Structured Programming CSE 103

Professor Dr. Mohammad Abu Yousuf

Book Reference : C Programming Language

C: The Complete Reference by Herbert Schildt (4th Edition)

Programming In Ansi C, by Balagurusamy, Publisher: Tata McGraw-Hill (7th/8th Edition)

C Language Overview

- C is a structured programming language. It is considered a high-level language because it allows the programmer to concentrate on the problem at hand and not worry about the machine that the program will be using.

C Language Overview

- ❖The C programming language is a general-purpose, high-level language.

- ❖Originally developed by Dennis M. Ritchie at Bell Labs.

- ❖C was originally first implemented on the DEC PDP-11 computer in 1972.

- ❖In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard.

C Language Overview (Cont..)

- C was invented to write an operating system called UNIX.

- C is a successor of B language, which was introduced around 1970.

- In 1983 the American National Standards Institute (ANSI) formed a committee to establish a standard definition.

  --Called ANSI Standard C.

- The UNIX OS was totally written in C by 1973.

Why use C?

The C has now become a widely used professional language for various reasons.

- ❑ Easy to learn
- ❑ Structured language
- ❑ It produces efficient programs.
- ❑ It can be compiled on a variety of computer platforms.

In structured programming paradigm, we write functions (sometimes called: procedures, sub routines, methods) to perform certain tasks within the program.

Some Terminologies

- Algorithm
  - A step-by-step procedure for solving a particular problem.
  - Independent of the programming language.
- Program
  - A translation of the algorithm/flowchart into a form that can be processed by a computer.
  - Typically written in a high-level language like C, C++, Java, etc.

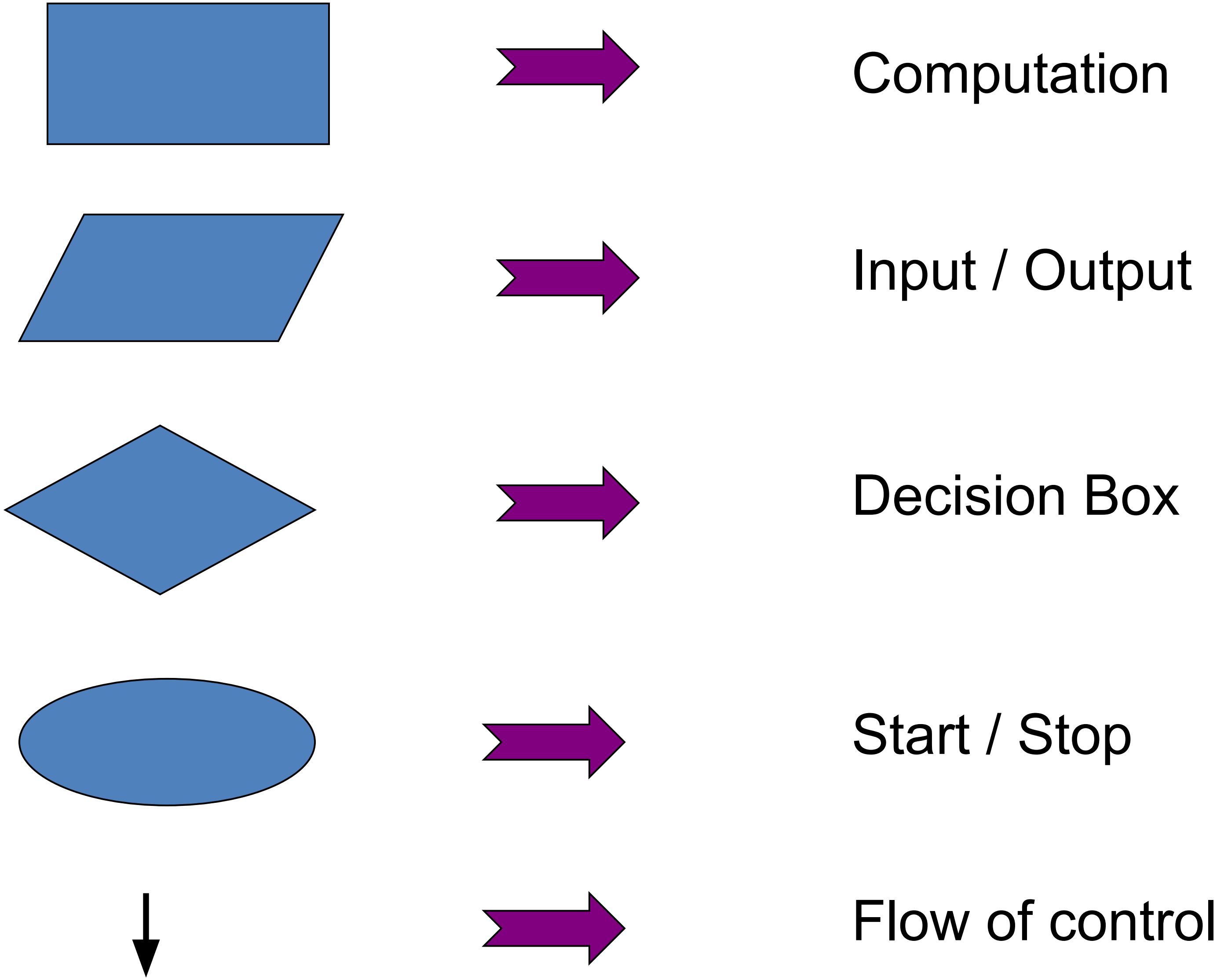
Analyze the Problem (1)

