CS1100
Computational Engineering

Introduction to C Programming Language

The C Programming Language

- An imperative general-purpose programming language
- Used extensively in the development of UNIX
- Extremely effective and expressive
- Not a “very high level” nor a “big” language
- Has compact syntax, modern control flow and data structures and a rich a set of operators
- Extensive collections of library functions

Origins of C

- Developed by Dennis M. Ritchie at Bell Labs
  - first implemented on DEC PDP-11 in 1972
- Based on two existing languages
  - BCPL and B languages
  - BCPL: Martin Richards, 1967 - systems programming
  - B: Ken Thomson, 1970 - early versions of UNIX

  The C Programming Language- Kernighan, Ritchie, 1978
- ANSI C: a standard adopted in 1990
  - unambiguous, machine-independent definition of C

  The C Programming Language (2nd edition)- Kernighan, Ritchie, 1988

Developing and Using a C program

A C program typically goes through six phases

1. Edit: the program is created and stored on disk
   - *Emacs* and *vi* are popular editors on Linux
   - usually part of IDE on Windows platforms
2. Preprocess: handles preprocessor directives
   - include other files, macro expansions etc
3. Compile: translates the program
   - into machine language code or object code
   - stores on disk

Other Phases

4. Link: combines
   - the object code of the program
   - object code of library functions and other functions

   creates an executable image with no “holes”
5. Load:
   - transfers the executable image to the memory
6. Execute:
   - computer carries out the instructions of the program

Programs = Solutions

- A program is a sequence of instructions
  - *This is from the perspective of the machine or the compiler!*

- A program is a (frozen) solution
  - *From the perspective of a human a program is a representation of a solution* devised by the human. *Once frozen (or written and compiled) it can be executed by the computer – much faster, and as many times as you want.*

