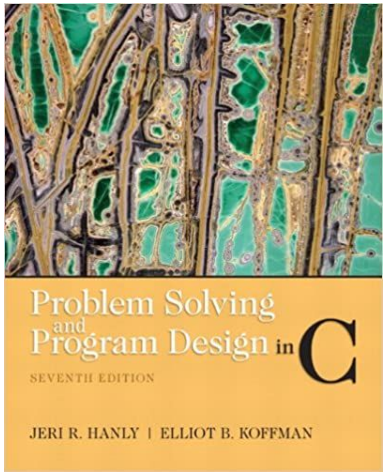
BIRZEIT UNIVERSITY



Faculty of Engineering and Technology
Department of Computer Science

Introduction to Computers and Programming (Comp 133)

References :
Book : Problem Solving and Program Design in C (7th Edition) 7th Edition
Slides : Dr. Radi Jarrar , Dr. Abdallah Karakra , Dr. Majdi Mafarja.

Overview of C

Chapter 2

Programming language

- A programming language is a set of rules that provides a way of telling a computer what operations to perform.

- C a high-level programming language developed in 1972 by Dennis Ritchie at AT&T Bell Laboratories.

Chapter 2

- Introduction

Levels of Programming Languages

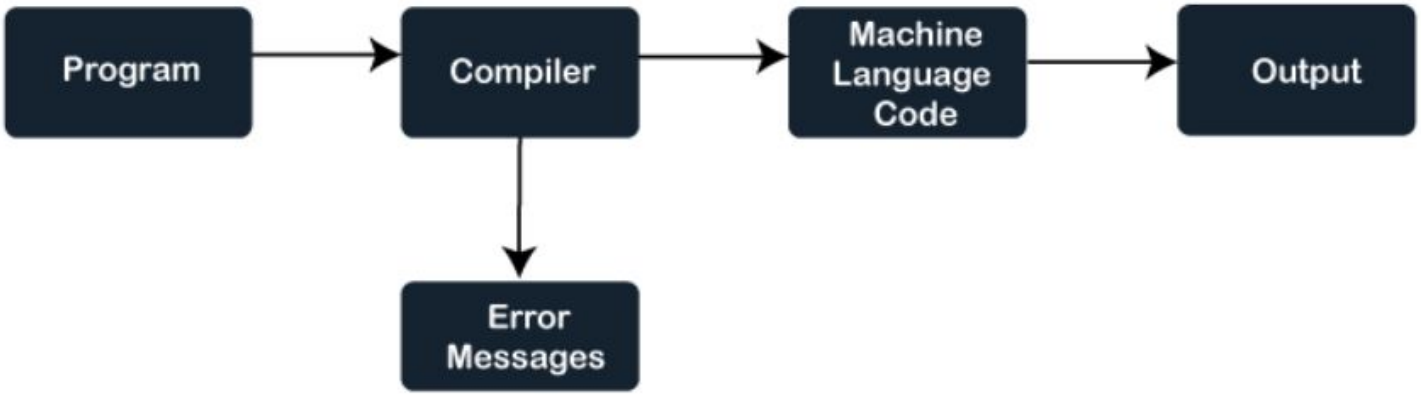
- Machine language
  - 0011001....011
- Assembly Language
  - mov ax, 0b00h
  - add ax, dx
- High Level Languages
- Fourth Generation Languages (4GL)
- Fifth Generation Languages (5GL

High Level Languages

- Procedure-oriented languages
  - FORTRAN , COBOL , Pascal , C
- Object-oriented languages
  - C++, C#, Java
- Event-driven languages
  - Visual Basic
- Declarative languages
  - Functional (Lisp, F#)

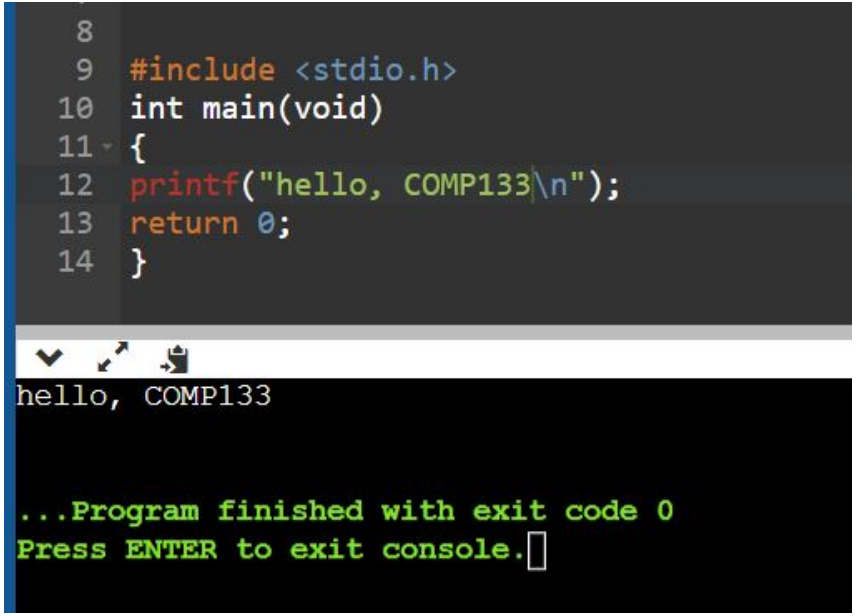
Compiler

- Compiler is a software programs that convert a high-level language into a machine language (0's and 1's binary form) that a computer can understand and perform tasks as per the program's instructions



C-program

```
#include <stdio.h>

int main(void)

{

printf("Hello, COMP 133\n");

return 0;

}
```