Design Algorithm

1. ***Flowcharts***
2. ***pseudo-code***

Analyze the Problem (2)

Flowchart: basic symbols



Computation

Input / Output

Decision Box

Start / Stop

Flow of control

Analyze the Problem (3)

- <u>Pseudo-code:</u>

```
<Algorithm name>
     // input ?
    // function?
    // Output?
Begin
  <data definition>
  <actions>
End
```

The comment lines **“//”**

What is a program?

- A sequence of instructions that a computer can interpret and execute.

  -If I tell you the way from Bashundhara to Dhanmondi … I will tell sequence of instructions…. Any wrong instruction leads to a undesired result.

Programming Language

Three types of programming languages

1. Machine languages

✔Expressed in binary.

✔Directly understood by the computer.

✔Not portable; varies from one machine type to another.

-Program written for one type of machine will not run on another type of machine.

✔Difficult to use in writing programs.

2. Assembly languages

English-like abbreviations representing elementary computer operations (translated via assemblers)

Example:

```
LOAD    BASEPAY
ADD     OVERPAY
STORE   GROSSPAY
```

Programming Language (Cont..)

3. High-level languages

   Codes similar to everyday English

   Use mathematical notations (translated via compilers)

   Example:

```
grossPay = basePay + overTimePay
```

❑High-level languages are easier to use.

-They are closer to the programmer.

-Examples:

Fortran, Cobol, C, C++, Java.

-Requires an elaborate process of translation.

Using a software called *compiler*.

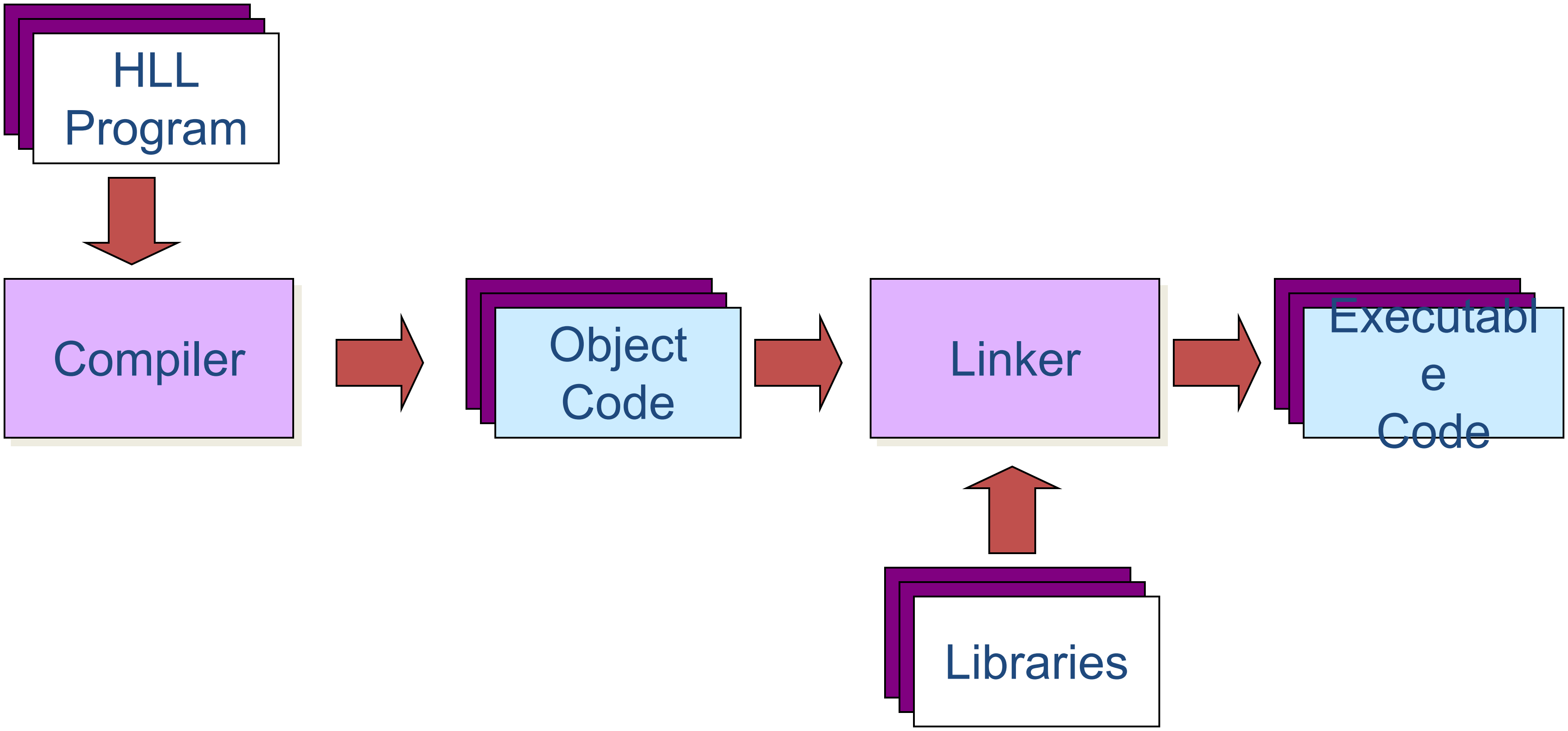
-They are portable across platforms.