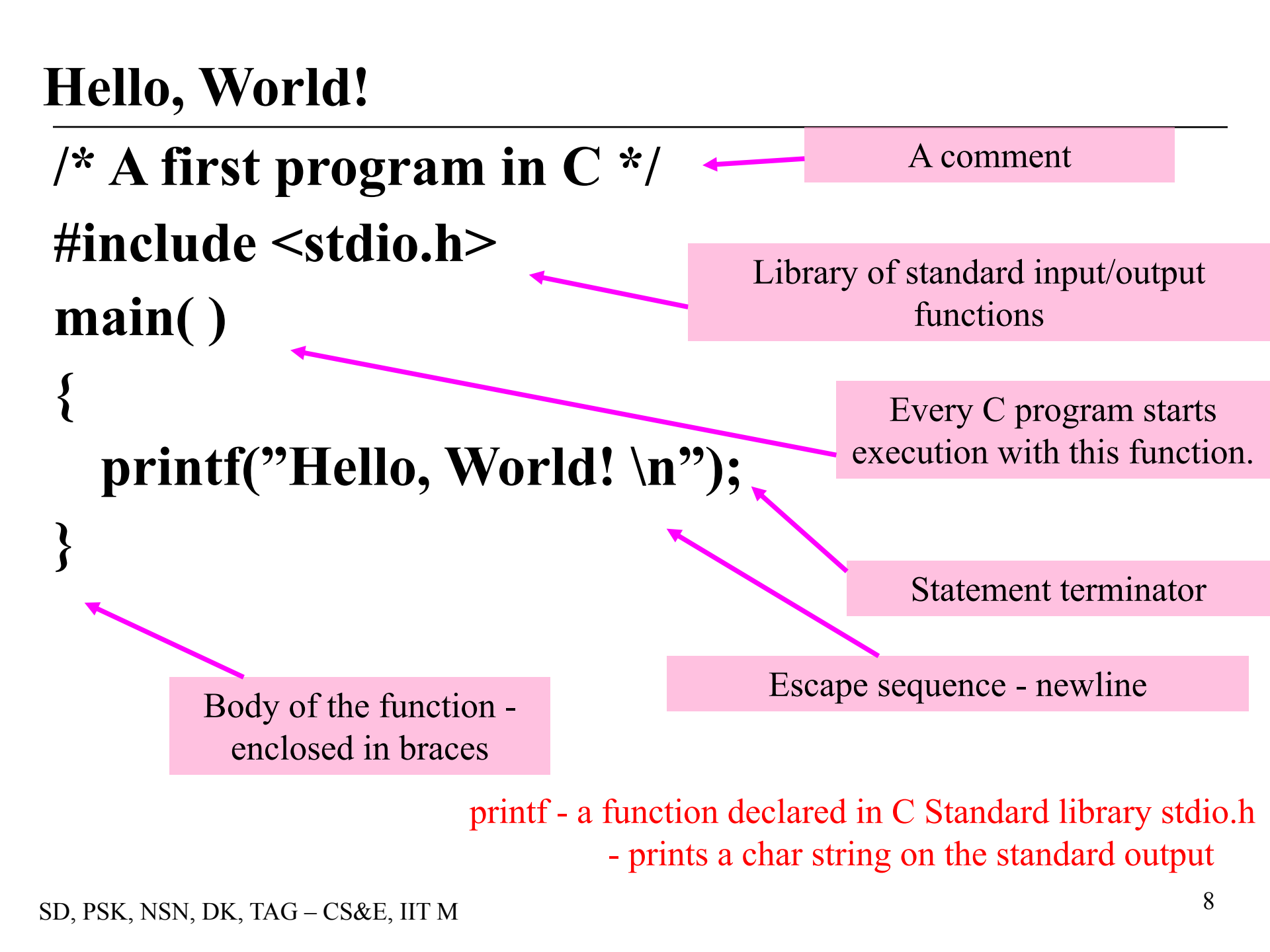
Programming = Problem Solving

- Software development involves the following
  - A study of the problem (requirements analysis)
  - A description of the solution (specification)
  - Devising the solution (design)
  - Writing the program (coding)
  - Testing

- The critical part is the solution design. One must work out the steps of solving the problem, analyze the steps, and then code them into a programming language.

Hello, World!

```
/* A first program in C */
#include <stdio.h>
main( )
{
   printf("Hello, World! \n");
}
```



printf - a function declared in C Standard library stdio.h
- prints a char string on the standard output

Programming Basics (emacs for programs)

- A variable – changes value during the execution of a program.
- A variable has a name, e.g. – name, value, speed, revsPerSec etc.
- Always referred to by its name
- Note: physical address changes from one run of the program to another.

Variables and Constants

- Names
  - made up of letters, digits and '_'
    - case sensitive: classSize and classsize are different
    - maximum size: 31 chars
  - first character must be a letter
  - choose meaningful and self-documenting names
    - MAX_PILLAR_RADIUS a constant
    - pillarRadius a variable
  - keywords are reserved
    - if, for, else, float, …

Assignments and Variables

- The value of a variable is modified due to an assignment
- The LHS is the variable to be modified and the RHS is the value to be assigned
- So RHS is evaluated first and then assignment performed
- E.g.: a = 1
  - a = c
  - a = MAX_PILLAR_RADIUS
  - a = a*b + d/e

Variable Declaration

- Need to declare variables
- A declaration: type variablename;
- Types: int, float, char, double, short, long, double double, long long
- E.g.: int x;
- Number of bytes of a variable depends on **type**.
- Assigning types helps write more correct programs.
  - Automatic type checking can catch errors like integer = char +char;

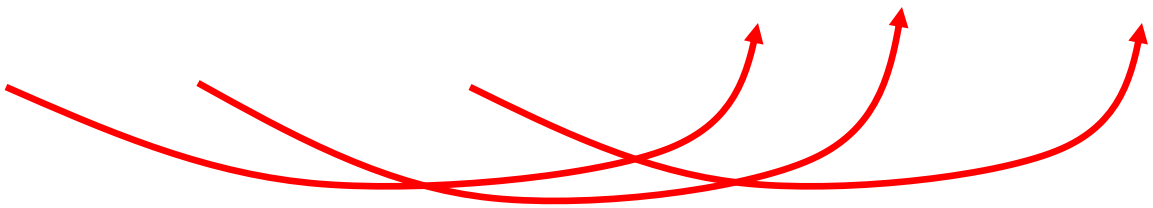
Variables need Declaration

Another simple C program

```
1  #include<stdio.h>
2  main()
3  {int int_size;
4    int chr_size, flt_size;
5    int_size = sizeof(int); chr_size =sizeof(char);
6    flt_size = sizeof(float);
7    printf("int, char, and float use %d %d and %d bytes\n",
8      int_size, chr_size, flt_size);
9  }
```

