

# 2

## C How to Program

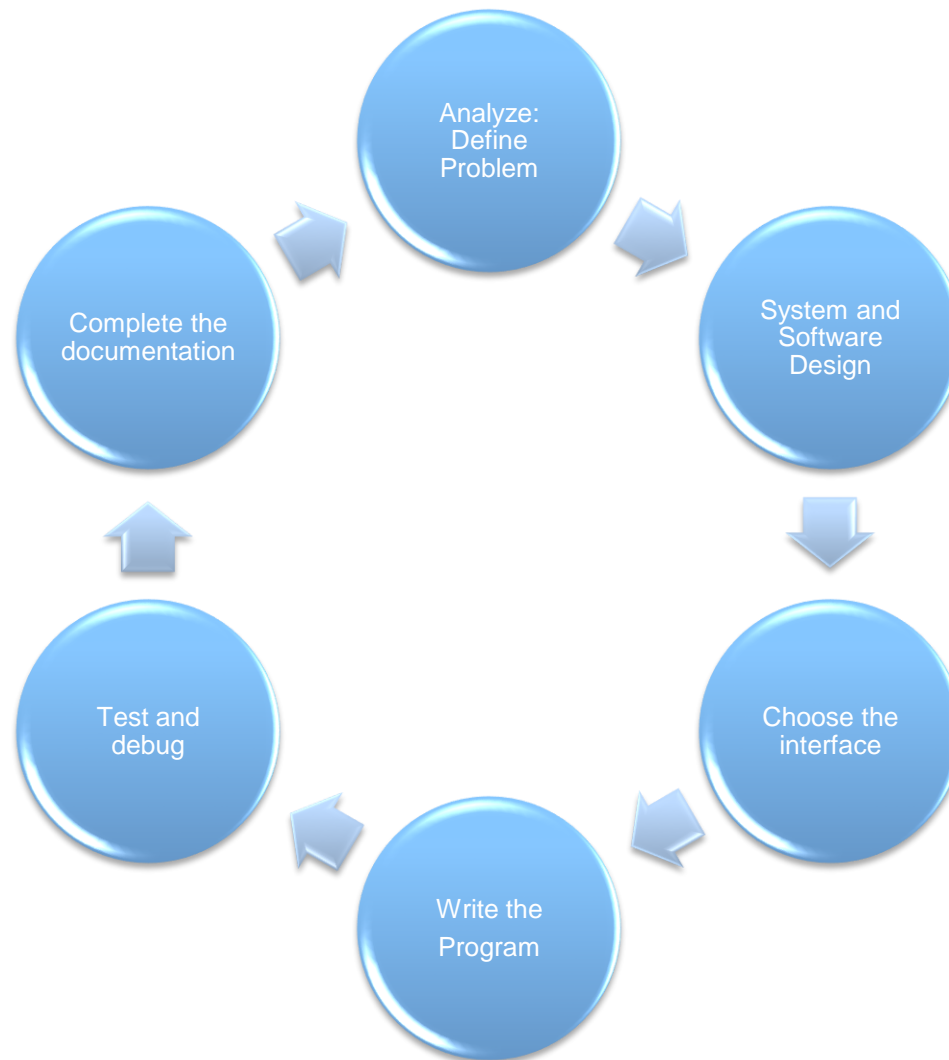
### Introduction To variables

# Introduction to variables

## Outline

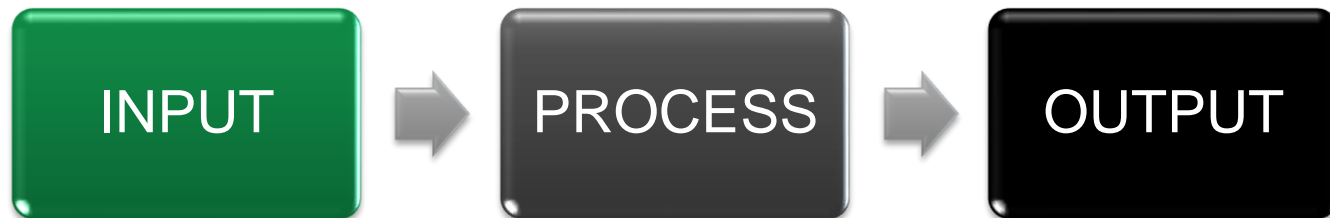
- 2.1** Program Life cycle.
- 2.2** Variables.
- 2.3** Arithmetic.
- 2.4** Assignment Operators.
- 2.5** Increment and Decrement Operators.
- 2.6** Input/Output in C
- 2.7** Our First C Program
- 2.8** A Simple C Program: (Printing a Line of Text) .
- 2.9** Escape character (\\)
- 2.10** A Simple C Program: (adding two numbers) .
- 2.11** Comments

## 2.1 Program Life cycle



# Define Problem

- Identify and determine what the **output** should be.
- Identify the data, or **input**, necessary to obtain the output.
- Determine how to **process** the input to obtain an output.



# Performing a task on the Computer

- How fast is a car traveling if it goes 50 miles in 2 hours?
  - Output: rate of speed in miles per hour
  - Input: distance and time the car has traveled
  - Process: calculate based on the formula:

$$\text{Rate} = \text{distance} / \text{time}$$

$$\begin{aligned}\text{Rate} &= 50 \text{ miles} / 2 \text{ hours} \\ &= 25 \text{ miles} / \text{hour}\end{aligned}$$

## 2.2 Variables

- Why are we discussing variables?
- And what is a variable?
- With all programming languages, what you are doing is storing things in the computer's memory, and manipulating or processing these stored things.
- If you want to add two numbers together, you put the numbers into storage areas and "tell" the system to add them up. The result is stored in another storage area.
- ***But you can't do this without variables.***



# Variables

- A variable is a bucket.
- It's a place to hold information until you need it.
- You will use variables throughout your programs to hold **temporary values** during calculations, to store user **input**, and to prepare information (**output**) that will be later displayed to users.

# Variables

- The computer memory is made of **small storage areas** used to hold the things that a program needs while it is running.
- As a programmer,
  - **you** specify these things, or you **provide** them to the computer;
  - the **computer** then **puts** them in these **storage areas**.
  - When **you need** one of them, **you let** the **computer know**.
  - The **machine locates** it and makes it available to you to use as you see fit.



# How do we tell compiler of C what variables we need?

- **So a variable is a storage area in the computer's memory.**
- Every variable has:
  - a **name**,
  - a **type**,
  - a **value**,
  - **address**
- When **new value** is **placed** into a variable, it **replaces** the previous value.
- **Reading** variables from memory does not change their value.
- .

# Data Variables

`int partNo;`

`partNo=34;`

**Name**  
**Type**  
**Value**  
**Address**



# Variable Name

- There are rules you should, and usually must, follow when naming your variables.
- The name of a variable:
  1. Must begin with a **letter** (A-Z), or \_
  2. **Cannot** have a **period**, **\$**, **+**, **-**, **/**, and other special characters.
  3. Can **have up to 255** characters. Please, just because it is allowed, don't use 255 characters.
  4. Must be **unique** inside of the function
  5. the Name **can't** be **keyword** used in C Language
  6. the Name of variable **can't** include **space**
- Once a variable has a name, you can use it as you see fit. For example, you can assign it a value and then use the variable in your program as if it represented that value.

