Outline

- C Statements and Block of Statement
- Comments in C
- Operators in C
- B-E-DM-AS Rule
- Examples : Basic operations
- Program examples
- Good Programming practice

Variables in C (quick recap)

- Variables are
  - Named blocks of memory
  - Valid identifier.
- Variable have two properties in syntax:
  - Name — a unique identifier
  - Type — what kind of value is stored.
- It is identifier, that
  - Value may change during the program execution.
- Every variable stored in the computer's memory
  - Has a name, a value and a type.

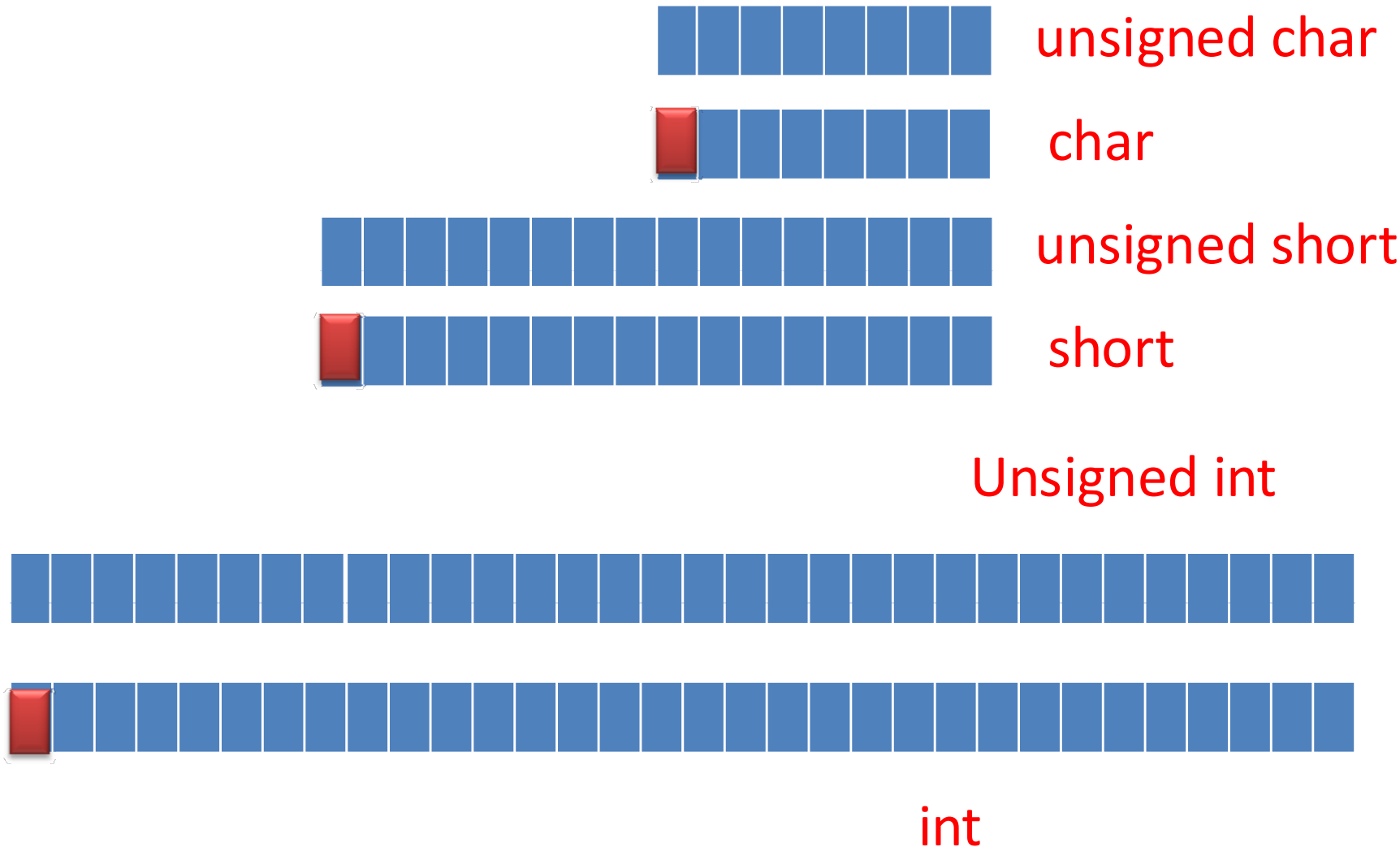
Variable Naming Conventions (quick recap)

- C programmers generally agree on the following **conventions** for naming variables.
  - Begin variable names with lowercase letters
  - Use meaningful identifiers
  - Separate "words" within identifiers with underscores or mixed upper and lower case.
  - Examples: `surfaceArea`, `surface_Area`, `surface_area`
  - **Be consistent!**

Numeric Data Type (quick recap)

- **char, short, int, long int**
  - `char : 8 bit number (1 byte=1B)`
  - `short: 16 bit number (2 byte)`
  - `int : 32 bit number (4B)`
  - `long int : 64 bit number (8B)`
- **float, double, long double**
  - `float : 32 bit number (4B)`
  - `double : 64 bit number (8B)`
  - `long double : 128 bit number (16B)`