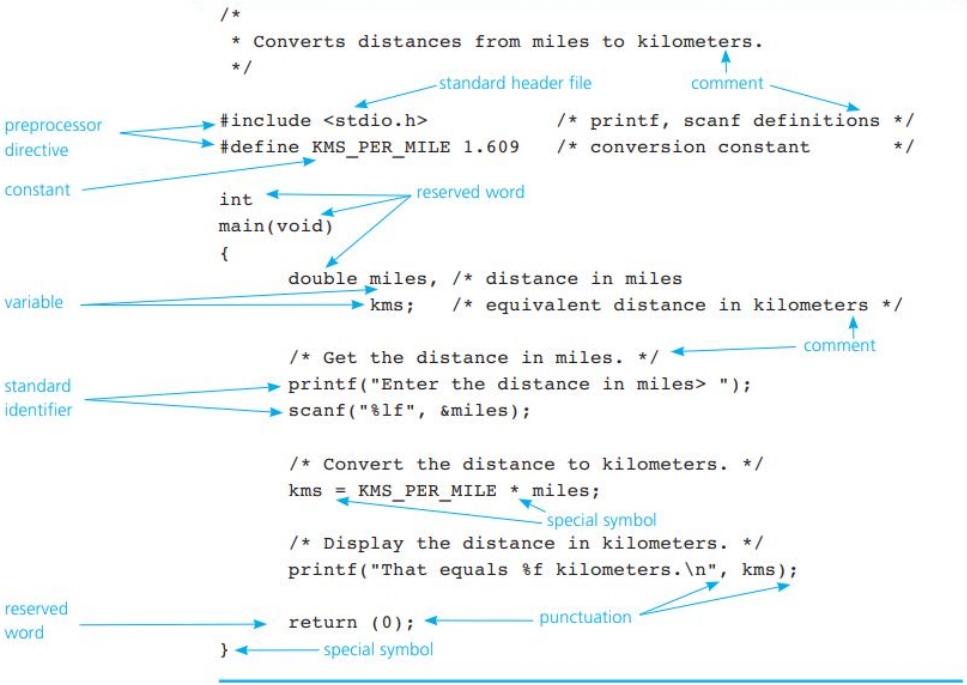


Chapter 2

- C Language Elements

Elements

FIGURE 2.1 C Language Elements in Miles-to-Kilometers Conversion Program




```
//C program for area of circle      Comment
#include <stdio.h> // standard header file (contains printf and scanf )
#define PI  3.141   //we use define for creating constant
int main()              // int, float , and return   (reserved words)
{
    float r, a;   //  r, a are variables
   printf("Please enter the radius: ");  //standard identifier
   scanf("%f", &r);    //standard identifier
   a = PI * r * r;      //  = , *,{, }  special symbols
   printf("%f\n", a); // standard identifier
   return 0;
}
```

Comments

- Each programing language has style for writing comments.
- In C-style comments :
  - /* comment */ -- More than one line
  - // comment
  - void main() {

```
/* This is how comments are implemented in C

to comment out a block of text */

// or like this for a single line comment

printf( "Hello World\n" ) ;

}
```

Preprocessor Directives

- **Preprocessor** a system program that modifies a C program prior to its compilation.
- **Library** a collection of useful functions and symbols that may be accessed by a program.
- #include
  - Gives a program access to a library
- <stdio.h>
  - Standard header file
  - Contains information about standard input and
  - Output functions such as scanf and printf

Preprocessor Directives

- **Preprocessor directive** a C program line beginning with # that provides an instruction to the preprocessor.
- #include
  - Notify the preprocessor that some names used in the program are found in <stdio.h> (E.g. scanf, printf)
- #define
  - Using only data values that never change should be given names.
  - #define MAX_LENGTH 100
- Constant macro
  - #define **PI** 3.141593

Reserved Words

- **Reserved word** is a word that has a special meaning in C.

TABLE 2.1 Reserved Words in Fig. 2.1

Reserved Word	Meaning
`int`	integer; indicates that the main function returns an integer value
`void`	indicates that the main function receives no data from the operating system
`double`	indicates that the memory cells store real numbers
`return`	returns control from the main function to the operating system

Standard Identifiers & User-Defined Identifiers

- **Standard Identifiers :** Like reserved words, standard identifiers have special meaning in C.
- **User-Defined Identifiers :** Our own identifiers to name memory cells that will hold data.

TABLE 2.3 Reserved Words and Identifiers in Fig. 2.1

Reserved Words	Standard Identifiers	User-Defined Identifiers
`int, void, double, return`	`printf, scanf`	`KMS_PER_MILE, main, miles, kms`

Variable Declarations and Data Types

- **Variable** a name associated with a memory cell whose value can change.
- **Variable declarations** statements that communicate to the compiler the names of variables in the program and the kind of information stored in each variable.
  - Syntax: type variable_name (or variable-list);

int , double, char….

Miles, Name , sum ….

Basic Data Types

- There are five basic data types char, int, float, double, and void. All other data types in C are based on these.

char	1 byte (8 bits) with range -128 to 127 **(‘a’, ‘6’, ‘*’, ……..)**
int	4 bytes with range -2,147,483,648 to 2,147,483,647
float	4 bytes with range 10^{-38} to 10^{38} with 7 digits of precision
double	8 bytes with range 10^{-308} to 10^{308} with 15 digits of precision
void	generic pointer, used to indicate no function parameters etc.

int i ; // a memory cell is reserved to hold an integer value

char a, b, ch ; //three character variables are defined

Basic Data Types

- Char : represent an individual character value • include a letter, adigit, a special symbol •E.g., ‘A’, ‘z’, ‘2’, ‘9’, ‘*’, ‘:’, ‘ ” ’, ‘ ‘
- A real number has an integral part and a fractional part that are separated by a decimal point

TABLE 2.4 Type double Constants (real numbers)