# Keywords

Keywords			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

# Exercise

- Which of the following are legal variable name, and which are not valid variable names:
  - 12PipersPiping
  - The12thDayOfChristmas
  - \_syscall
  - \$bill
  - break
  - Student Number

# Excercise

- ❖ 12PipersPiping
  - ❑ **Illegal**, starts with a digit.
- ❖ The12thDayOfChristmas
  - ❑ **Legal**, matches definition.
- ❖ \_syscall
  - ❑ **Legal**, underscore is allowed anywhere in an identifier.
- ❖ \$bill
  - ❑ **Illegal**, dollar is not a legal character in an identifier.
- ❖ break
  - ❑ **Illegal**, break is a keyword in C Language.
- ❖ Student Number
  - ❑ **Illegal**, "Student Name" includes space .



# Declaring a Variable

- Before we use a variable, we need first to let the computer know that we will be using a certain variable.
- Informing C system about a variable prior to using that variable is referred to as **declaring** a variable.
- When a variable has been declared the computer reserves an area of memory for it.

# Declaring a Variable

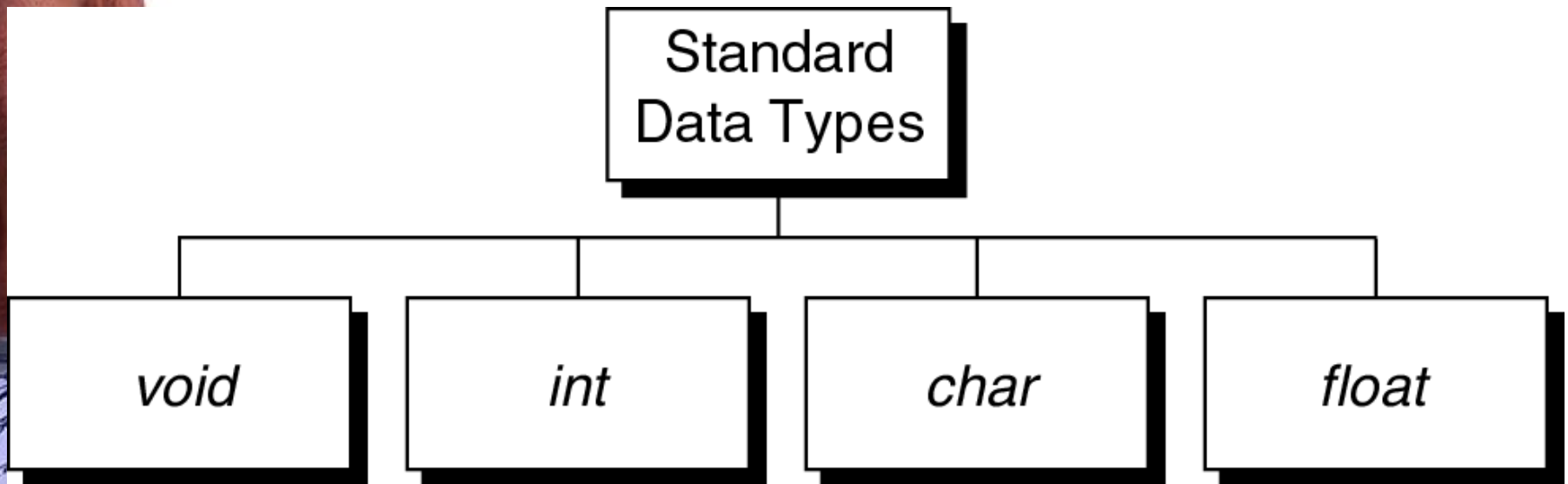
- To declare a variable: First you need to specify the data type, followed by the name for the variable, following the same rules we reviewed above.
- IMPORTANT: each statement in C must be terminated with a (;)
- Here is an example of declaring and using a variable:

**DataType StudentName;**

# Data Types

- **Data Type (definition):** A **label** applied to data that **tells** the C System how to **interpret** and **manipulate** data.
  1. Type tells the computer how much **space** to **reserve** for variables and how to interpret operations on them.
  2. A type defines a set of values and a set of operations that can be **applied** on those values.

# Standard Data Types



Standard type cannot be broken down further into sub-types. Rather, they can be used to build more complex data types call ***Derived Types*** (e.g. pointers, array, union etc.).



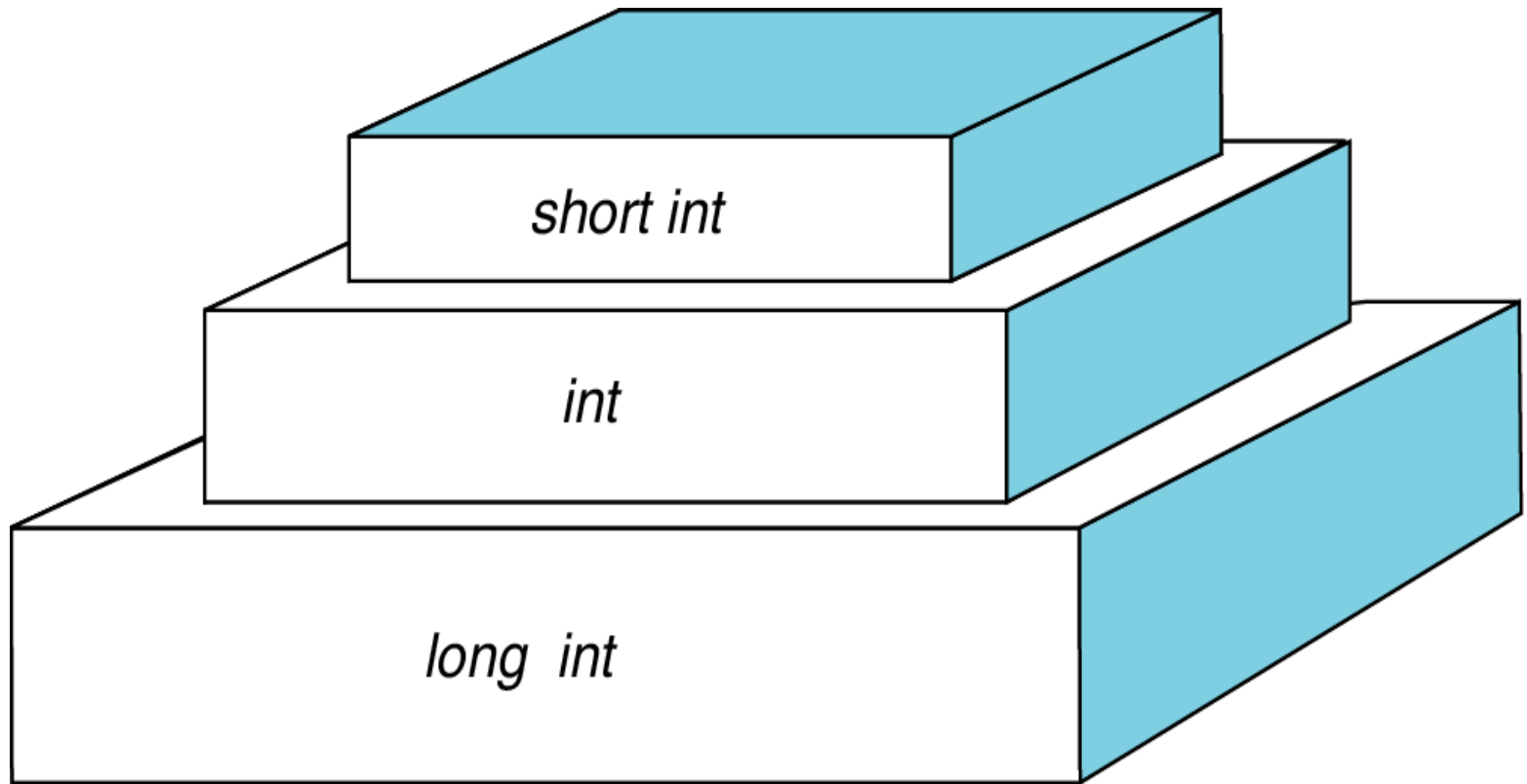
## void

- The void type has no values and no operations.
- Both the set of values and the set of operations are empty.
- Used in C to indicate the lack of a return type or parameters