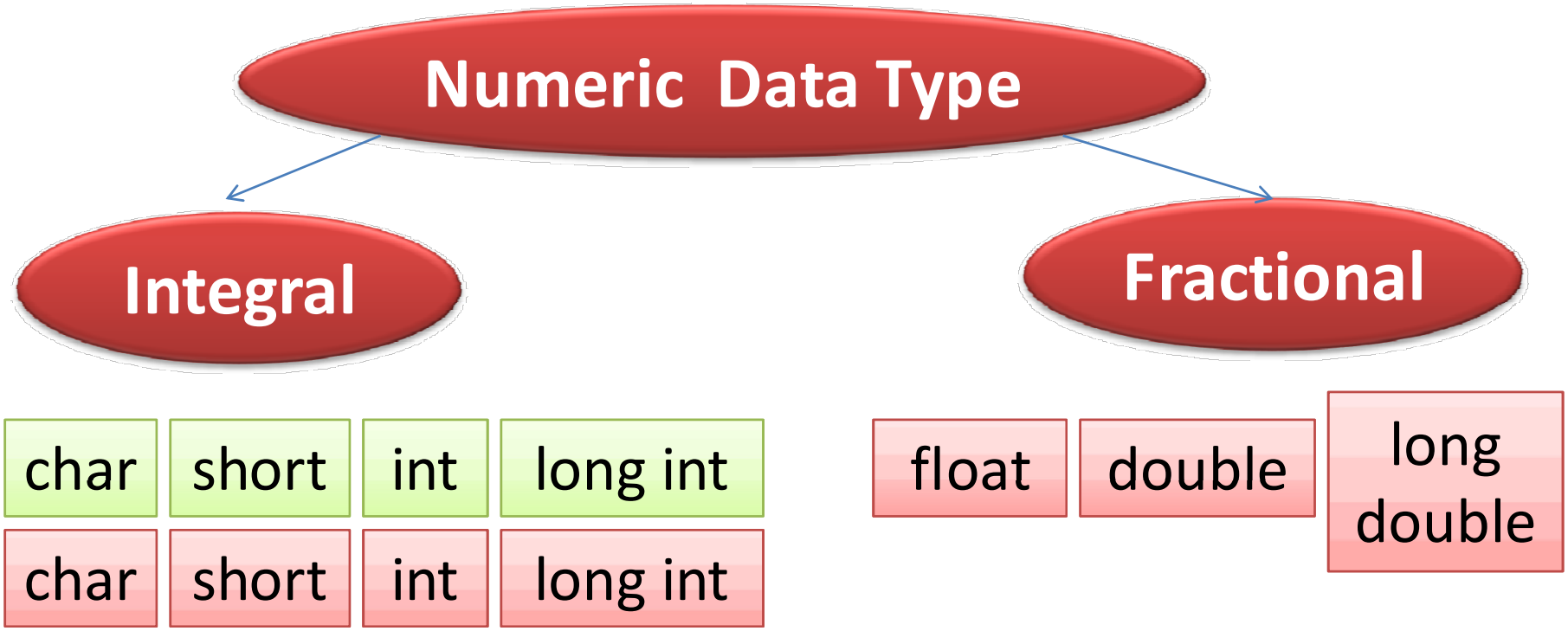
Numeric Data Type (quick recap)



Testing size of Numeric Data

```
#include<stdio.h>
int main(){
  printf("size of char %d\n", sizeof(char)); //1
  printf("size of short %d\n",sizeof(short)); //2
  printf("size of int %d\n",sizeof(int)); //4
  printf("size of long int %d\n",sizeof(long int)); //8

  printf("size of float \n",sizeof(float)); //4
  printf("size of double %d\n",sizeof(double));//8
  printf("size of long double  %d\n",
                                  sizeof(long double));//16
  return 0;
}
```

Numeric Data Type (quick recap)



- **char, short, int, long int**
  - **Signed and unsigned**
- **float, double, long double**

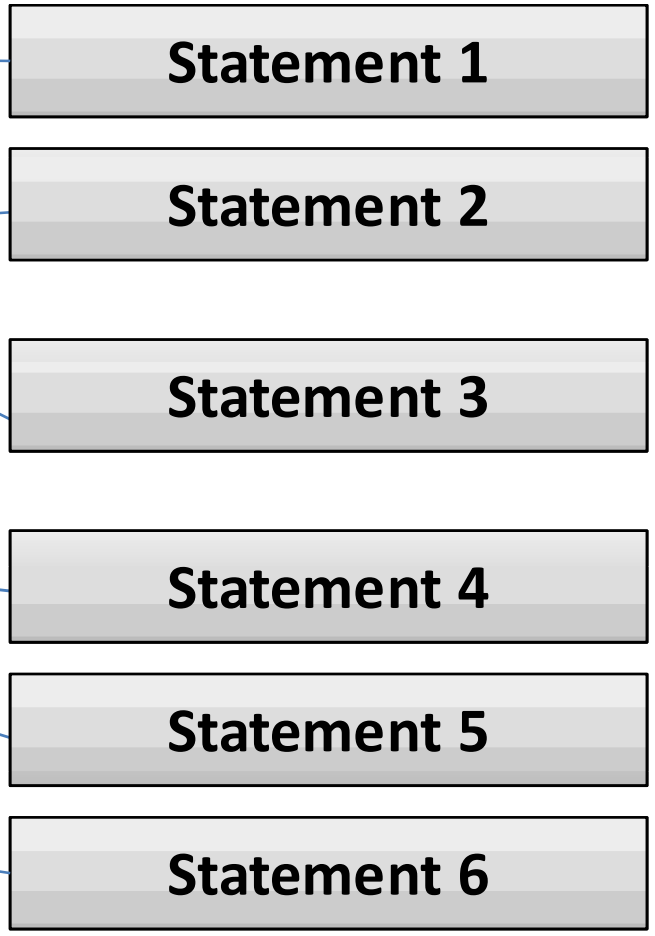
C Statements

- Statements are terminated with a semicolon and that is ';'
- e.g:

```
char acharacter;
int i, j = 18, k = -20;
printf("Initially, given
        j = 18 and k = -20\n");
```