Valid double Constants	Invalid double Constants
`3.14159`	`150` (no decimal point)
`0.005`	`.12345e` (missing exponent)
`12345.0`	`15e-0.3` (`0.3` is invalid exponent)
`15.0e-04` (value is `0.0015`)	
`2.345e2` (value is `234.5`)	`12.5e.3` (`.3` is invalid exponent)
`1.15e-3` (value is `0.00115`)	`34,500.99` (comma is not allowed)
`12e+5` (value is `1200000.0`)	

Integer Types in C

TABLE 2.5 Integer Types in C

Type	Range in Typical Microprocessor Implementation
`short`	–32,767 .. 32,767
`unsigned short`	0 .. 65,535
`int`	–2,147,483,647 .. 2,147,483,647
`unsigned`	0 .. 4,294,967,295
`long`	–2,147,483,647 .. 2,147,483,647
`unsigned long`	0 .. 4,294,967,295

Floating-Point Types in C

TABLE 2.6 Floating-Point Types in C

Type	Approximate Range*	Significant Digits*
`float`	10^{-37} .. 10^{38}	6
`double`	10^{-307} .. 10^{308}	15
`long double`	10^{-4931} .. 10^{4932}	19

*In a typical microprocessor-based C implementation

ASCII Codes for Characters

TABLE 2.7 ASCII Codes for Characters

Character	ASCII Code
`' '`	32
`'*'`	42
`'A'`	65
`'B'`	66
`'Z'`	90
`'a'`	97
`'b'`	98
`'z'`	122
`'0'`	48
`'9'`	57

Variable Names

Names of variables and functions in C are called identifiers and are **case sensitive**. Rules defining a variable :

1. An identifier must consist only of **letters**, **digits**, and **underscores** (NO special characters like: +*&^%#$@ … etc).
2. An identifier cannot begin with a digit.
3. A C reserved word cannot be used as an identifier.
4. An identifier defined in a C standard library should not be redefined.

Reserved words : A word that has special meaning in C. (int, float, double, char , return ,…, etc.)

Variable Names

- letter_1, letter_2, inches, cent, CENT_PER_INCH, Hello, variable

TABLE 2.2 Invalid Identifiers

Invalid Identifier	Reason Invalid
`1Letter`	begins with a letter
`double`	reserved word
`int`	reserved word
`TWO*FOUR`	character * not allowed
`joe's`	character ' not allowed

Initialising Variables

- The identifier is not initialised to zero or to any other value automatically and so will contain random values unless specifically initialised before use.
- *Syntax :- type var-name = constant ;*

- E.g. char ch = 'a'; // Character constants are normally represented between single quotes.

  double d = 12.2323 ;

  int i, j = 20 ; /* note in this case i is not initialised */

Escape Sequences

- **Escape Sequences** special character constants preceded by the backslash character '\', and have special meanings in C.

- **\n** newline -> printf("Hello, COMP 133**\n**");
- **\t** tab
- **\b** backspace
- **\'** single quote
- **\"** double quote
- **\0** null character

Input/Output Operations and Functions

- The *printf* Function

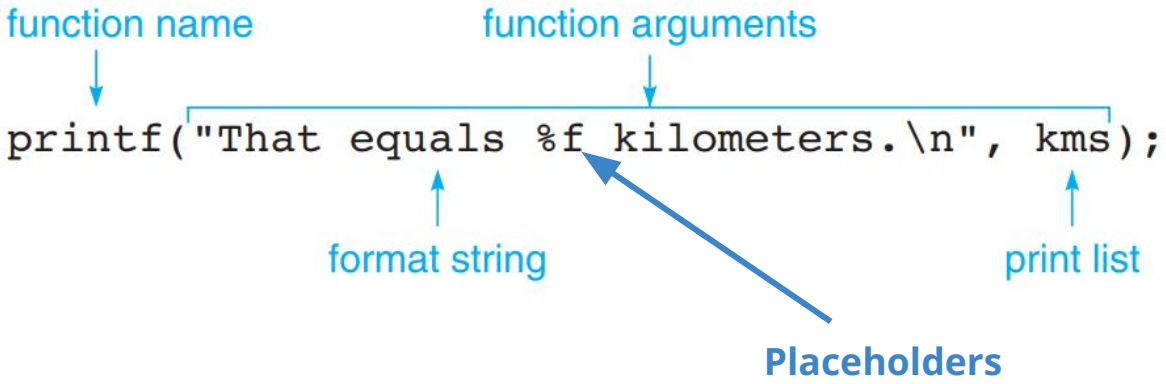


TABLE 2.8 Placeholders in Format Strings

Placeholder	Variable Type	Function Use
%c	char	printf/scanf
%d	int	printf/scanf
%f	double	printf
%lf	double	scanf

printf()

- printf (“The area is %f, a);
- scanf(“ %f ”,&r);
- printf (“the result is %d”, sum);
- scanf (“%lf”,& num);
- printf (“the number is %f”, num)
- printf("%d + %d", i, j) ;

Field Width Specifiers & format output

- **Field width specifiers** are used in the control string to format output. The number of columns used to display a value.
- Syntax : % [total width printed] [.decimal places printed][format specifier]
  - [ ] is optional arguments.

```c
#include <stdio.h>

int main()
{
   int meters = 21 , feet = 68 , inches = 11;
   printf("Results:  %3d meters = %4d ft. %2d in.\n", meters, feet, inches);

    return 0;
}
```

input

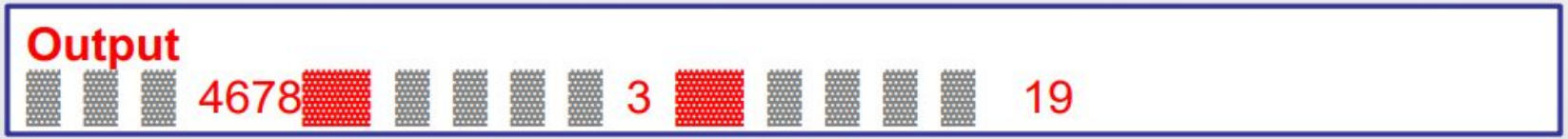
```
Results:   21 meters =   68 ft. 11 in.
```