## integer

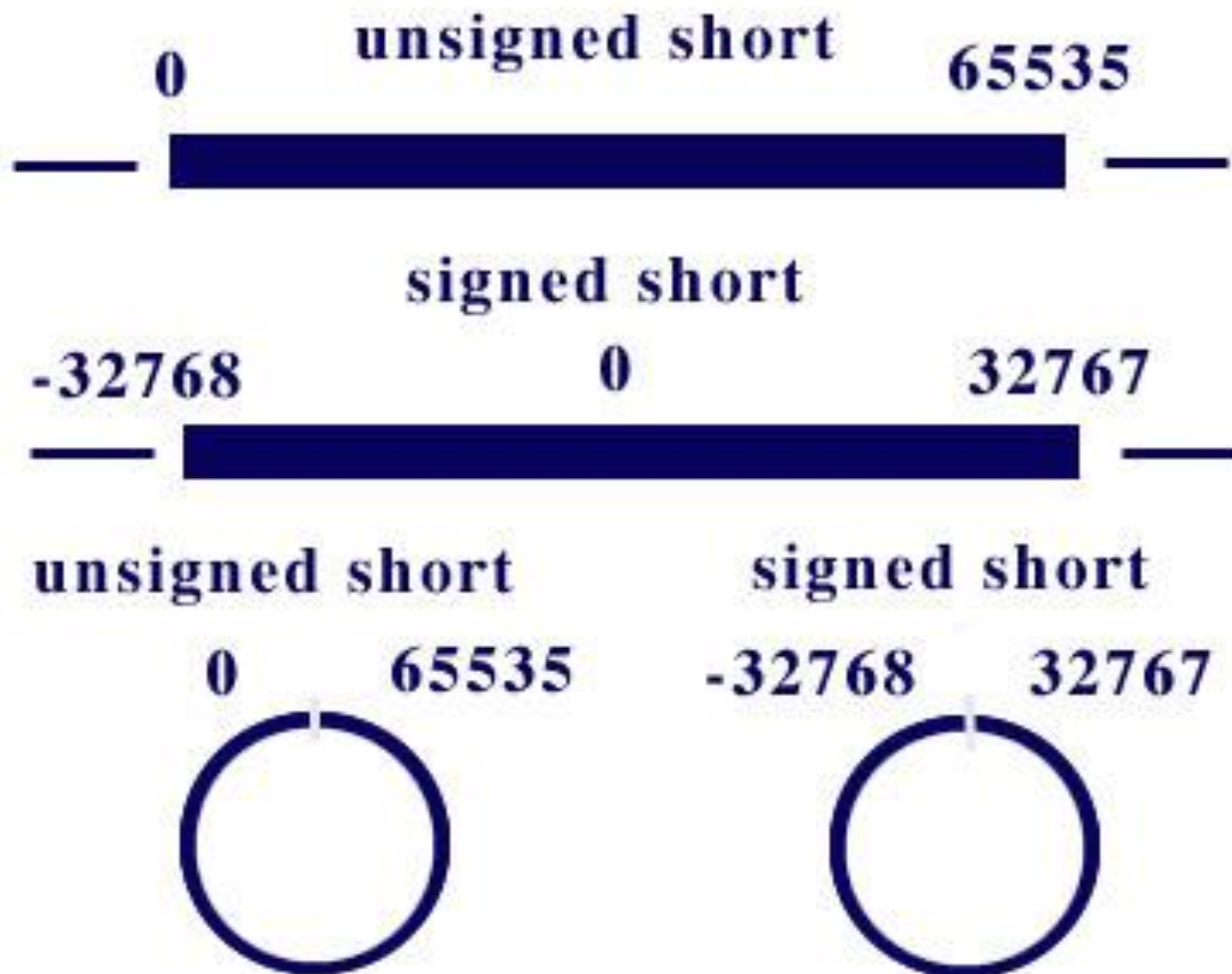
- A number without a fraction part : integral number.
- C supports three different sizes of the integer data type :
  - ***short*** int
  - ***int***
  - ***long*** int

# Integer Types (int)






# Short int



# Typical Integer Sizes

Type	Sign	Byte Size	Number of bits	Minimum value	Maximum value
short int	signed	2	16	-32,768	32,767
	unsigned			0	65,535
int	signed	2	16	-32,768	32,767
	unsigned			0	65,535
long	Signed	4	32	-2,147,483,648	2,147,483,647
	unsigned			0	4,294,967,295

1. If the integer is signed, then one bit must be used for the sign (0 is plus, 1 is minus).
2. The unsigned integer can therefore store a positive number that is twice as large as the signed integer of the same size.



A *character* in C can be interpreted as a *small integer* (between 0 and 255). For this reason, C often treats a *character* like an integer.

Examples : Character

ASCII code value

a	97(decimal) or 01100001(binary)
x	120(decimal) or 01111000(binary)