C Programming : Sum of A and B

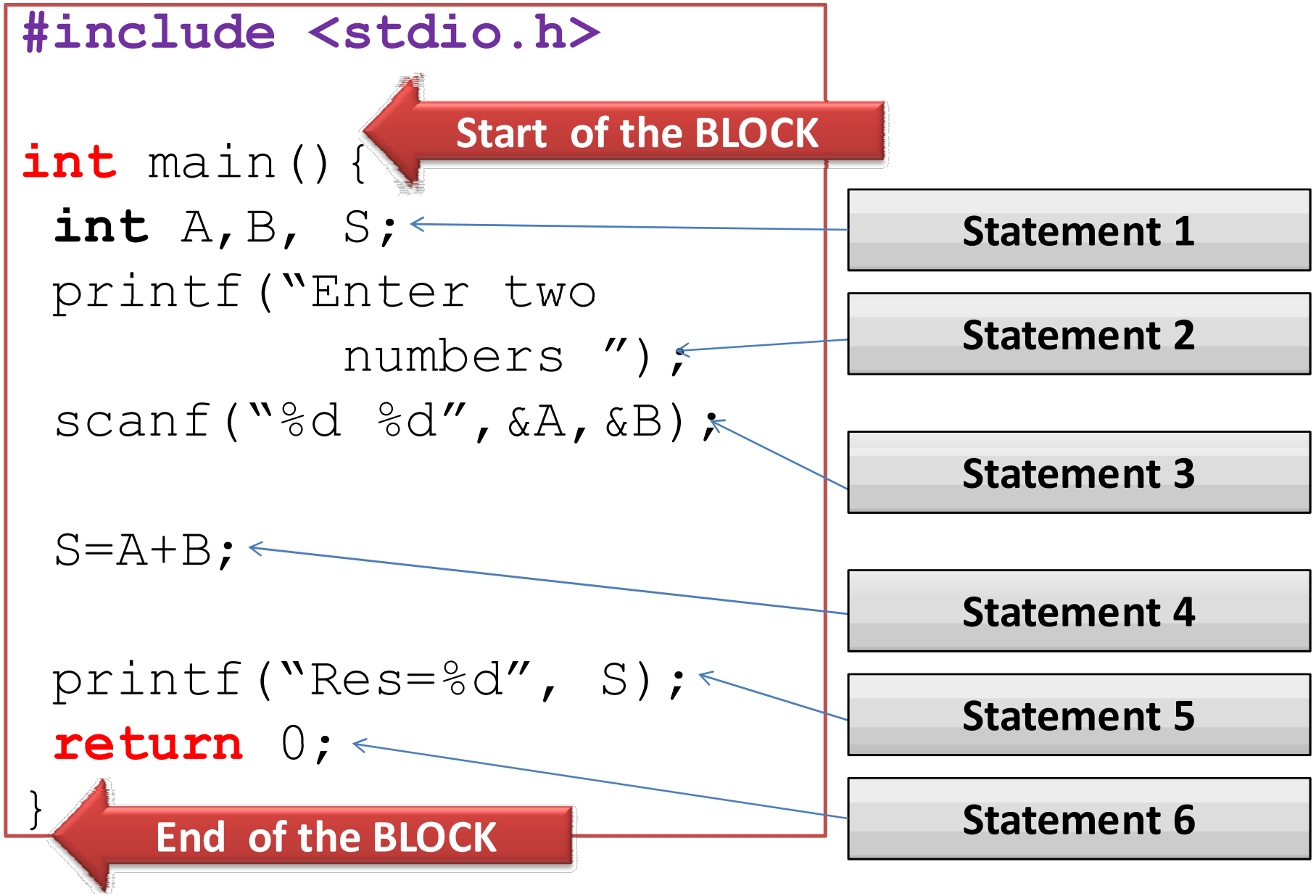
```
#include <stdio.h>

int main(){
 int A,B, S;
 printf("Enter two
            numbers ");
 scanf("%d %d",&A,&B);

 S=A+B;

 printf("Res=%d", S);
 return 0;
}
```



C: Block of Statements

- Group of statements (compound statement) are enclosed by curly braces: { and }.
- Mark the start and the end of code block.