A special operator

A function from stdio.h



Exercise

- Type the above program using an editor of your choice (vim/emacs/gedit/vscode).
- Compile it using gcc.
- Run the *a.out* file

- If you already know C:
- Write a program that reads the coefficients of a quadratic and prints out its roots

Modifying Variables (*rm* with *–i option*)

- Each C program is a sequence of modification of variable values
- A modification can happen due to operations like +, -, /, *, etc.
- Also due to some functions/operators provided by the system like *sizeof*, *sin*, etc.
- Also due to some functions (another part of your program) created by the programmer

An Addition Program

```
#include <stdio.h>
main( )
{
   int operand1, operand2, sum;
   printf("Enter first operand\n");
   scanf("%d", &operand1);
   printf("Enter second operand\n");
   scanf("%d", &operand2);
   sum = operand1 + operand2;
   printf("The sum is %d \n", sum);
   return 0;
}
```

Declarations, must precede use

" %d " - conversion specifier

d - decimal

& - address of operand1

assignment

Returning a 0 is used to signify normal termination

Arithmetic Operators in C

Four basic operators

+ , – , *, /

addition, subtraction, multiplication and division

applicable to integers and floating point numbers

integer division - fractional part of result truncated

12/5 → 2, 5/9 → 0

modulus operator : %

x % *y* : gives the remainder after *x* is divided by *y*

applicable only for integers, not for float/double

Order of Evaluation (Operator Precedence)

first : parenthesized sub-expressions
- innermost first

second : * , / and % - left to right

third : + and – - left to right

$$a + b * c * d \,\%\, e - f / g$$
$$5 \quad 1 \quad 2 \quad 3 \quad 6 \quad 4$$

$$a + (((\, b * c \,) * d \,) \,\%\, e \,) - (f / g \,)$$

good practice – use parentheses rather than rely on precedence rules – better readability

Precedence – Another Example

- Value = a * (b + c) % 5 + x / (3 + p) – r – j
- Evaluation order –
  - (b + c) and (3 + p) : due to brackets
  - * and % and / have same precedence: a*(b + c) is evaluated first, then mod 5. Also, x/(3 + p).
  - Finally, the additions and subtractions are done from the left to right.
- Finally, the assignment of the RHS to LHS is done.
  - = is the operator that violates the left to right rule

Relational and Logical Operators

- A logical variable can have two values {true, false} or {1, 0}
- In C: int flag // 0 is false, any non-zero value is true
- Operators:

!	unary logical negation operator
< , <= , > , >=	comparison operators
= = , !=	equality and inequality
&&	logical AND operator
||	logical OR operator

- logical operators return true/false
- order of evaluation -- as given above

Logical AND

```
int x = 5, y = 2, z;

z = (x > 3) && (y < 4);
```

z will be assigned value of 1;

```
x = 2;
z = (x > 3) && (y < 4);
```

Z will be assigned 0.

Logical OR

int x = 5, y = 2, z;

z = (x > 3) || (y < 4);

z will be assigned value of 1;

x = 2;

z = (x > 3) && (y < 4);

Z will be assigned 1.

```
z = (x < 3) &&  ( (y > 4) && (a != 3));

z = x < 3 &&  y > 4 && a != 3;

z = (x < 3) &&  ( (y > 4) && !(a == 3));

z = (x < 3) &&  ( (y > 4) && !a);
```

Increment and Decrement Operators

- Unusual operators - prefix or postfix only to variables

  + + adds 1 to its operand

  – – subtracts 1 from its operand

- *n*++ increments *n* after its use
- ++*n* increments *n* before its use
- *n* = 4; *x* = *n*++; *y* = ++*n*;
- *x* is 4, *y* is 6 and *n* is 6 after the execution
- Avoid using these operators in expressions
- Stand-alond is Ok, e.g. n++;