Programming Language (Cont..)

Low-level language:

- A machine language or an assembly language. Low-level languages are closer to the hardware than are high-level programming languages, which are closer to human languages.

From HLL to executable

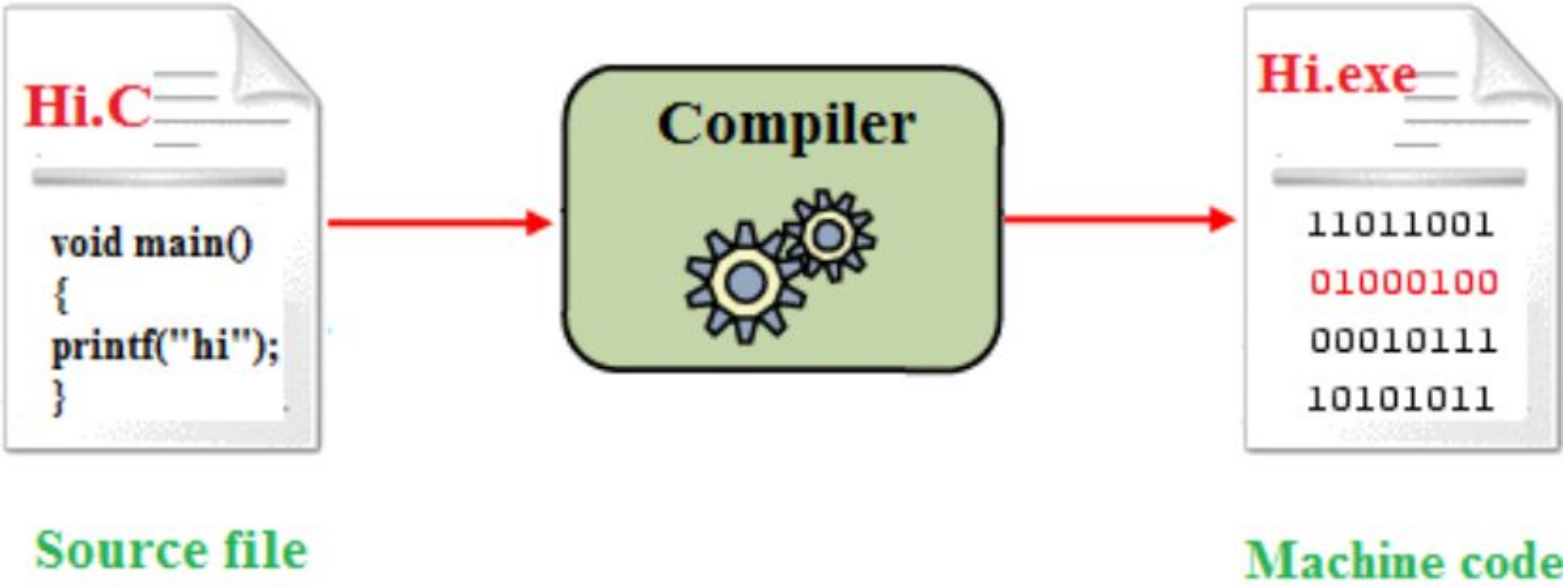


Some programmer jargon

- Some words that will be used a lot:
  - Source code: The stuff you type into the computer. The program you are writing.
  - Compile (build): Taking source code and making a program that the computer can understand.
  - Executable: The compiled program that the computer can run.
  - Language: The core part of C central to writing C code.
  - Library: Added functions for C programming which are bolted on to do certain tasks.
  - Header file: Files ending in .h which are included at the start of source code.