C Programming : Sum of A and B



Comments in C

- Single line of comment: `// comment here`
- More than single line of comment or expanded: `/* comment(s) here */`

```
#include <stdio.h> // for printf()
/* main() function, where program
      execution starts */
int main(){
   /* declares variable and
               initializes it*/
      int i = 8;
      printf("value of i=%d\n",i);
      return 0;
}
```

Declaring Variables

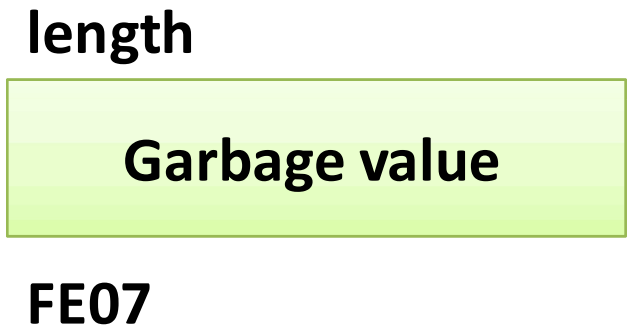
- Before using a variable, you must give the compiler some information about the variable; i.e., you must **declare** it.
- The **declaration statement** includes the **data type** of the variable.
- Examples of variable declarations:

```
int   length ;
float  area ;
```

Declaring Variables

- When we declare a variable
  - Space is set aside in memory to hold a value of the specified data type
  - That space is associated with the variable name
  - That space is associated with a unique **address**
- Visualization of the declaration

```
int  length ;
```



Using Variables: Initialization

- Variables may be be given initial values, or **initialized**, when declared. Examples:

```
int length=7;
```

length

7

```
float diameter=5.9;
```

diameter

5.9

```
char initial ='A';
```

initial

'A'

Using Variables: Initialization

- Do not "hide" the initialization
  - Put initialized variables on a separate line
  - A comment is always a good idea
  - Example:

```
int height;  /* rectangle height */
int width=6; /* rectangle width  */
int area;    /* rectangle area   */
```

NOT int height, width = 6, area ;

Using Variables: Assignment

- Variables may have values assigned to them through the use of an **assignment statement**.
  - Uses the **assignment operator =**
- This operator (=) does not denote equality.
- It assigns the value of the righthand side of the statement (the **expression**) to the variable on the lefthand side.
- **Only single variables may appear on the lefthand side of the assignment operator.**
- Examples:

```
diameter = 5.9 ;
area = length * width ;
```

Using Variables: Assignment

- variable= <const|Expression>
- <Expresion> can be simple or complex expression

```
area = length * width ;
```

Arithmetic Operators in C

Name	Operator	Example
Addition	+	num1 + num2
Subtraction	-	initial - spent
Multiplication	*	fathoms * 6
Division	/	sum / count
Modulus	%	m % n

Division

- Integer division
  - If both operands of a division expression are integers,
  - you will get an integer answer.
- The fractional portion is thrown away.
- Examples :

```
17  /  5  =  3
 4  /  3  =  1
35  /  9  =  3
```

Division : float

- Division where at least one operand is a floating point number will produce a floating point answer.
- Examples:

```
17.0  /  5     =  3.4
 4    /  3.2   =  1.25
35.2  /  9.1   =  3.86813
```

- What happens? The integer operand is temporarily converted to a floating point, then the division is performed.

Division By Zero

- Division by zero is mathematically undefined.
- If you allow division by zero in a program, it will cause a **fatal error**.
- Your program will terminate execution and give an error message.
- **Non-fatal errors** do not cause program termination, just produce incorrect results.

Modulus

- The expression **m % n** yields the integer remainder after **m** is divided by **n**.
- Modulus is an integer operation -- both operands MUST be integers.
- Examples : 17 % 5 = 2

  6 % 3 = 0

  9 % 2 = 1

  5 % 8 = 5

Uses for Modulus

- Used to determine if an integer value is even or odd

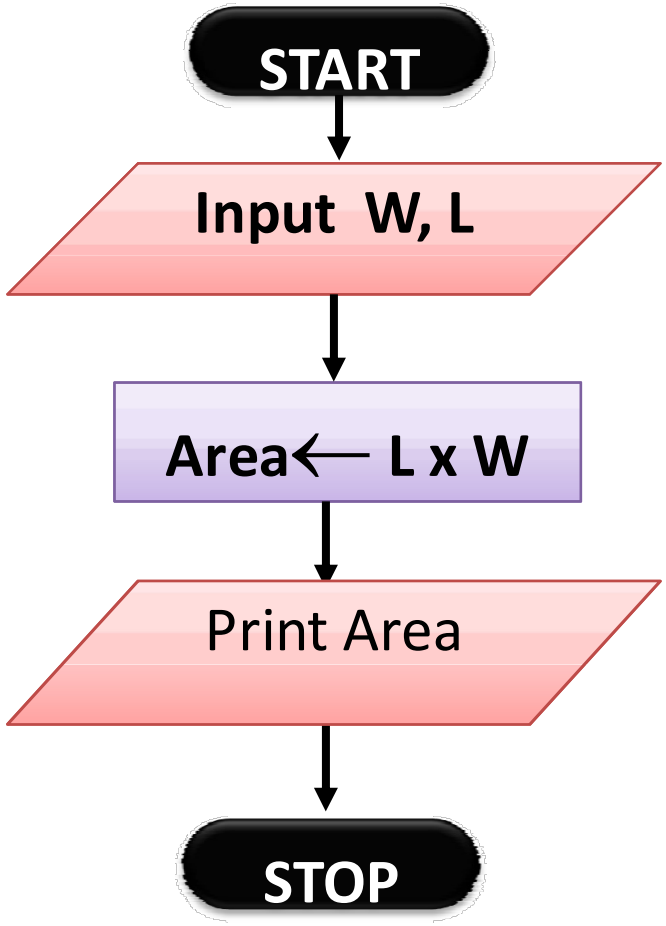
  5 % 2 = 1 odd 4 % 2 = 0 even

  If you take the modulus by 2 of an integer, a result of 1 means the number is odd and a result of 0 means the number is even