Assignment Statement/Expression

- Form: *variable-name* = *expression*
  - E.g.: total = test1Marks + test2Marks + endSemMarks;
  - int *i*; float *x*;

    i = *x*; fractional part of *x* is dropped

    x = *i*; *i* is converted into a float
- Multiple assignment:

  $x = y = z = a + b;$

  $x = (y = (z = a + b));$
- Not Recommended Practice to use such complex expressions

Assignment Operators

- X = X *op* (expr) can be written as X *op*= expr
  - *op* : +, – , *, / , %
- E.g.: n= 1; *n* = *n* + 10; → *n* += 10;
- n= 1; n += 10; n -= 9; n *= 3;
- Value of n after above sequence of instructions is
  - n = 1
  - n = 11
  - n = 2
  - n = 6

Output Statement

printf (format-string, var_1, var_2, …, var_n);

format-string indicates:

how many variables to expect

type of the variables

how many columns to use for printing them

any character string to be printed

– sometimes this would be the only output

enclosed in double quotes

Examples - Output

int *x*; float *y*;

x = 20; *y* = – 16.7889;

printf("Value *x* = %*d* and value *y* = %9.3*f* \n", *x*, *y*);

'%*d*', '%9.3*f*' : conversion specifiers

'*d*', '*f*' : conversion characters

The output:

Value *x* = 20 and value *y* = □ □–16.789

□ - blank space (9 spaces)

General Form

General conversion specifier: %*w.p c*

optional

w : total width of the field,

p : precision (digits after decimal point)

c : conversion character

Conversion Characters:

d : signed decimal integer

u : unsigned decimal integer

o : unsigned octal value

x : unsigned hexadecimal value

f : real decimal in fractional notation

e : real decimal in exponent form

Input Statement

scanf(format-string, &var_1, &var_2, …, &var_n);

format-string:

types of data items to be stored in var_1, var_2, etc

enclosed in double quotes

Example: scanf("%*d*%*f*", &marks, &aveMarks);

data line: 16 14.75

For certain types of inputs (e.g., numeric) *scanf* skips spaces and scans more than one line to read the specified number of values

Conversion Specifiers for "scanf"

d - read a signed decimal integer

u - read an unsigned decimal integer

o - read an unsigned octal value

x - read an unsigned hexadecimal value

f - read a real decimal in fractional notation

e - read a real decimal in exponent form

c - read a single character

s - read a string of characters

Solving a Quadratic Equation (*rm –i* is safe)

```
#include<stdio.h>
#include<math.h>
Ax^2 + b x + c = 0
int main()
{ float coeff1, coeff2, coeff3;
   float root1, root2, discrim, denom;
   printf("Enter the 1st coefficient (a):");  /* prompt
   */
   scanf("%f",&coeff1);     /* read and store */
   printf("Enter the 2nd coefficient (b):");
```

Quadratic (continued) (*use vi* to create *files*)

```
printf("Enter the 3rd coefficient:");
scanf("%f", &coeff3);
        /* Now compute the roots*/
discrim = pow(coeff2, 2) – 4*coeff1*coeff3;
denom = 2*coeff1;
root1 = (– coeff2 + sqrt(discrim))/denom;
root2 = (– coeff2 – sqrt(discrim))/denom;
printf("the roots were %f, %f \n", root1, root2);
}
```

$b^2 - 4ac$

Exercise (see *http://www.gnu.org*)

Modify the program so that the quadratic is also output.

Summary: Variables are modified as the program runs.

Problem Solving withVariables

- Write a program that will take two degree 5 polynomials as input and print out their product.
- What are the inputs?
  - Coefficients from each polynomial. Six from each.
  - We need 12 *Input variables.*
- How many outputs are there?
  - We need 12 *Output variables*

Another Exercise (*www.howstuffworks.com*)

- Write a program that takes as input 3-digit numbers and prints them out in English.
- Example: 512 – Five Hundred and Twelve

Solve the problem first, identify input variables,

Output variables, intermediate variables.

What values are taken by the intermediate variables, how they are calculated from input values, and output variables.