Compilation

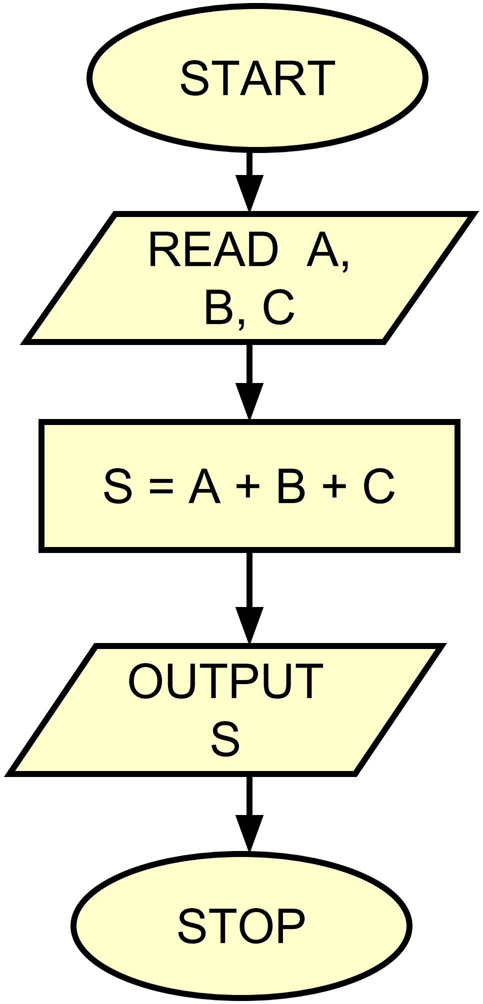
- Compilation translates your source code (in the file hello.c) into object code (machine dependent instructions for the particular machine you are on).
- Linking the object code will generate an executable file.



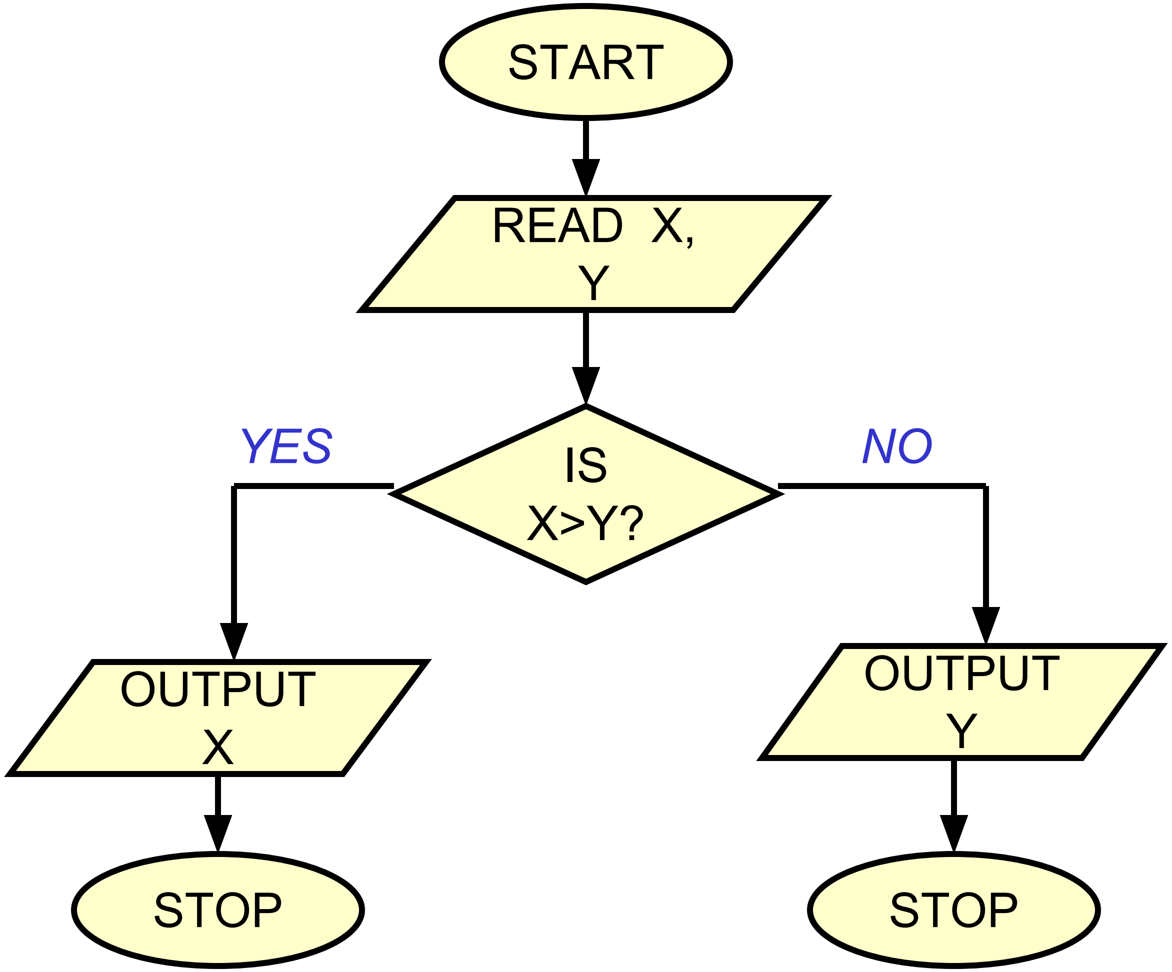
Problem solving

- Step 1:
  - Clearly specify the problem to be solved.
- Step 2:
  - Draw flowchart or write algorithm.
- Step 3:
  - Convert flowchart (algorithm) into program code.
- Step 4:
  - Compile the program into object code.
- Step 5:
  - Execute the program.