C Example 1: Area of Rectangle

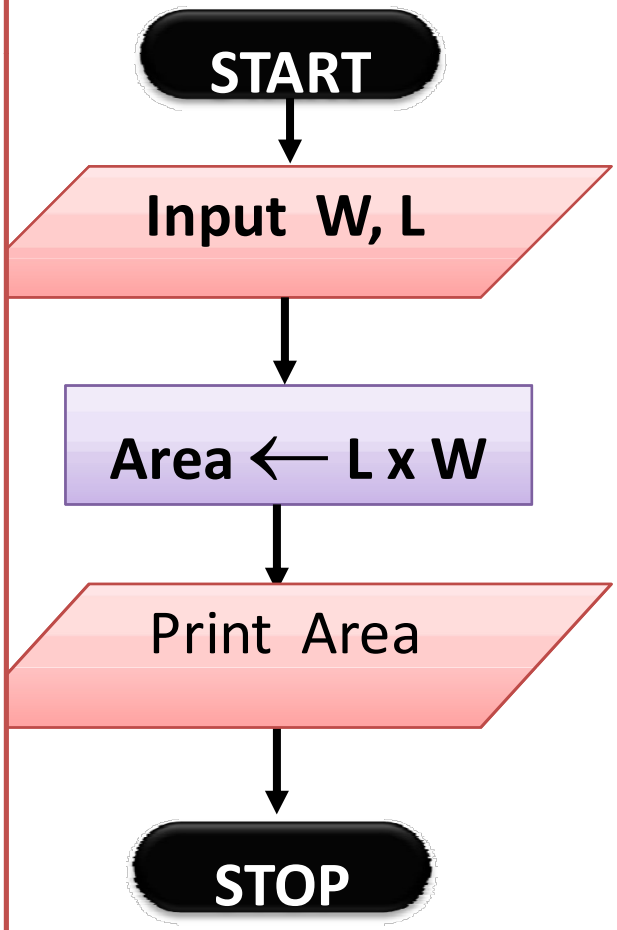
Read the two sides of a rectangle and calculate its area.

- Step 1: Input W,L
- Step 2: Area ← L x W
- Step 3: Print Area



C Example 1: Area of Rectangle

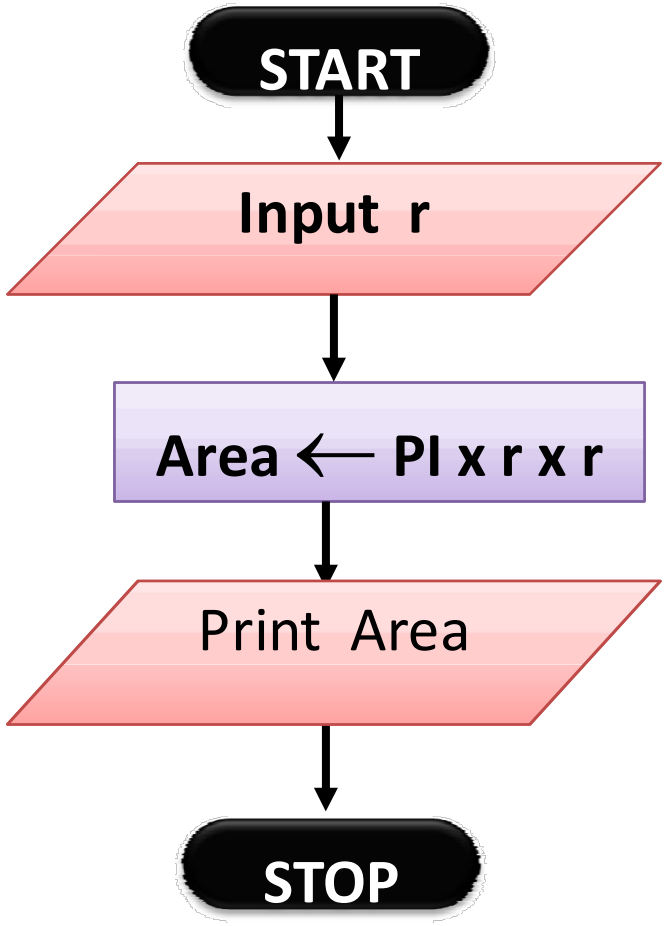
```c
#include <stdio.h>

int main(){
 int L,W, Area;
 printf("Enter L & B ");
 scanf("%d %d",&L,&B);

 Area=L*B;

 printf("Area=%d",Area);
 return 0;
}
```