Field Width Specifiers & format output

int x= 4678, y=3 , z=19

1. printf (“%d %d %d”, x,y,z)

2. printf (“%7d %5d %6d”, x,y,z)



Field Width Specifiers & format output

- float x=56.2757 y=2.3849 z=114.2 ;
  printf (“%8.3f%-7.2f%7.4f”,x,y,z);

  56.276 2.38 114.2000

- double a= 38.56, b= 201.117;
  printf("Is it%6.1f%9.4f", a, b);

  Is it 38.6 201.1170

- float x=333.256;
  printf("%0.2f",x); 333.26

Field Width Specifiers & format output

TABLE 2.14 Displaying 234 and –234 Using Different Placeholders

Value	Format	Displayed Output	Value	Format	Displayed Output
234	%4d	␣234	-234	%4d	-234
234	%5d	␣␣234	-234	%5d	␣-234
234	%6d	␣␣␣234	-234	%6d	␣␣-234
234	%1d	234	-234	%2d	-234

Field Width Specifiers & format output

TABLE 2.16 Formatting Type double Values

Value	Format	Displayed Output	Value	Format	Displayed Output
3.14159	%5.2f	▮3.14	3.14159	%4.2f	3.14
3.14159	%3.2f	3.14	3.14159	%5.1f	▮▮3.1
3.14159	%5.3f	3.142	3.14159	%8.5f	▮3.14159
.1234	%4.2f	0.12	-.006	%4.2f	-0.01
-.006	%8.3f	▮▮-0.006	-.006	%8.5f	-0.00600
-.006	%.3f	-0.006	-3.14159	%.4f	-3.1416

TABLE 2.15 Displaying x Using Format String Placeholder %6.2f

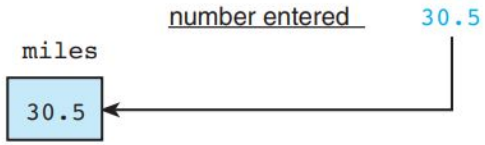
Value of x	Displayed Output	Value of x	Displayed Output
-99.42	-99.42	-25.554	-25.55
.123	▮▮0.12	99.999	100.00
-9.536	▮-9.54	999.4	999.40

Field Width Specifiers & format output

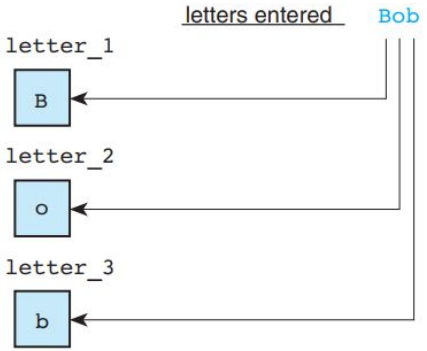
```c
int i = 15, j = -13 ;
float f = 13.3576 ;
printf( "%5d\n", i ) ;
/* prints "___15" where _ indicates a space character … here you have 3 spaces!! */

printf( "%-5d\n", i ) ;
/*prints 15___ where 15 is left justified */

printf( "%05d\n", i ) ;
/*prints 00015 0 (zero) causes a field to be padded using zeros rather than space characters */

printf( "%+d\n", j ) ;
/*prints: -13 + (plus sign) displays a plus sign preceding positive values and a minus preceding negative values, */

printf( "%6.2f\n", f ) ;
/* prints "_13.36" which has a total width of 6 and displays 2 decimal places */

printf( "%*.*f\n", 6,2,f ) ;
/* prints "_13.36" as above. Here * is used as replacement character for field widths */
```

scanf()

- ***scanf*** function get the data from standard input device, and stores it in the variable, in most cases the standard input device is the keyboard.
- scanf("%lf", &miles);



- scanf("%c%c%c", &letter_1, &letter_2, &letter_3);



%f	double	printf
%lf	double	scanf

Arithmetic Operations

TABLE 2.9 Arithmetic Operators

Arithmetic Operator	Meaning	Examples
+	addition	5 + 2 is 7 5.0 + 2.0 is 7.0
-	subtraction	5 - 2 is 3 5.0 - 2.0 is 3.0
*	multiplication	5 * 2 is 10 5.0 * 2.0 is 10.0
/	division	5.0 / 2.0 is 2.5 5 / 2 is 2
%	remainder	5 % 2 is 1

Arithmetic Operations

TABLE 2.9 Arithmetic Operators

Arithmetic Operator	Meaning	Examples
+	addition	5 + 2 is 7 5.0 + 2.0 is 7.0
-	subtraction	5 - 2 is 3 5.0 - 2.0 is 3.0
*	multiplication	5 * 2 is 10 5.0 * 2.0 is 10.0
/	division	5.0 / 2.0 is 2.5 5 / 2 is 2
%	remainder	5 % 2 is 1

Operators / and %

If the **/ and %** operators is used with a negative and a positive integer, the result may vary from one C implementation to another.