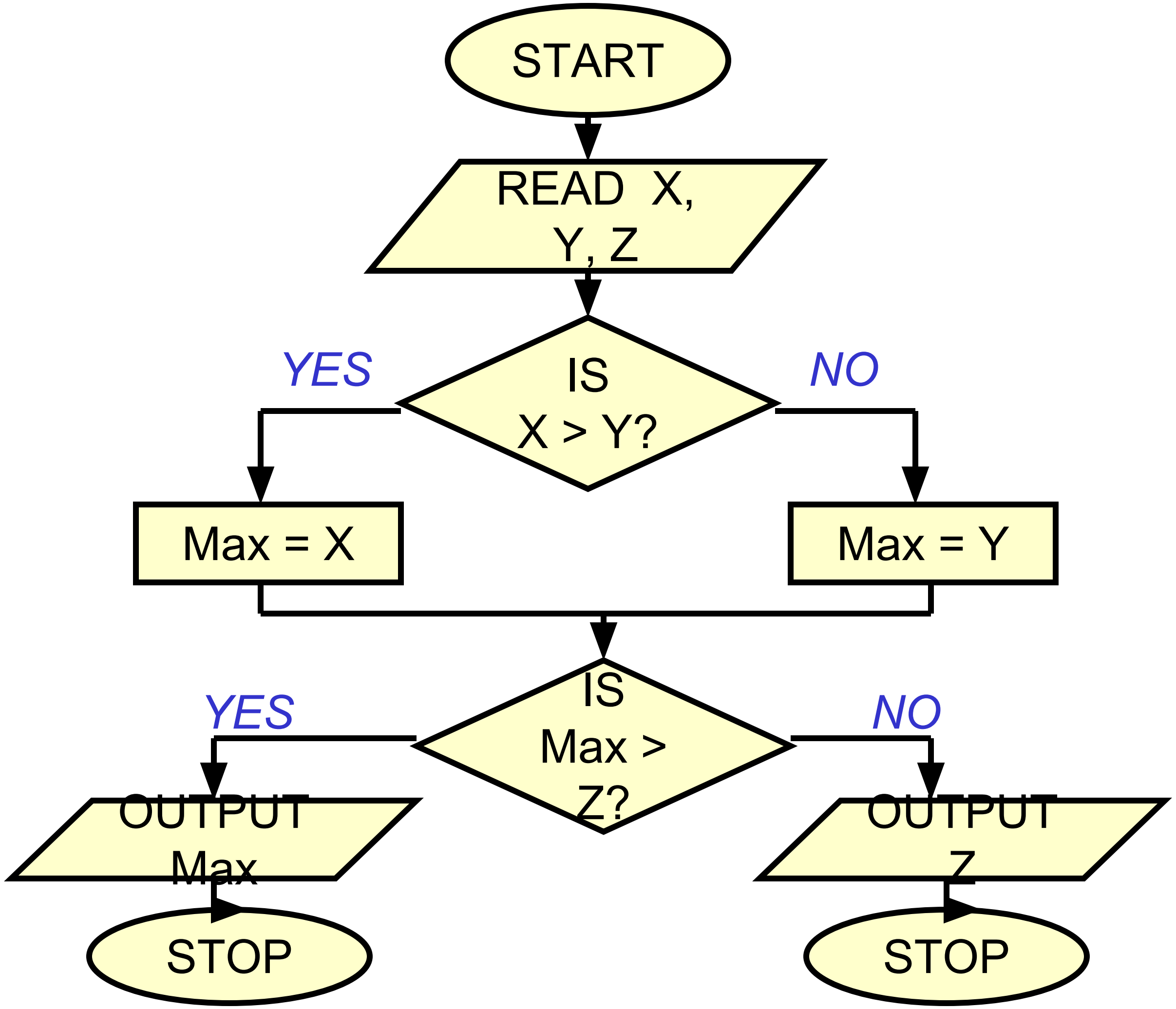
Example 1: *Adding three numbers*



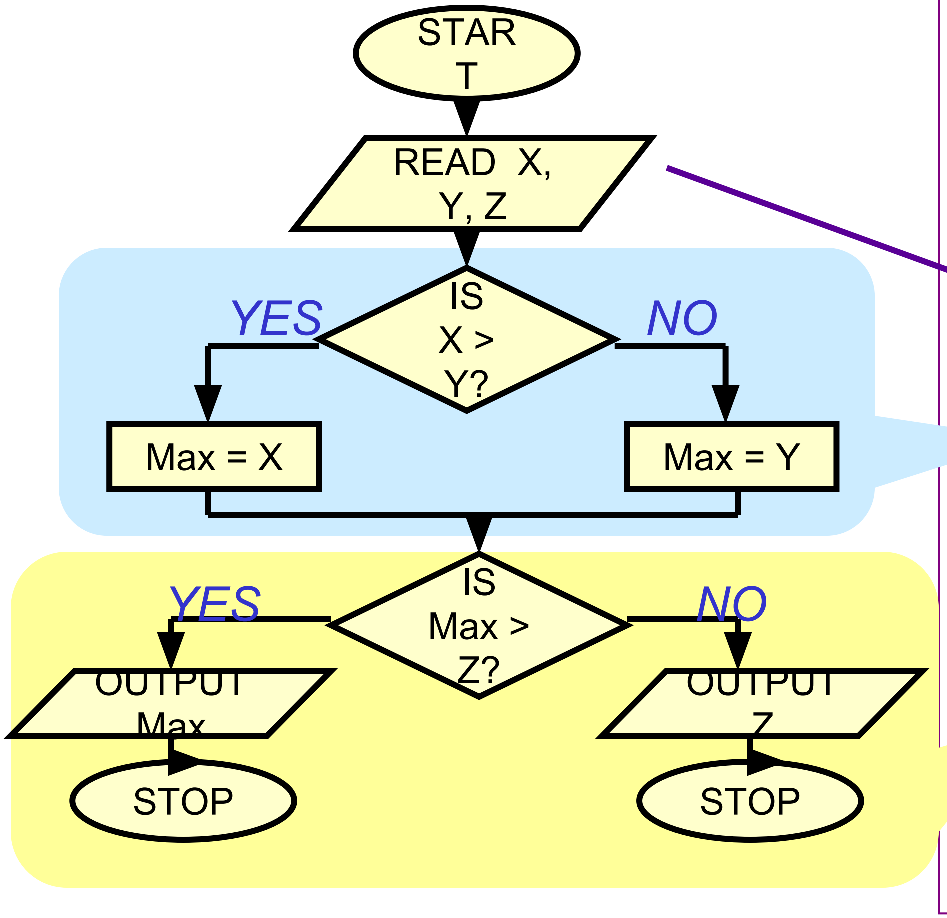
Example 2: *Larger of two numbers*



Example 3: *Largest of three numbers*



Example 3: *Largest of three numbers*

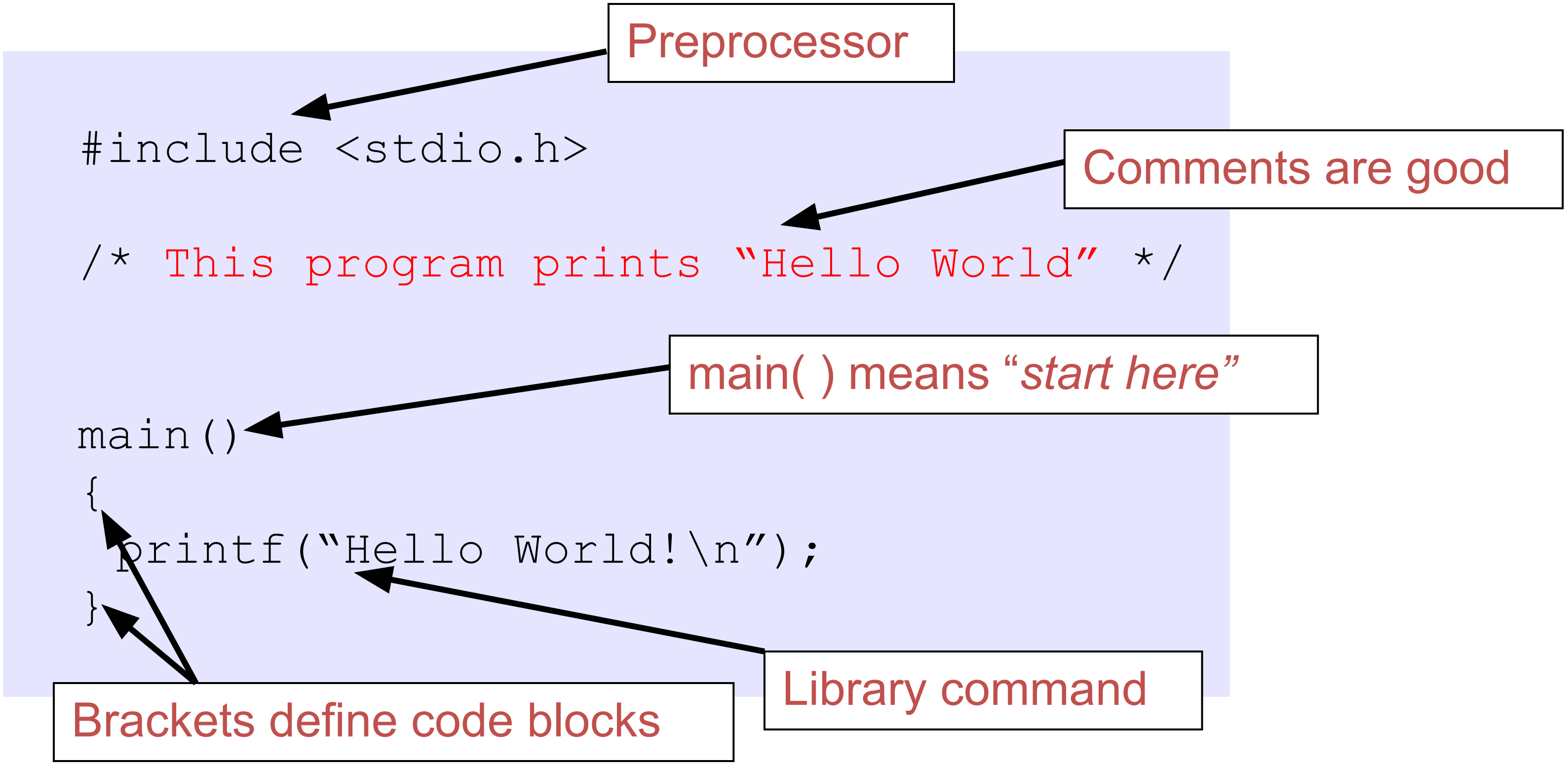


```
#include <stdio.h>
/* FIND THE LARGEST OF THREE NUMBERS */

main()
   {
         int   a, b, c, max;
         scanf (“%d %d %d”, &x, &y, &z);

         if  (x>y)
         max = x;
         else max = y;

         if (max > z)
         printf(“Largest is %d”, max);
         else printf(“Largest is %d”, z);
}
```