C Example 2: Area of Circle

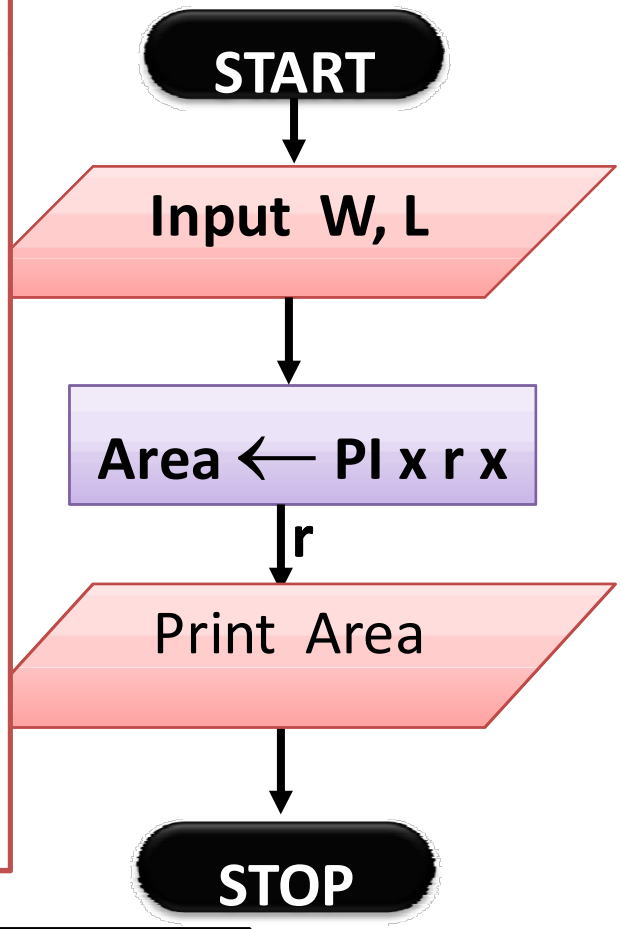
Read the radius of circle and calculate its area.

- Step 1: Input r
- Step 2:Area ← Pl x r x r
- Step 3: Print Area



C Example 2: Area of Rectangle

```
#include <stdio.h>
$define PI 3.142
int main(){
 float rad, Area;
 printf("Enter radius");
 scanf("%f",&r);

 Area=PI*r*r;

 printf("Area=%f",Area);
 return 0;
}
```



The literal PI value get replaced by 3.142

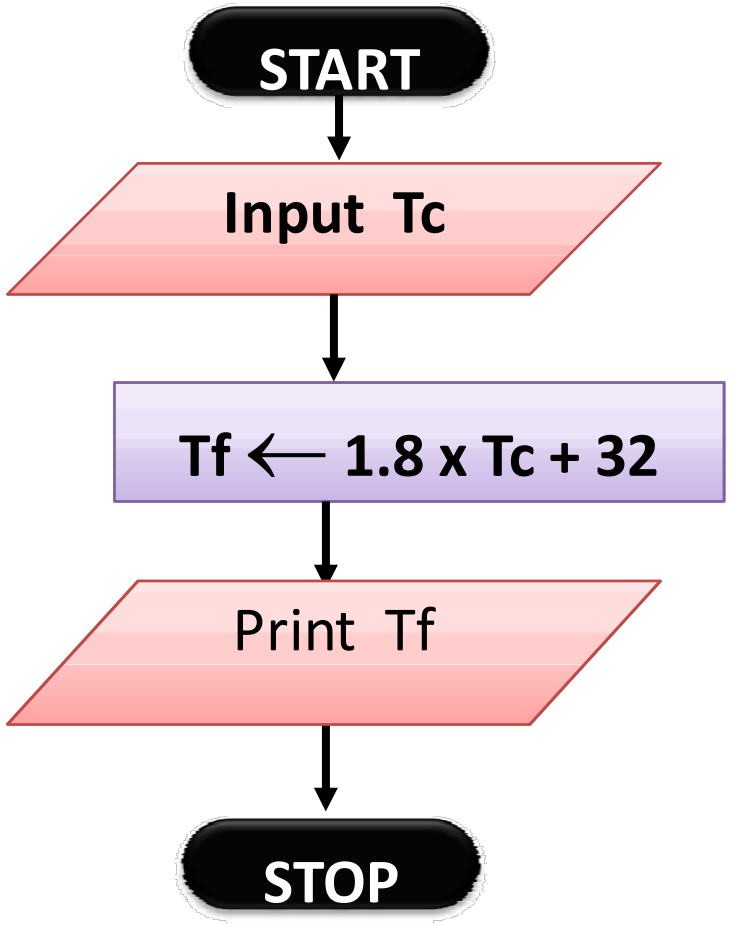
```
$gcc -E arearect.c >Preproces.c
```

C Example 3: Temp Conversion

Read the temp in Celsius and calculate temp in Fahrenheit.