TABLE 2.10 Results of Integer Division

`3 / 15 = 0`	`18 / 3 = 6`
`15 / 3 = 5`	`16 / -3` varies
`16 / 3 = 5`	`0 / 4 = 0`
`17 / 3 = 5`	`4 / 0` is undefined

TABLE 2.11 Results of % Operation

`3 % 5 = 3`	`5 % 3 = 2`
`4 % 5 = 4`	`5 % 4 = 1`
`5 % 5 = 0`	`15 % 5 = 0`
`6 % 5 = 1`	`15 % 6 = 3`
`7 % 5 = 2`	`15 % -7` varies
`8 % 5 = 3`	`15 % 0` is undefined

Operators / and %

- **int / int = int . E.g : 12/3= 4 , 9/8=1**
- **Int / float = float. E.g : 9/8.0=1.125000**
- **Float / int = float. E.g : 9.0/8=1.125000**
- **Float / float = float E.g : 9.0/8.0=1.125000**
- **Undefined :** when divide by zero **18/0 , 16%0**

```
int main()
{
   double x,y;
   x=15/2;
   y=15/2.0;
   printf("x=%f \ny= %f", x,y);  // Output ????
   return 0;
}
```

Assignment Operator

- int x ; x = 20 ; int y; y= 30; x=y ; (x and y same data type).
- **Multiple assignments :** x = y = z = 100 ;
- **Type Conversions** : the value of the right-value of an assignment is converted to the type of the left-value. This may sometimes yield compiler warnings if information is lost in the conversion.
  - **Int x; double y=2.345;**
  - **x=y ; ?**

Type Conversions

```
int x ;
char ch ;
float f ;

ch = x ;
 /* ch is assigned lower 8 bits of x,the remaining bits are discarded,
 so we have a possible information loss */

x = f ;
/* x is assigned non fractional part of f only
within int range, information loss possible */

f = x ; /* value of x is converted to floating point */
```

Type conversion through casts

- **Type cast** : converting an expression to a different type by writing the desired type in parentheses in front of the expression.
  - n = **(int)**(9 * 0.5);
  - The value of n is 4

```
main() {

    int sum = 17, count = 5;
    double mean;

    mean = (double) sum / count;
    printf("Value of mean : %f\n", mean );
}
```

Value of mean : 3.400000

Type conversion through casts

```
void main() {

    int  i = 15;
    char c = 'c'; /* ascii value is 99 */
    float sum;

    sum = i + c;
    printf("Value of sum : %f\n", sum );
}
```

Value of sum:
114.000000

Chapter 2

- Arithmetic Expressions

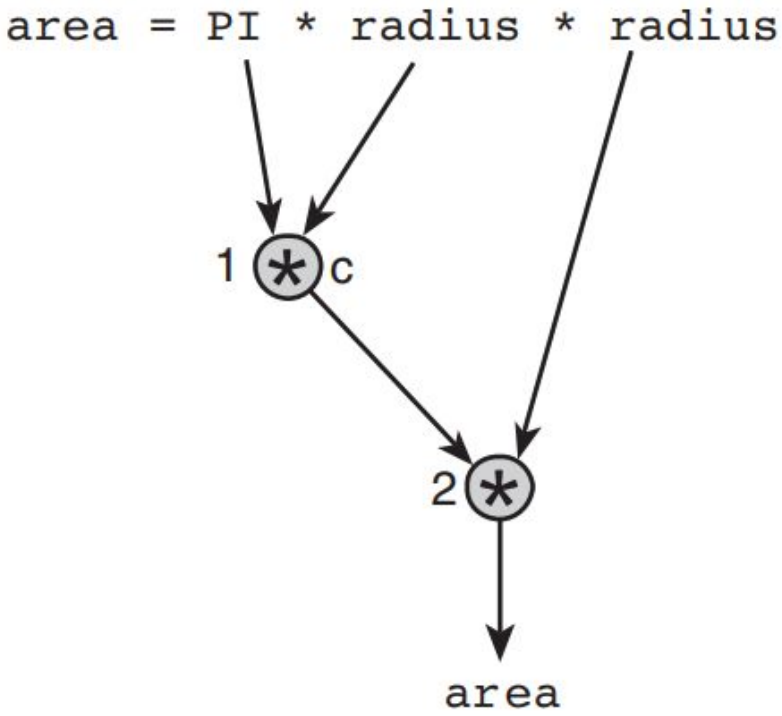
Arithmetic Expressions

- Precedence Rules:

1	**()**
2	*** / %**
3	**+ -**

Arithmetic Expressions

- Evaluation Tree for **area = PI * radius * radius;**



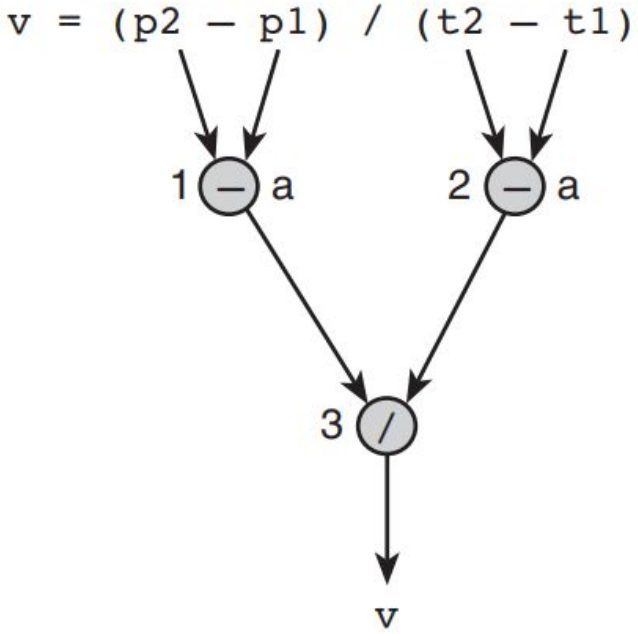
Arithmetic Expressions

- Step-by-Step Expression Evaluation . **PI= 3.14159 , radius=2.0**

```
area     =      PI     *    radius    *    radius
            3.14159          2.0            2.0
                  6.28318
                                  12.56636
```

Arithmetic Expressions

- Evaluation Tree and Evaluation for v = (p2 − p1) / (t2 − t1);