Our First C Program: Hello World

```c
#include <stdio.h>
int main ( )
{
    printf ("Hello, World!\n");
    return 0;
}
```

Our First C Program: Hello World



```
#include <stdio.h>

/* This program prints “Hello World” */


main()
{
 printf(“Hello World!\n”);
}
```

Preprocessor

Comments are good

main( ) means “*start here*”

Library command

Brackets define code blocks

Our First C Program: Hello World (Cont..)

- 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

- **`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

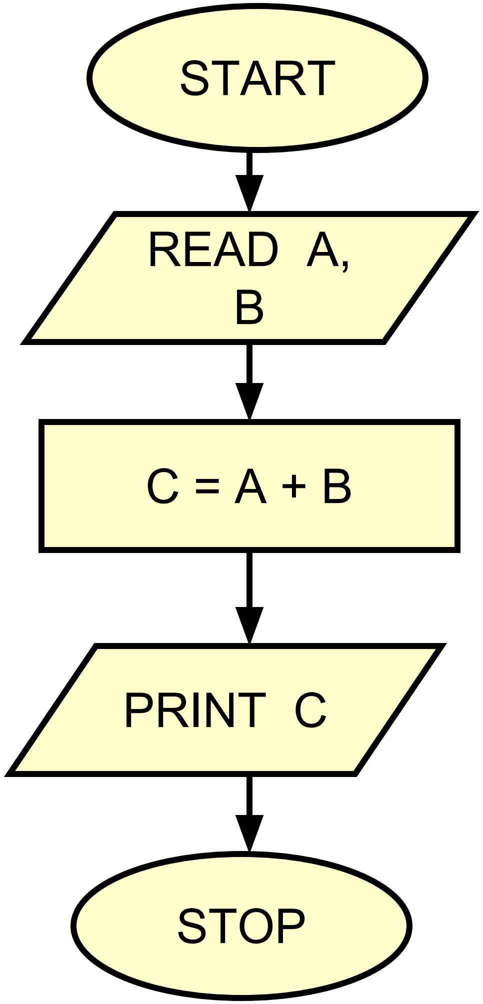
Our First C Program: Hello World (Cont..)

- **`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

Our First C Program: Hello World (Cont..)

- **`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

Example 1: *Adding two numbers*



START

READ A, B

C = A + B

PRINT C

STOP

```
#include <stdio.h>
main()
{
  int a, b, c;

  scanf(“%d%d”,&a, &b);


  c = a + b;


  printf(“Output is : %d”,c);
}
```

Variable Declaration

Example 2 : *Area of a circle*

```
#include <stdio.h>
int main ()
{    int radius;
     float area;
     printf ("Enter radius (i.e. 10) : ");
     scanf ( "%d", &radius);
     area = 3.14159 * radius * radius;
     printf ("\nArea = %f\n\n", area);
     return 0;
     }
```

Example 3

```
#include <stdio.h>
int main ()
{
    int i, j;
    for (i = 0; i < 10; i++)
    {
     printf ("\n");
     for (j = 0; j < i+1; j++ )
               printf ( "A");
    }
    printf("\n");
    return 0;
}
```

Structure of a C program

- Every C program consists of one or more functions.
  - One of the functions must be called *main*.
  - The program will always begin by executing the main function.

- Each function must contain:
  - A function *heading*, which consists of the function *name*, followed by an optional list of *arguments* enclosed in parentheses.
  - A list of argument *declarations*.
  - A *compound statement*, which comprises the remainder of the function.

Desirable Programming Style

- Clarity
  - The program should be clearly written.
  - It should be easy to follow the program logic.

- Meaningful variable names
  - Make variable/constant names meaningful to enhance program clarity.
    - ‘area’ instead of ‘a’
    - ‘radius’ instead of ‘r’

- Program documentation
  - Insert comments in the program to make it easy to understand.
  - Never use too many comments.

- Program indentation

  -- Use proper indentation.

  – Structure of the program should be immediately visible.