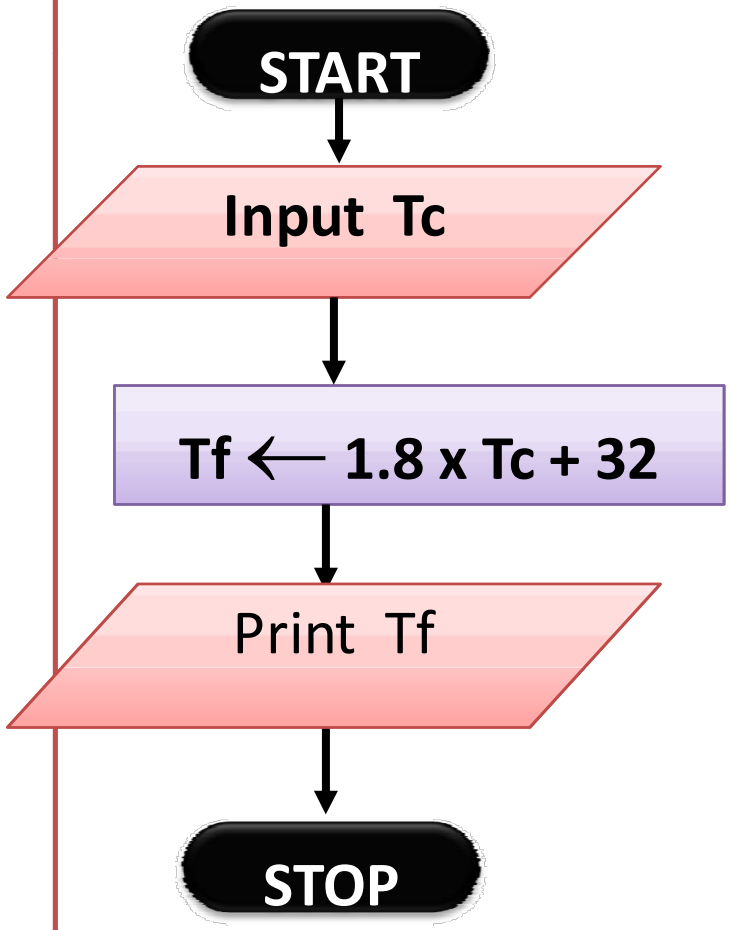
Tf=(9/5)*Tc +32

- Step 1: Input Tc
- Step 2:Tf ←(1.8xTc) +32
- Step 3: Print Tf



C Example 3: Temp Conversion

```c
#include <stdio.h>

int main(){
 float Tc,Tf;
 printf("Enter Tc");
 scanf("%f",&Tc);

 Tf=(1.8*Tc)+32;

 printf("Tf=%f",Tf);
 return 0;
}
```