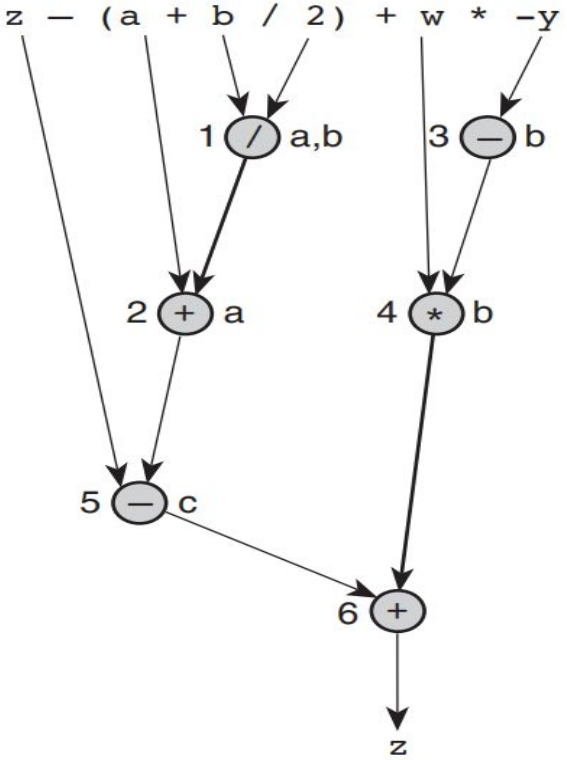


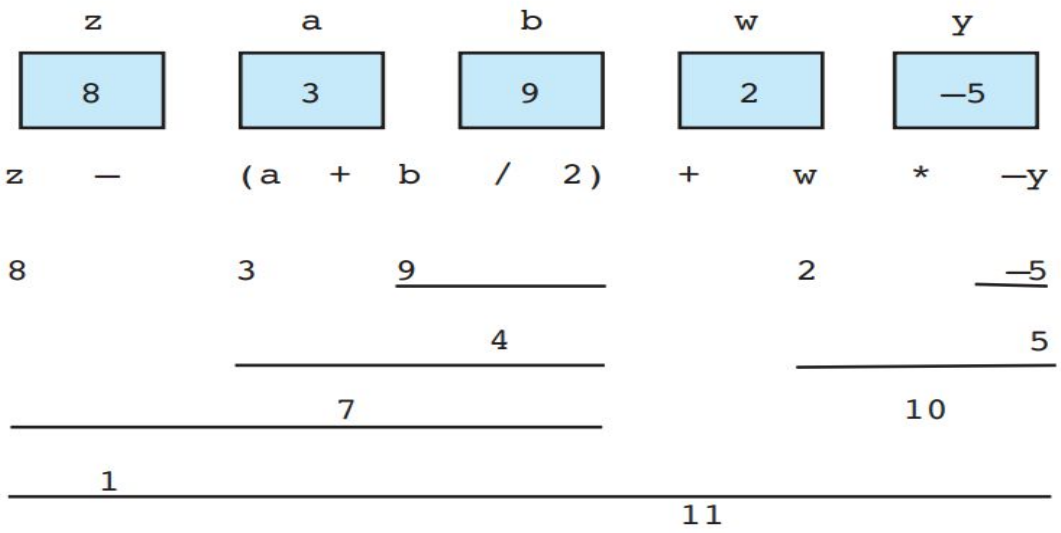
p1	p2	t1	t2
4.5	9.0	0.0	60.0

```
v  =   (p2  –  p1)   /   (t2  –  t1)
       9.0     4.5       60.0     0.0
          4.5                  60.0
                0.075
```

Arithmetic Expressions

- Evaluation Tree and Evaluation for z − (a + b / 2) + w * −y





Arithmetic Expressions

TABLE 2.13 Mathematical Formulas as C Expressions

Mathematical Formula	C Expression
1. $b^2 - 4ac$	`b * b - 4 * a * c`
2. $a + b - c$	`a + b - c`
3. $\frac{a + b}{c + d}$	`(a + b) / (c + d)`
4. $\frac{1}{1 + x^2}$	`1 / (1 + x * x)`
5. $a \times -(b + c)$	`a * -(b + c)`

Arithmetic Expressions

- Write a complete C program that prompts the user to enter the radius of a circle and displays the circumference. Circumference=2 πr

```c
#include <stdio.h>
#define PI 3.14159
int main(void)
 {
 double radius, circum;
 printf("Please enter radius of circle> ");
 scanf("%lf", &radius);
 circum = 2 * PI * radius;
 printf("The circumference is %.2f.\n", circum);
 return 0;
 }
```

Chapter 2

- Common programming errors

Common programming errors

- **Syntax Errors** - a violation of the C grammar rules, detected during program translation (compilation)
  - Missing semicolon
  - Undeclared variable
  - Last comment is not closed because of blank in */ close-comment sequence
- **Logic Errors** - an error caused by following an incorrect algorithm
  - Sum = x-y (minus instead of plus)
- **Run-Time Errors** – an attempt to perform an invalid operation, detected during program execution.
  - Result= x / 0 (undefined)
- **Debugging :** removing errors from a program.

Syntax Errors example

```
268 int
269 main(void)
270 {
271         double kms
272
273         /* Get the distance in miles. */
274         printf("Enter the distance in miles> ");
***** Semicolon added at the end of the previous source line
275         scanf("%lf", &miles);
***** Identifier "miles" is not declared within this scope
***** Invalid operand of address-of operator
276
277         /* Convert the distance to kilometers. */
278         kms = KMS_PER_MILE * miles;
***** Identifier "miles" is not declared within this scope
279
280         /* Display the distance in kilometers. * /
281         printf("That equals %f kilometers.\n", kms);
282
283         return (0);
284 }
***** Unexpected end-of-file encountered in a comment
***** "}" inserted before end-of-file
```