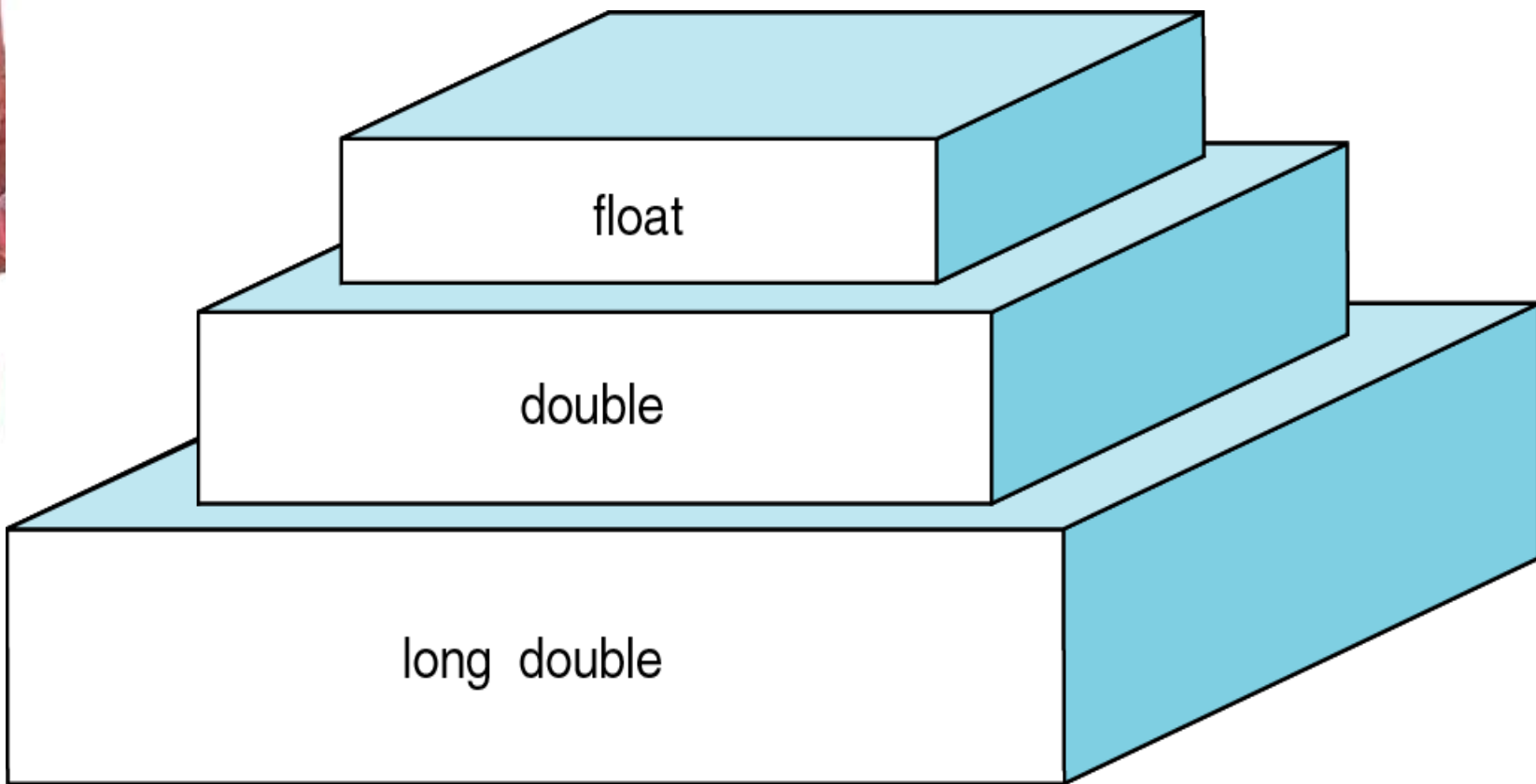
# Floating Point

- A floating-point type is a number with a fractional part, e.g. 56.78
- The C language supports three different sizes of floating-point data type : float, double, long double.
- $\text{sizeof}(\text{float}) \leq \text{sizeof}(\text{double}) \leq \text{sizeof}(\text{long double})$ .
- Floating-point is always signed.

# Typical Floating-Point Sizes

Type	Byte Size	Number of Bits
float	4	32
double	8	64
long double	10	80

# Floating-Point Types



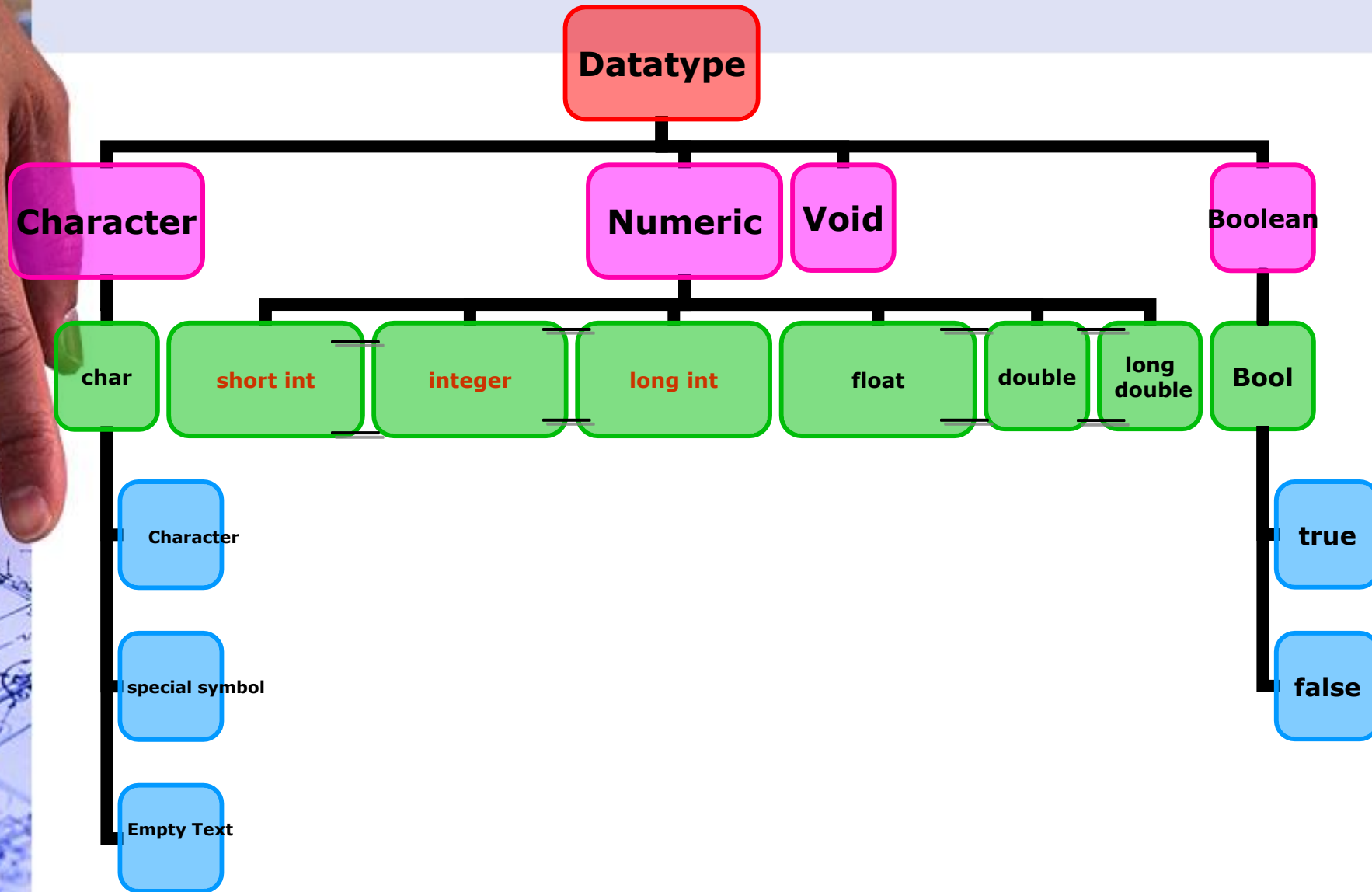


# Type Summary

<b>void</b>	void
<b>character</b>	char
<b>integer</b>	unsigned short int; unsigned int; unsigned long int
	short int; int; long int
<b>float</b>	float; double; long double
<b>Boolean</b>	bool

Each type (char, int, float) has its own internal format , therefore when the same number value is stored in different types, the internal bit configuration will be different, e.g. the ASCII character plus (+) has a value of 43. Its bit configuration is different from the integer 43, which is also different from the float 43.0

# Variable Types (Data Type)



# How do we tell C what variables we need?

- **Example:**

- Variable Name      **StudentNumber**
- Variable Type      **int**
- Variable Value      **200712345**
- **Int StudentNumber;**
- **StudentNumber = 200712345;**