Indentation Example: *Good Style*

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
    int   a, b, c;

    scanf(“%d%d%d”, &a, &b, &c);

    if  ((a>b) && (a>c))
            printf(“\n Largest is %d”, a);
    else
            if  (b>c)
                printf(“\n Largest is %d”, b);
            else
                printf(“\n Largest is %d”, c);
}
```

Indentation Example: ***Bad Style***

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */
main()
{
int   a, b, c;
scanf(“%d%d%d”, &a, &b, &c);
if  ((a>b) && (a>c))
printf(“\n Largest is %d”, a);
    else
if  (b>c)
  printf(“\n Largest is %d”, b);
else
printf(“\n Largest is %d”, c);
}
```

Keywords of C

- Flow control (6) – `if, else, return, switch, case, default`

- Loops (5) – `for, do, while, break, continue`

- Common *types* (5) – `int, float, double, char, void`

- Structures (3) – `struct, typedef, union`

- Counting and sizing things (2) – `enum, sizeof`

Keywords of C (Cont..)

- Rare but still useful *types* (7) – `extern, signed, unsigned, long, short, static, const`

- Evil keywords which we avoid (1) – `goto`

- (3) – `auto, register, volatile`

The C Character Set

- The C language alphabet:
  - Uppercase letters ‘A’ to ‘Z’
  - Lowercase letters ‘a’ to ‘z’
  - Digits ‘0’ to ‘9’
  - Certain special characters:

!	#	%	^	&	*	(	)
-	_	+	=	~	[	]	\
|	;	:	‘	“	{	}	,
.	<	>	/	?	blank		

Some simple operations for variables

- In addition to `+`, `−`, `*` and `/` we can also use

  `+=`, `−=`, `*=`, `/=`, `−−` and `%` (modulo)

`n++` *increment n*

`n--` *decrement n*

`a+=5` *is equivalent to* `a = a+5;`

`a-=5` *is equivalent to* `a = a-5;`

`a*=5` *is equivalent to* `a = a*5;`

`a/=5` *is equivalent to* `a = a/5;`

`(x % y)` gives the remainder when `x` is divided by `y`

Identifiers and Keywords

- Identifiers
  - Names given to various program elements (variables, constants, functions, etc.)
  - May consist of *letters*, *digits* and the *underscore* ('_') character, with no space between.
  - First character must be a letter or underscore.
  - An identifier can be arbitrary long.
    - Some C compilers recognize only the first few characters of the name (16 or 31).
  - Case sensitive
    - 'area', 'AREA' and 'Area' are all different.

Valid and Invalid Identifiers

- Valid identifiers

  X

  abc

  simple_interest

  a123

  LIST

  stud_name

  Empl_1

  Empl_2

  avg_empl_salary

- Invalid identifiers

  10abc

  my-name

  “hello”

  simple interest

  (area)

  %rate

C Variables Names (1)

- Variables are named memory locations that have a type, such as integer or character, which is inherited from their type.

- The type determines the values that a variable may contain and the operations that may be used with its values.

- Its value can be changed, and it can be reused many times.

C Variables Names (2)

- The syntax to declare a variable:

```
        type     variable_name;
```

Example:

```
        int a;
        float b;
        char c;
```

C Variables Names (3)

Variable Names:

- Names may contain letters, digits and underscores
- The first character must be a letter or an underscore.
  - the underscore can be used but watch out!!
- Case matters!
- C keywords cannot be be used as variable names.

```
present, hello, y2x3, r2d3, ...      /* OK */
_1993_tar_return                /* OK but don’t */
Hello#there                     /* illegal */
double                      /* shouldn’t work */
2fartogo                        /* illegal */
```

C Variables Names (4)

Suggestions regarding variable names:

- DO: use variable names that are descriptive
- DO: adopt and stick to a standard naming convention
  - sometimes it is useful to do this consistently for the entire software development site

- AVOID: variable names starting with an underscore
  - often used by the operating system and easy to miss
- AVOID: using uppercase only variable names
  - generally these are pre-processor macros (later)

Data Types in C (1)

int : integer quantity

Typically occupies 4 bytes (32 bits) in memory.

char : single character

Typically occupies 1 byet (8 bits) in memory.

float : floating-point number (a number with a decimal point)

Typically occupies 4 bytes (32 bits) in memory.

double : double-precision floating-point number

Data Types in C (2)

- There are a number of qualifiers which can be applied to the basic types
  - length of data
    - short int: v "shorter" int, <= number of bits in an int
      v can also just write "short"
    - long int: v a "longer int", >= number of bits in an int
      v often the same number of bits as an int
      v can also just write "long"
    - long double v generally extended precision floating point

Data Types in C (3)

– signed and unsigned

- unsigned int v an int type with no sign

  v if int has 32-bits, range from 0..2^{32}-1

  v also works with long and short

- unsigned char v a number from 0 to 255

- signed char   va number from –128 to 127 (8-bit signed value)

  v very similar to byte in Java

Some Examples of Data Types

- int

  0, 25, -156, 12345, –99820

- char

  ‘a’,   ‘A’,   ‘*’,   ‘/’,   ‘ ’, ‘2’

- float

  23.54, –0.00345, 25.0

Thank You