Run-Time Error example

```
263 int
264 main(void)
265 {
266         int     first, second;
267         double temp, ans;
268
269         printf("Enter two integers> ");
270         scanf("%d%d", &first, &second);
271         temp = second / first;
272         ans = first / temp;
273         printf("The result is %.3f\n", ans);
274
275         return (0);
276 }

Enter two integers> 14 3
Arithmetic fault, divide by zero at line 272 of routine main
```

Logic Errors example

```
3.  int
4.  main(void)
5.  {
6.      int    first, second, sum;
7.
8.      printf("Enter two integers> ");
9.      scanf("%d%d", first, second); /* ERROR!! should be &first, &second */
10.     sum = first + second;
11.     printf("%d + %d = %d\n", first, second, sum);
12.
13.     return (0);
14. }

Enter two integers> 14    3
5971289 + 5971297 = 11942586
```

Chapter 2

- File I/O

File I/O

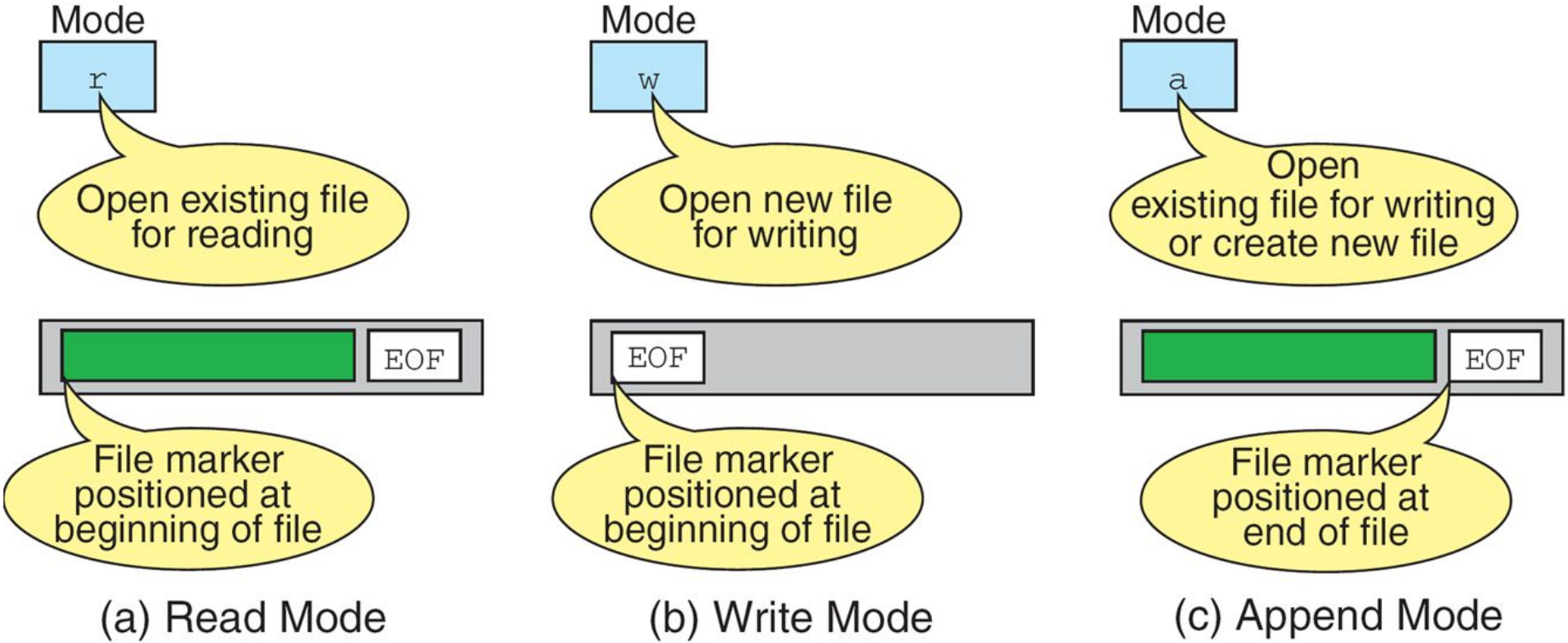
- Declare a file pointer variables.
  - **FILE *input;**
  - **FILE *output;**
- Then call a function that opens the files
  - **input = fopen(“File_Input.txt”, “r”);**
  - **output = fopen(“File_Output.txt”, “w”);**
- Now we can read or write from / to file.
  - **fscanf(input, “%d”, &x);**
  - **fprintf(output, “%d\n”, x);**
- Last step close the file.
  - **fclose(input)**
  - **fclose(output)**

File Open Modes

Mode	Meaning
r	Open text file in read mode • If file exists, the marker is positioned at beginning. • If file doesn't exist, error returned.
w	Open text file in write mode • If file exists, it is erased. • If file doesn't exist, it is created.
a	Open text file in append mode • If file exists, the marker is positioned at end. • If file doesn't exist, it is created.

https://www.slideserve.com/calais/file-handling-in-c-powerpoint-ppt-presentation

File Open Modes



https://www.slideserve.com/calais/file-handling-in-c-powerpoint-ppt-presentation

End of File

- There are a number of ways to test for the end-of-file condition.
  - Using function feof()

```
if (feof(input))
printf("\n End of file reached.");
```

Another ways to use macro EOF.

```
If(input==EOF)
printf ("End-of-file encountered.\n”) ;
```

EOF is a constant variable in C, and its value is -1.

Reading from Files - Example

```c
#include <stdio.h>

void main() {
    FILE *input;
    input = fopen("test.txt", "r");
    double x;
    int status = fscanf(input, "%lf", &x);
    while(status != EOF) {
        printf("%f\n", x);
        status = fscanf(input, "%lf", &x);
    }
    fclose(input);
}
```

From : Dima Taji – Birzeit University – COMP1310 – First Semester 2021/2022

Writing to Files – Example

- Write a program that reads numbers from the user.
- Every time a new number is entered, calculate the square root of that number.
- Prints the number and its square root to a file.