- **Example:**

- Variable Name      **MyGrade\_ComputerSkills2**
- Variable Type      **float**
- Variable Value      **3.75**
- **float MyGrade\_ComputerSkills2 = 3.75;**

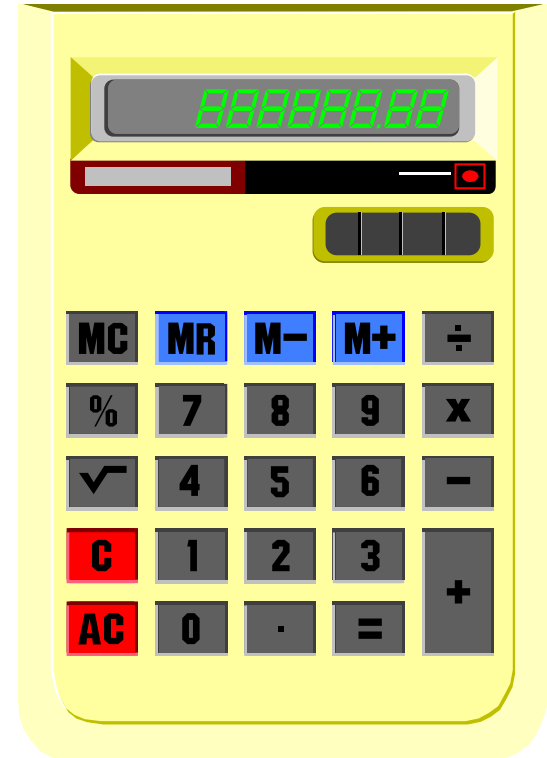
- **Example:**

- Variable Name      **IsMarried**
- Variable Type      **boolean**
- Variable Value      **false**
- **bool IsMarried = false;**

## 2.3 Arithmetic

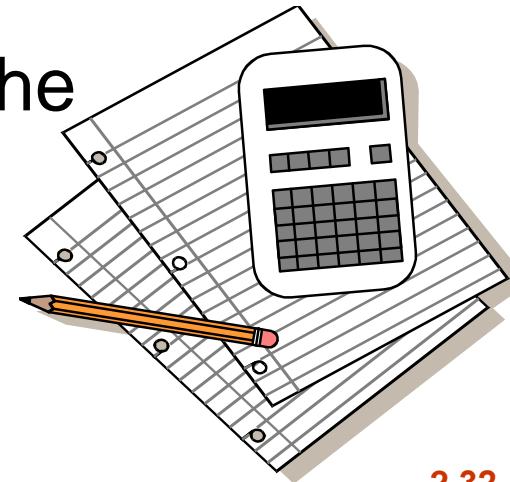
### Arithmetic Operators

- Common
  - Addition  $+$
  - Subtraction  $-$
  - Multiplication  $*$
  - Division  $/$
  - Modulus  $\%$
- Note
  - No exponentiation operator



# Arithmetic Operators

- The four operators  $+$ ,  $-$ ,  $*$ , and  $/$  work as we expect with the “normal” precedence rules (e.g.,  $5+2*3 = 11$ )
- Parenthesis can be inserted to change the order of operations (e.g.,  $(5+2)*3 = 21$ )
- Be careful of integer division -- any remainder is discarded
- The  $\%$  (modulo) operator gives the remainder of integer division



# Arithmetic operators

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	<b>f + 7</b>
Subtraction	-	$p - c$	<b>p - c</b>
Multiplication	*	$bm$	<b>b * m</b>
Division	/	$x / y$	<b>x / y</b>
Modulus	%	$r \text{ mod } s$	<b>r % s</b>



# Operator precedence

- Operator precedence
  - Some arithmetic operators act before others (i.e., multiplication before addition)
    - Use parenthesis when needed
  - Example:
    - Find the average of three variables **a**, **b** and **c**
      - Do not use:  $a + b + c / 3$
      - Use:  $(a + b + c) / 3$

Operator(s)	Operation(s)	Order of evaluation (precedence)
()	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they are evaluated left to right.
*, /, or %	Multiplication, Division, Modulus	Evaluated second. If there are several, they are evaluated left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

# Operator precedence

Step 1.  $y = 2 * 5 * 5 + 3 * 5 + 7;$  (Leftmost multiplication)

$2 * 5$  is 10

Step 2.  $y = 10 * 5 + 3 * 5 + 7;$  (Leftmost multiplication)

$10 * 5$  is 50

Step 3.  $y = 50 + 3 * 5 + 7;$  (Multiplication before addition)

$3 * 5$  is 15

Step 4.  $y = 50 + 15 + 7;$  (Leftmost addition)

$50 + 15$  is 65

Step 5.  $y = 65 + 7;$  (Last addition)

$65 + 7$  is 72

Step 6.  $y = 72$  (Last operation—place 72 in y)

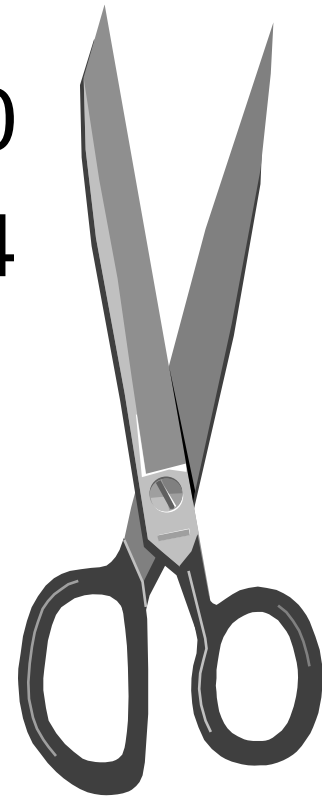
# Mod

- Produces the remainder of the division
- Examples

$5 \% 2$  evaluates to 1

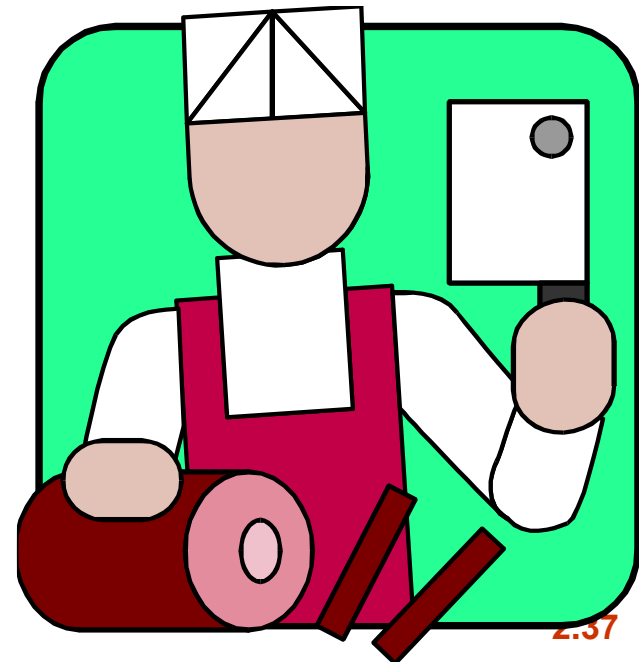
$12 \% 4$  evaluates to 0

$4 \% 5$  evaluates to 4



# Integer Division

- Integer division produces an integer result
  - It rounds down the result
- Examples
  - $3 / 2$  evaluates to 1
  - $4 / 6$  evaluates to 0
  - $10 / 3$  evaluates to 3



# Rules for Division

- C treats integers different than doubles.
- 100 is an int.
- 100.0 , 100.0000, and 100. are doubles.
- The general rule for division of int and double types is:
  - double/double -> double (normal)
  - double/int -> double (normal)
  - int/double -> double (normal)
  - int/int -> int (special case: any decimal places discarded)

# Rules for Division



- Example :
  - $220. / 100.0$       double/double -> double      **result is 2.2**
  - $220. / 100$       double/int -> double      **result is 2.2**
  - $220 / 100.0$       int/double -> double      **result is 2.2**
  - $220 / 100$       int/int -> int      **result is 2**
- Summary: division is normal unless both the numerator and denominator are int, then the result is an int (the decimal places are discarded).



