. 11 in.

- 3 columns for the first value
- 4 columns for the second value
- 2 columns for the third value
Formatting Numbers in Program. Output

• Formatting Values of Type double

\texttt{printf(\"The area is: \%6.4f\n\", area);}

- 6 columns for the real part
- 4 columns for the partial part

\begin{align*}
123456.1234 \\
3456.1200
\end{align*}
Formatting Numbers in Program. Output

Formatting Values of Type double

<table>
<thead>
<tr>
<th>value</th>
<th>Placeholder</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.12345</td>
<td>%5.2f</td>
<td>3.12</td>
</tr>
<tr>
<td>3.12345</td>
<td>%3.3f</td>
<td>3.123</td>
</tr>
<tr>
<td>3.12345</td>
<td>%8.5f</td>
<td>3.12345</td>
</tr>
</tbody>
</table>
Interactive Mode, Batch Mode, and Data Files

- **Interactive Mode:**
  User must enter the data

- **Batch Mode:**
  Program take its data from a file
  metric <mydata > myoutput

- **Data Files:**
  Program take its data from a file
Interactive Mode, Batch Mode, and Data Files

• # include<stdio.h>
  # define PI 3.14159

Int

main(void)
{

double radius, area, circum;
FILE *inp, /*Pointer to input file*/
*outp; /*Pointer to output file*/
Interactive Mode, Batch Mode, and Data Files

• /*Open the input and output files*/
  
  \[ \text{inp} = \text{fopen(“b:circle.dat”, “r”)}; \]
  \[ \text{outp} = \text{fopen(“b:circle.out”, “w”)}; \]

  \text{fscanf(} \text{inp, “%lf”, \&radius);} \\
  \text{fprintf(} \text{outp, “The radius is %.2f\n”, radius);} \\

  \text{area} = \pi \times \text{radius} \times \text{radius}; \\
  \text{circum} = 2 \times \pi \times \text{radius};
Interactive Mode, Batch Mode, and Data Files

- /*store the area and circum. in output file*/

```c
fprintf(outp, "The area is %.2f\n", area);
fprintf(outp, "The circumference is %.2f\n", circum);

close(inp);
close(outp);
return(0);
```
Common Programming Errors (Bugs)

- **Syntax Errors (Compilation Errors):**
  Detected by the compiler if a statement has a syntax error.
  The program cannot be translated.

**Run-Time Errors:**
Detected by the computer during execution time.
Such as division by zero.