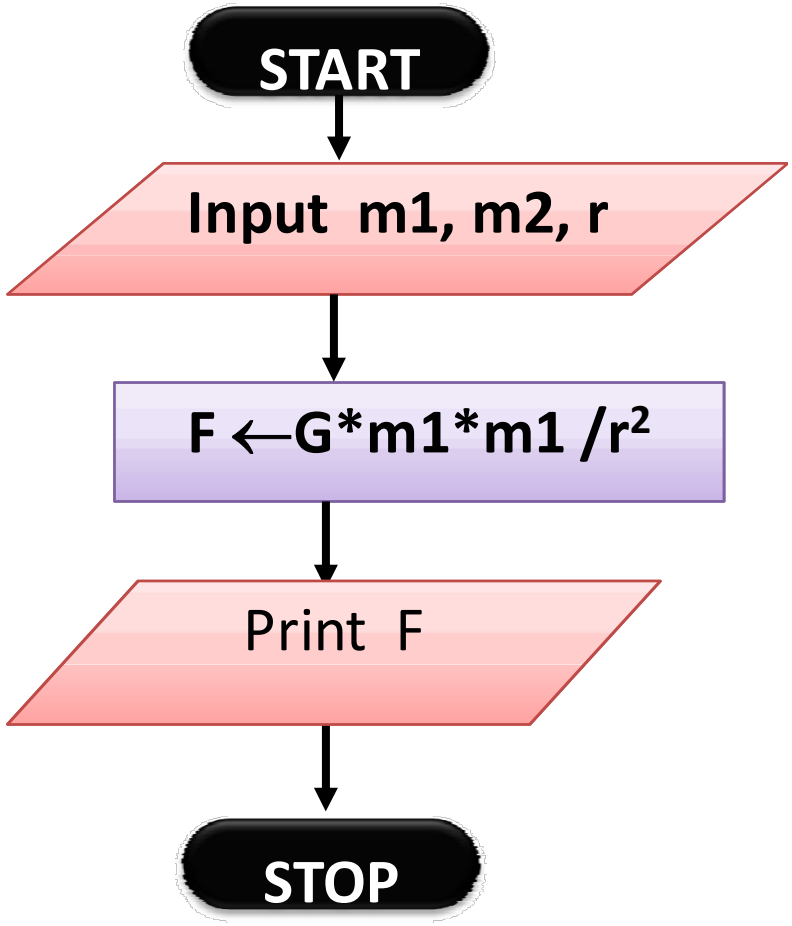


C Example 4: Force Between Two bodies

Read the masses m1 and m2 of bodies, and dist and calculate Force

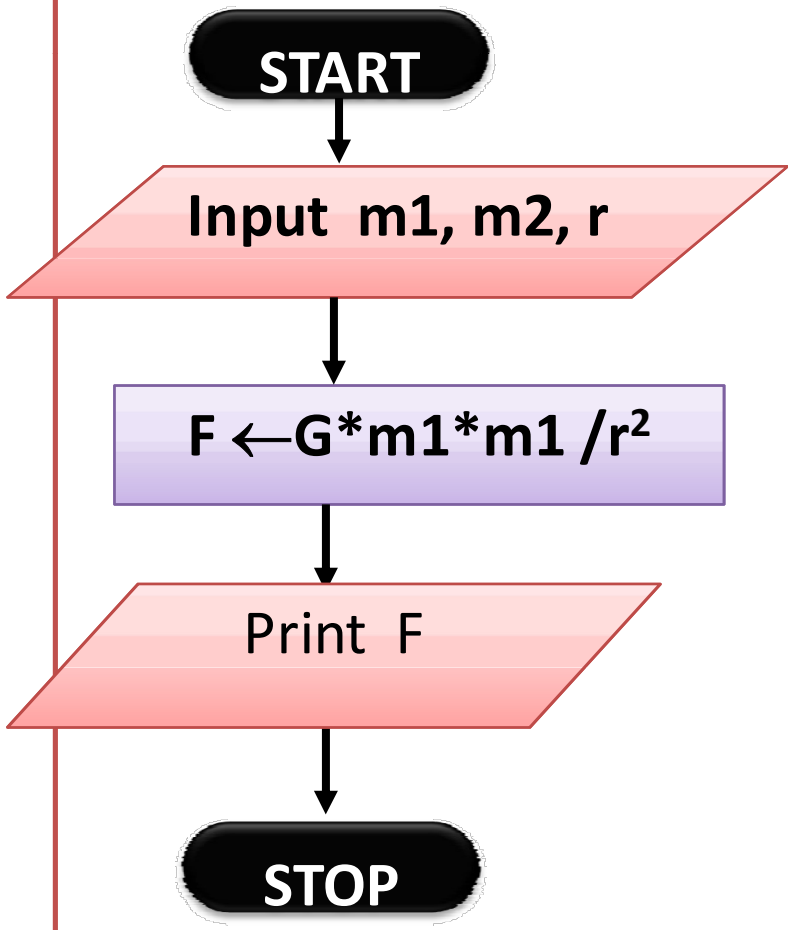
$F=G*m1*m1 /r^2$

- Step 1: Input m1, m2
- Step 2:$F \leftarrow G*m1*m1 /r^2$
- Step 3: Print F



C Example 3: Temp Conversion

```c
#include <stdio.h>
int main(){
 float F, m1,m2,r;
 float G=6.673e-11;
 printf("Enter m1 m2");
 scanf("%f %f",&m1,&m2);
 printf("Enter r");
 scanf("%f",&r);

 F=(G*m1*m2)/(r*r);

 printf("Tf=%f",F);
 return 0;
}
```



Expression Evaluation

Algebra: BEDMAS/PEDMAS Rule

- B-E-DM-AS or P-E-DM-AS or B-O-DM-AS
- B/P : Bracket or Parenthesis ( )
  - In C, only ( ) used for expression
  - Curly braces {}, and square bracket [] used for some other purpose.
    - Again [] may involves in expression as in the form of array access
- E : Exponentiation or Order (O)
- DM: Division and Multiplication
- AS : Addition and Subtraction

BEDMAS Example

- Evaluate 8+3*4/2
  - DM have higher priority as compared to AS
  - All DM get evaluated left to right

    8+**3*4**/2 = 8+ **12/2** = 8+6 = 14
- Evaluate 15-(6+1)+30/(3*2)

  15-**(6+1)**+30/**(3*2)**= 15-(7)+30/(6)

  15-7+5=8+5=13
- Evaluate $(95/19)^2+3$
  - **$(95/19)^2$**$+3 = (5)^2+3 = 25+3 =28$

BEDMAS equivalent in C
Arithmetic Operators Precedence Rule

Operator(s)	Precedence & Associativity
( )	Evaluated first. If **nested (embedded)**, innermost first.
* / %	Evaluated second. If there are several, evaluated left to right.
+ -	Evaluated third. If there are several, evaluated left to right.
=	Evaluated last, right to left.

Using Parentheses

- Use parentheses to change the order in which an expression is evaluated.

  `a + b * c` Would multiply b * c first, then add a to the result.

  If you really want the sum of a and b to be multiplied by c, use parentheses to force the evaluation to be done in the order you want.

  ```
  (a + b) * c
  ```

- Also use parentheses to clarify a complex expression.

Practice With Evaluating Expressions

Given integer variables a, b, c, d, and e, where a = 1, b = 2, c = 3, d = 4,

evaluate the following expressions:

```
a + b - c + d
a * b / c
1 + a * b % c
a + d % b - c
e = b = d + c / b - a
```

Practice With Evaluating Expressions

Given integer variables a, b, c, d, and e, where a = 1, b = 2, c = 3, d = 4,

evaluate the following expressions:

```
a + b - c + d  =3-3+4=0+4=4
a * b / c  =  2/3+4=0+4=4
1 + a * b % c  =1+2%3=1+2=3
a + d % b - c  =1+0-3=1-3=-2
e = b = d + c / b - a
```

Good Programming Practice

- It is best not to take the **“big bang” approach** to coding.
- Use an **incremental approach** by writing your code in incomplete, yet working, pieces.
- Don’t write big expression : break in to smaller pieces

Good Programming Practice (con't)

- For example, for your assignments in Lab
  - Don't write the whole program at once.
  - Just write enough to display the user prompt on the screen.
  - **Get that part working first (compile and run).**
  - Next, write the part that gets the value from the user, and then just print it out.
  - **Get that working code(compile and run).**
  - Next, change the code so that you use the value in a calculation and print out the answer.
  - **Get that working (compile and run).**
  - Continue this process until you have the final version.
  - **Get the final version working.**
- **Bottom line: Always have a working version of your program!**