# Exercise

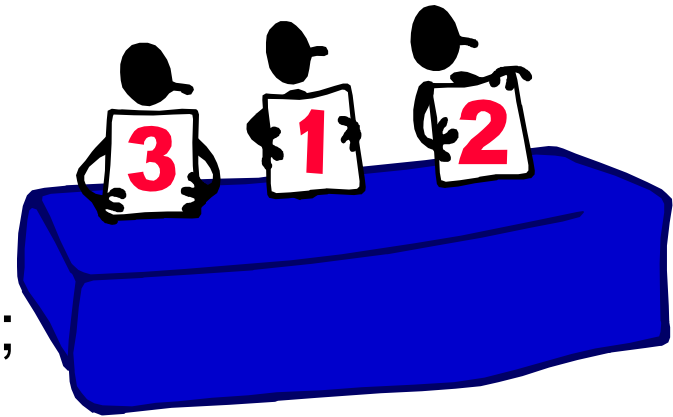
- Find the value of m in the following statements:

1. `int m=1 + 2 * 3 / 4 - 5;`

2. `int m=2 * 4 / 5 + 3 * 5 % 4;`

3. `int m=3.0 * 3 / 4;`

4. `int m=(1 + 3) * ((2 + 4 * 6) * 3) / 2 +2;`



# Forcing a Type Change

- You can change the type of an expression with a cast operation
- Syntax:

variable1 = type(variable2);

variable1 = type(expression);

- Example:

int x=1, y=2;

double result1 = x/y; // result1 is 0.0

double result2 = double(x)/y; // result2 is 0.5

double result3 = x/double(y); // result3 is 0.5

double result4 = double(x)/double(y);

// result4 is 0.5

double result5 = double(x/y); // result5 is 0.0

## 2.4 Assignment Operators

- Assignment operators abbreviate assignment expressions

**c = c + 3;**

can be abbreviated as **c += 3;** using the addition assignment operator

- Statements of the form

*variable = variable operator expression;*

can be rewritten as

*variable **operator**= expression;*

- Examples of other assignment operators:

**d -= 4    (d = d - 4)**

**e \*= 5    (e = e \* 5)**

**f /= 3    (f = f / 3)**

**g %= 9    (g = g % 9)**

## 2.5 Increment and Decrement Operators

- Increment operator (**++**)
  - Can be used instead of **c+=1**
- Decrement operator (**--**)
  - Can be used instead of **c-=1**
- **Preincrement**
  - Operator is used before the variable (**++c** or **--c**)
  - Variable is changed before the expression it is in is evaluated
- **Postincrement**
  - Operator is used after the variable (**c++** or **c--**)
  - Expression executes before the variable is changed

# Increment and Decrement Operators

- If **c** equals **5**, then  
    `x = ++c + 5;` // the value of x is 11  
    **but**  
    `x = c++ + 5;` // the value of x is 10
  - In either case, **c** now has the value of **6**
- When variable not in an expression
  - Preincrementing and postincrementing have the same effect  
        **`++c;`**
  - Has the same effect as  
        **`c++;`**

## 2.6 Input/Output in C

- C has **no** built-in statements for input or output. (Not part of the C language itself.)



- A library of functions is supplied to perform these operations. The I/O library functions are listed the “header” file **<stdio.h>**.
- You do not need to memorize them, just be familiar with them.

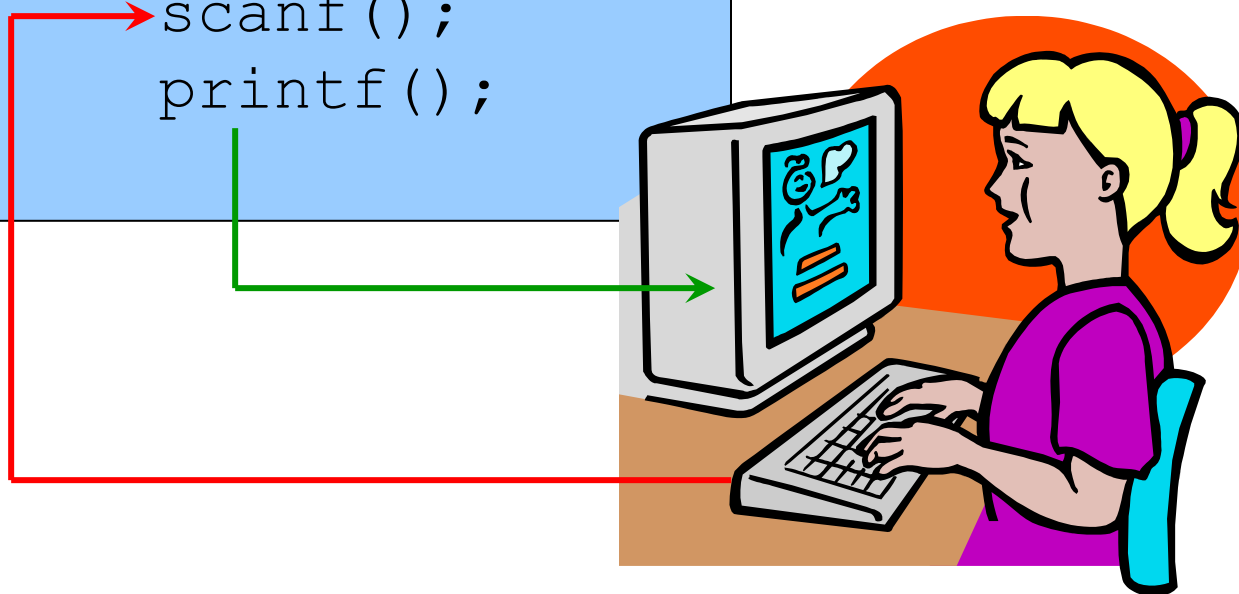


# scanf and printf Functions



```
#include <stdio.h>

int main() {
    scanf();
    printf();
}
```





# `printf()` and `scanf()` Functions

- There are various functions available in C library that can be used for input and output activities. The two functions that will be explained here are `printf()` and `scanf()`
- `printf()` function is used for displaying characters on output devices (normally video display)
- `scanf()` function is used for reading/capturing characters from input devices (normally keyboard)

# printf Function



- General format for **printf** function:  
**printf( output\_format ,[value\_list] );**
- **output\_format** tells function about the format that should be followed when displaying output
- **value\_list** is a list of variables, constants, expressions or combination of them, which values are parts of the displayed output