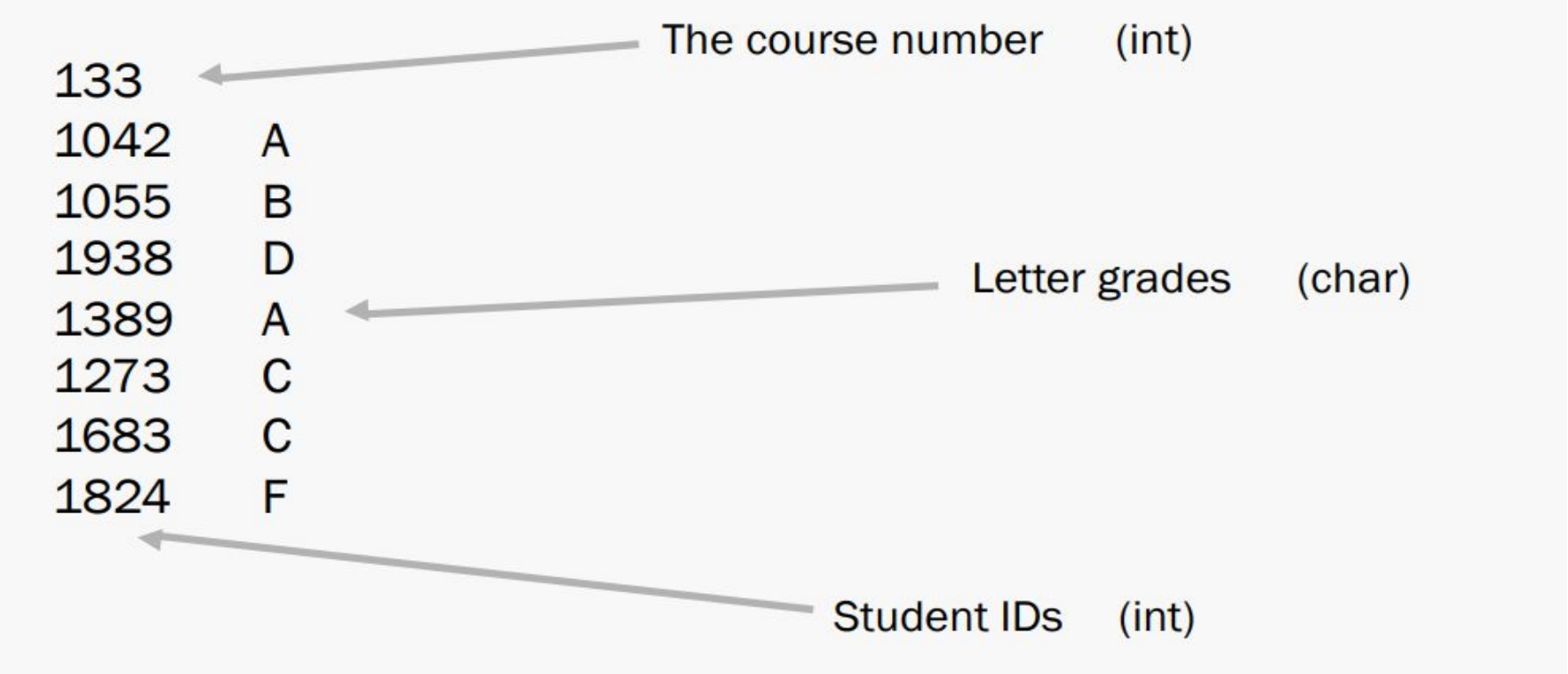
From : Dima Taji – Birzeit University – COMP1310 – First Semester 2021/2022

Writing to Files – Example

```c
#include <stdio.h>
#include <math.h>


void main() {
    FILE *output;
    output = fopen("test.txt", "w");
    double x;
    printf("Enter a number: ");
    scanf("%lf", &x);
    while(x != -1) {
        double root = sqrt(x);
        fprintf(output, "%f\t%f\n", x, root);
        printf("Enter a number: ");
        scanf("%lf", &x);
    }

    fclose(output);
}
```

From : Dima Taji – Birzeit University – COMP1310 – First Semester 2021/2022

Reading Different Data Types from Files



From : Dima Taji – Birzeit University – COMP1310 – First Semester 2021/2022

Reading Different Data Types from Files

```c
#include <stdio.h>

void main() {
    FILE *input;
    input = fopen("grades.txt", "r");

    FILE *output;
    output = fopen("grades_report.txt", "w");

    int course;
    int status = fscanf(input, "%d", &course);

    int As = 0, Bs = 0, Cs = 0, Ds = 0, Fs = 0;

    int ID;
    char grade;

    status = fscanf(input, "%d\t%c", &ID, &grade);

    while(status != EOF) {
        switch(grade) {
            case 'A':
                As += 1;
                break;
            case 'B':
                Bs += 1;
                break;
            case 'C':
                Cs += 1;
                break;
            case 'D':
                Ds += 1;
                break;
            case 'F':
                Fs += 1;
                break;
        }
        status = fscanf(input, "%d\t%c", &ID, &grade);
    }
```

From : Dima Taji – Birzeit University – COMP1310 – First Semester 2021/2022

Reading Different Data Types from Files

```
41          fprintf(output, "Grade report for Comp%d\n", course);
42          fprintf(output, "A\t%d\n", As);
43          fprintf(output, "B\t%d\n", Bs);
44          fprintf(output, "C\t%d\n", Cs);
45          fprintf(output, "D\t%d\n", Ds);
46          fprintf(output, "F\t%d\n", Fs);
47
48          fclose(input);
49          fclose(output);
50      }
```

From : Dima Taji – Birzeit University – COMP1310 – First Semester 2021/2022

Thank You.

جامعة بيرزيت
BIRZEIT UNIVERSITY