# printf Function

## Example:

```
printf("C How to Program\n\n");  
printf("List of Students\n");  
printf("Ahmad bin Ali");
```

**What does `\n`  
mean?**

**C How to Program**

**List of Students**

**Ahmad bin Ali \_**

# printf Function

Formats for displaying output values:

- **%s** for **string**
- **%c** for **character**
- **%hd** for **short integer**
- **%d** for **integer**
- **%ld** for **long integer**
- **%f** or **%e** for **float/double**
- **%lf** for **long double**

# printf Function - String

Format for string: **%s**

**Example:**

```
printf( "%s", "Display a string\n" );
```

**Output format**

**Value (a string constant)**

**Similar to:**

```
printf( "Display a string\n" );
```

**Output format**

Normally, it is used to display an array of characters

**Example:**

```
char name[ ] = "Nadiah";
```

```
printf( "%s", name );
```

**Value (an array of characters)**

**Output  
format**



# printf Function - Character

Format for character: **%c**

**Example:**

```
printf("%c%c%c", 'U', 'K', 'M');
```

U K M

# printf Function - Float

- Format for float: **%f**

- General format:

**%[<min\_field\_width>.<decimal\_places>]f**

**Example:**

```
printf("Value is: %10.4f", 32.6784728);
```

**4 digits**

**Value is: 32.6785**

**10 characters**

# printf Function - Float

**Example:**

```
printf("Value is: %.3f", 32.6784728);
```

3 digits

Value is: 32.678

# printf Function - Float

## Example:

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

```
void main( ) {
```

```
→ int age;
```

```
→ float height;
```

```
→ age = 21;
```

```
→ height = 1.73;
```

```
→ printf("Ali is %d years old and his height is %.5f meters\n",  
age, height);
```

age

21

height

1.73

Ali is 21 years old and his height is 1.73000 meters

# scanf Function

- General format for **scanf** function:  
**scanf( input\_format , list\_of\_variables );**
- **input\_format** tells function about the format that should be followed when capturing data
- **list\_of\_variables** are locations in memory, in which the captured data is kept
- **input\_format** should contain specification of each intended input data
- User needs to key-in data based on the format and specification set by the program
- Example:  

```
float a; int b;  
scanf ("%f%d", &a, &b);
```

# scanf Function

```
int number;
```

```
scanf("d%", &number);
```

Format section

Variables section



# scanf Function

## Example:

printf("Key-in a character and a number: ");

scanf("%c %d", &char, &num);

printf("Character: %c\n", char);

printf("Number: %d\n", num);

char

m

num

103

Key-in a character and a number: m103

Character: m

Number: 103

# End of (Input/Output in C)



Yes !! That's all?  
What's next???


OPERATORS &  
EXPRESSIONS *on the way ...*



## 2.7 Our First C Program — Hello World



# Our First C Program — Hello World

A hand is visible on the left side of the slide, pointing towards the code block. The hand is resting on a document that appears to be a technical drawing or blueprint, with various lines and text visible.

```
/* Hello.c
   Our first program */

#include <stdio.h>

/* function main begins program execution */
main()
{
    printf("hello, world\n");
}
```

# The **#include** Directive

include information  
about standard library

```
/* Hello.c
   Our first program */

#include <stdio.h>

/* function main begins program execution */
main()
{
    printf("hello, world\n");
}
```

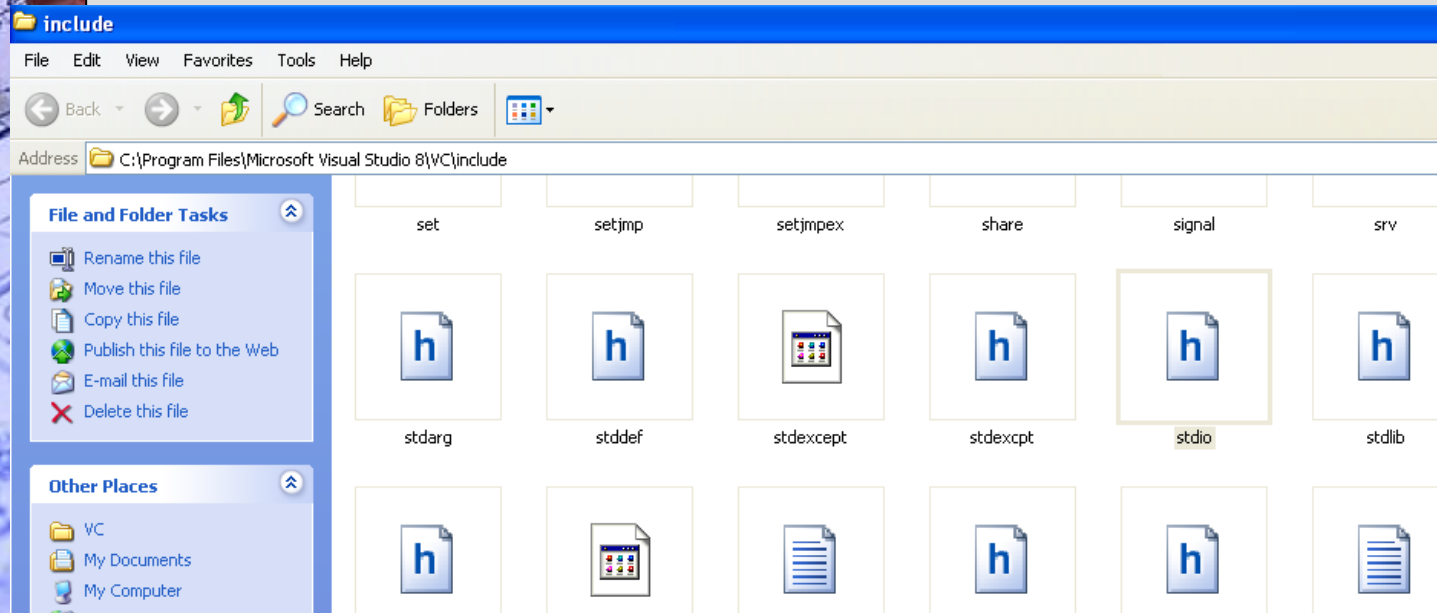


# The `#include` Directive

include information  
about standard library

```
/* Hello.c
   Our first program */

#include <stdio.h>
```



`*/`



# Entry Point of C Programs

```
/* Hello.c
   Our first program */

#include <stdio.h>

/* function main starts execution */
main()
{
    printf("hello, world\n");
}
```

define a function called **main** that receives no argument

Function **main** appears exactly once in every C program.

Function body

# Function Body

```
/* Hello.c
   Our first program */

#include <stdio.h>


/* function */
main()
{
    printf("hello, world\n");
}
```

Function body starts with {

Function body

Function body ends with }

# C Statements



```
/* Hello.c
   Our first program */

#include <stdio.h>


/* function main begins program execution */
main()
{
    printf("hello, world\n");
}
```

C statements end with ;



a C statement

## 2.8 A Simple C Program: Printing a Line of Text

A hand with a finger pointing towards the code block on the left side of the slide.

```
1  /* Fig. 2.1: fig02 01.c
2     A first program in C */
3  #include <stdio.h>
4
5  int main()
6  {
7     printf( "Welcome to C!\n" );
8
9     return 0;
10 }
Welcome to C!
```

A hand is visible on the left side of the slide, pointing towards the text. The hand is resting on a blue-tinted image of a technical drawing or blueprint.

## Comments

- Text surrounded by `/*` and `*/` is ignored by computer
- Used to describe program
- **`#include <stdio.h>`**
  - Preprocessor directive
    - Tells computer to load contents of a certain file
  - **`<stdio.h>`** allows standard input/output operations

# A Simple C Program: Printing a Line of Text

- **int main()**
  - C programs contain one or more functions, exactly one of which must be **main**
  - Parenthesis used to indicate a function
  - **int** means that **main** "returns" an integer value
  - Braces ({ and }) indicate a block
    - The bodies of all functions must be contained in braces



# A Simple C Program: Printing a Line of Text

- **printf( "Welcome to C!\n" );**
  - Instructs computer to perform an action
    - Specifically, prints the string of characters within quotes (" ")
  - Entire line called a statement
    - All statements must end with a semicolon (;)
  - Escape character (\)
    - Indicates that printf should do something out of the ordinary
    - **\n** is the newline character

# A Simple C Program: Printing a Line of Text

- **return 0;**
  - A way to exit a function
  - **return 0**, in this case, means that the program terminated normally
- Right brace }
- Indicates end of **main** has been reached
- Linker
  - When a function is called, linker locates it in the library
  - Inserts it into object program
  - If function name is misspelled, the linker will produce an error because it will not be able to find function in the library

## 2.9 Escape character (\)

Escape Sequence	Description
<code>\n</code>	Newline. Position the cursor at the beginning of the next line
<code>\t</code>	Horizontal tab. Move the cursor to the Next tab stop
<code>\a</code>	alert. Sound the system bell
<code>\\</code>	Backslash. Insert a backslash character in string
<code>\"</code>	Double quote. Inset a double-quote character in a string
<code>\r</code>	Carriage return

## 2.10

1. Initialize variables

2. Input

2.1 Sum

3. Print

Program Output

2.73

```
1  /* Fig. 2.5: fig02_05.c
2      Addition program */
3  #include <stdio.h>
4
5  int main()
6  {
7      int x1, x2, sum;          /* declaration */
8
9      printf( "Enter first No\n" ); /* prompt */
10     scanf( "%d", &x1 );          /* read an integer */
11     printf( "Enter second No\n" ); /* prompt */
12     scanf( "%d", &x2 );          /* read an integer */
13     sum = x1 + x2;             /* assignment of sum */
14     printf( "Sum is %d\n", sum ); /* print sum */
15
16     return 0; /* indicate that program ended successfully */
17 }
```

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

# 2.11 Comments

## Adding Comments in C Language

```
/* convertToGrade - returns a grade based on input mark */
/* assumption : user input is within 0-100 */
int main()
{   if (mark>80)           //for the range 81-100
        return 'A';
    else if (mark>65)      //for the range 66-80
        return 'B';
    else if (mark>45)      //for the range 46-65
        return 'C';
    else                   //for the range 0-45
        return 'F';
    return 0;
}
```

- ✓ It is a good practice to add comments for explaining what the program does.
- ✓ The c language compiler ignores any text marked as comments.
- ✓ Adding Comments make a program more readable.
- ✓ Comments cannot be nested in C i.e. you cannot have comments inside comments.

Two types of c language comments:

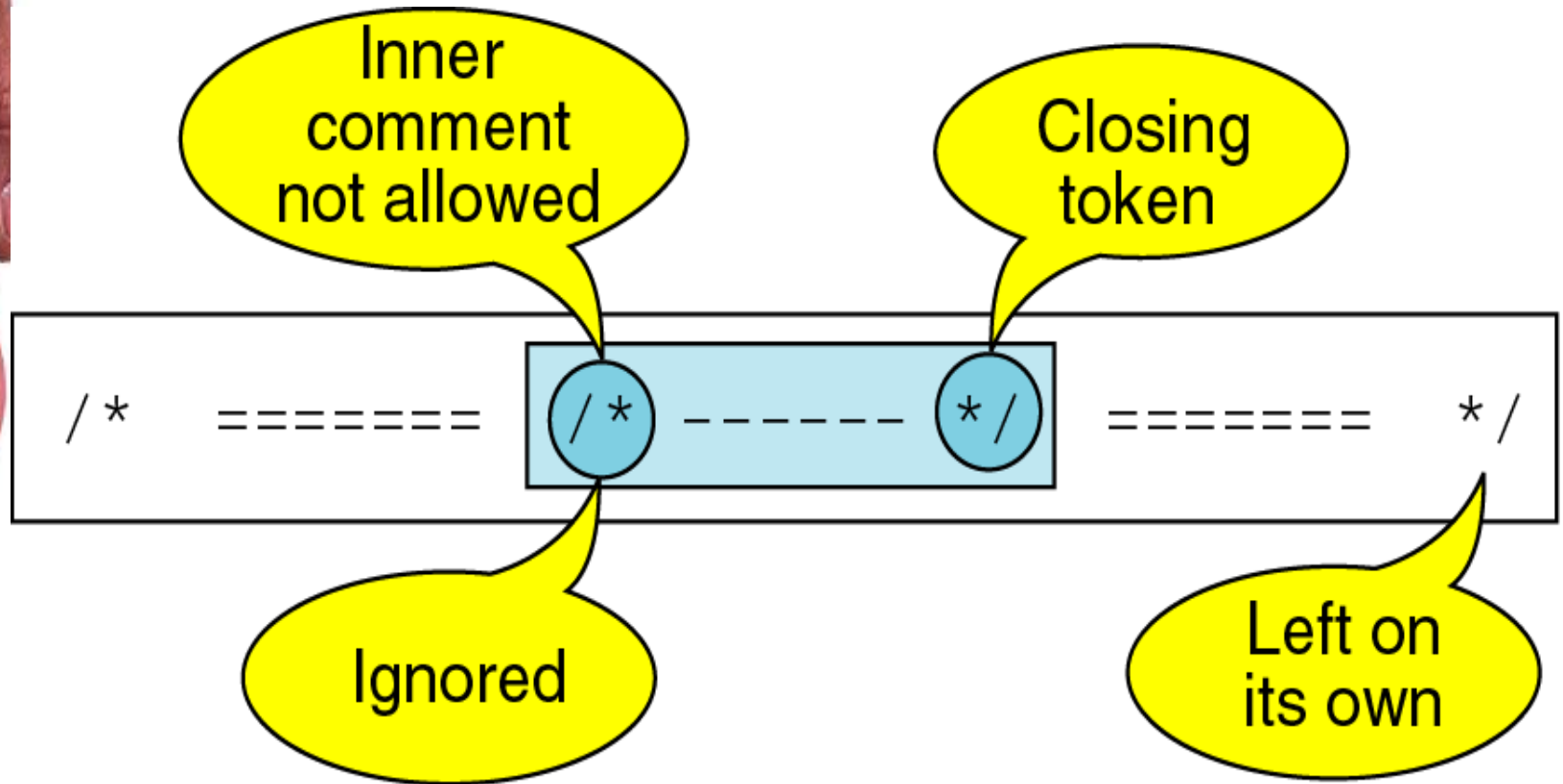
### Single-line comments

start with two slashes (//) and are limited to one line.

### Multiple-line comments

start with /\* and end with \*/

# Nested Comments problems





# Conclusion & Discussion

- Your Project
- Any problem during lab and tutorial sessions??

