Lecture-6, 7 and 8

Introduction to C

Programming Or Implementation Phase

- **Coding**
  - Translation or conversion of each operation in the flowchart or algorithm (pseudocode) into a computer-understandable language.
  - Coding should follow the format of the chosen programming language.
  - Many types or levels of computer programming language such as:
    - Machine language
    - Symbolic language or assembly language
    - Procedure-oriented language
  - The first two languages are also called *low-level programming language*. While the last one is called *high-level programming language*.

Programming Or Implementation Phase

- **Machine Language**
  - Machine language uses number to represent letters, alphabets or special character that are used to represent bit pattern.
  - Example:
    - an instruction to add regular pay to overtime pay, yielding total pay might be written in machine language as follows:

      16 128 64 8

    - in which 16 is a code that mean ADD to the computer. The 128 and 64 are addresses or location at which regular pay and overtime pay are stored. The 8 represents the storage location for the total pay.

Programming Or Implementation Phase

- Sometimes, bit pattern that represent letters and alphabets are used for coding.
- Example:

Instead of:	16	128	64	8
Use:	10000	10000000	1000000	1000

- This representation is ideal for a computer but difficult and tedious to the programmer to write a lengthy program.

Programming Or Implementation Phase

► **Symbolic Language or Assembly Language**

- ► A symbolic language or assembly language is closely related to machine language in that, one symbolic instruction will translate into one machine-language instruction.
- ► Contain fewer symbols, and these symbols may be letters and special characters, as well as numbers.
- ► As example, a machine language instruction

```
16   128   64   8
```

can be rewritten in assembly language as

```
ADD   LOC1   LOC2   LOC3
```

- ► Which means, add content of location LOC1 to location LOC2 and put the result in location LOC3.

Programming Or Implementation Phase

- **Procedure – Oriented Language**
  - Programmer has to know the computer hardware before he can write program in machine and assembly language. It means the language is machine dependent.
  - Using procedure – oriented language, the programmer can run the program in any computer hardware.
  - A special program called a *compiler* will translate program written using procedure – oriented language to machine language.

Programming Or Implementation Phase

- Some example of the language:
  - COBOL (COmmon Business Oriented Language)
  - FORTRAN (FORmula TRANslation)
  - Pascal
  - C
  - C++
  - BASIC, etc.
- These languages are also called *high-level programming language*

Programming Or Implementation Phase

Computer Language	Instruction Format
Machine language	16 128 64 8
Assembly language	ADD LOC1 LOC2 LOC3
BASIC	LET T = R + 0
FORTRAN	TOTAL = RPAY + OPAY
COBOL	ADD RPAY, OPAY GIVING TOTAL
Pascal	TOTAL : = RPAY + OPAY
C	TOTAL = RPAY + OPAY

Programming Or Implementation Phase

- **Compiling and Debugging**
  - Compiling is a process of a compiler translates a program written in a particular high–level programming language into a form that the computer can execute.
  - The compiler will check the program code know also as source code so that any part of the source code that does not follow the format or any other language requirements will be flagged as syntax error.
  - This syntax error in also called bug, when error is found the programmer will debug or correct the error and then recompile the source code again.
  - The debugging process is continued until there is no more error in the program.

Programming Or Implementation Phase

- **Testing**
  - The program code that contains no more error is called executable program. It is ready to be tested.
  - When it is tested, the data is given and the result is verified so that it should produced output as intended.
  - Though the program is error free, sometimes it does not produced the right result. In this case the program faces logic error.
  - Incorrect sequence of instruction is an example that causes logic error.

Programming Or Implementation Phase

► **Documentation and Maintenance**

- ► When the program is thoroughly tested for a substantial period of time and it is consistently producing the right output, it can be documented.
- ► Documentation is important for future reference. Other programmer may take over the operation of the program and the best way to understand a program is by studying the documentation.
- ► Trying to understand the logic of the program by looking at the source code is not a good approach.
- ► Studying the documentation is necessary when the program is subjected to enhancement or modification.
- ► Documentation is also necessary for management use as well as audit purposes.

The Modules and The Functions

- The programmer breaks the problem into modules, each with specific function.
- It is much easier to write and test many small modules than a single large program.
- Modules are arranged according to processing order in interactivity chart.

The rules for designing modules

1. Each module is an entity and has one entrance and one exit.
2. Each module has a single function such as printing, calculating or entering data.
3. Each module is short enough to be easily read and modified.
4. The length of module governed by its function and the number of instruction to be executed.
5. A module is developed to control the order of processing.

Types of modules

1. Control module
   - Show the overall flow of data through the program. All other modules are subordinate to it.
2. Init module
   - Also called the preparation module, process instruction that are executed only once – at the beginning.
3. Process Data module
   - May be processed only once, or may be part of a loop.
     1. Calculation Modules
        - Do arithmetic calculations.

Types of modules

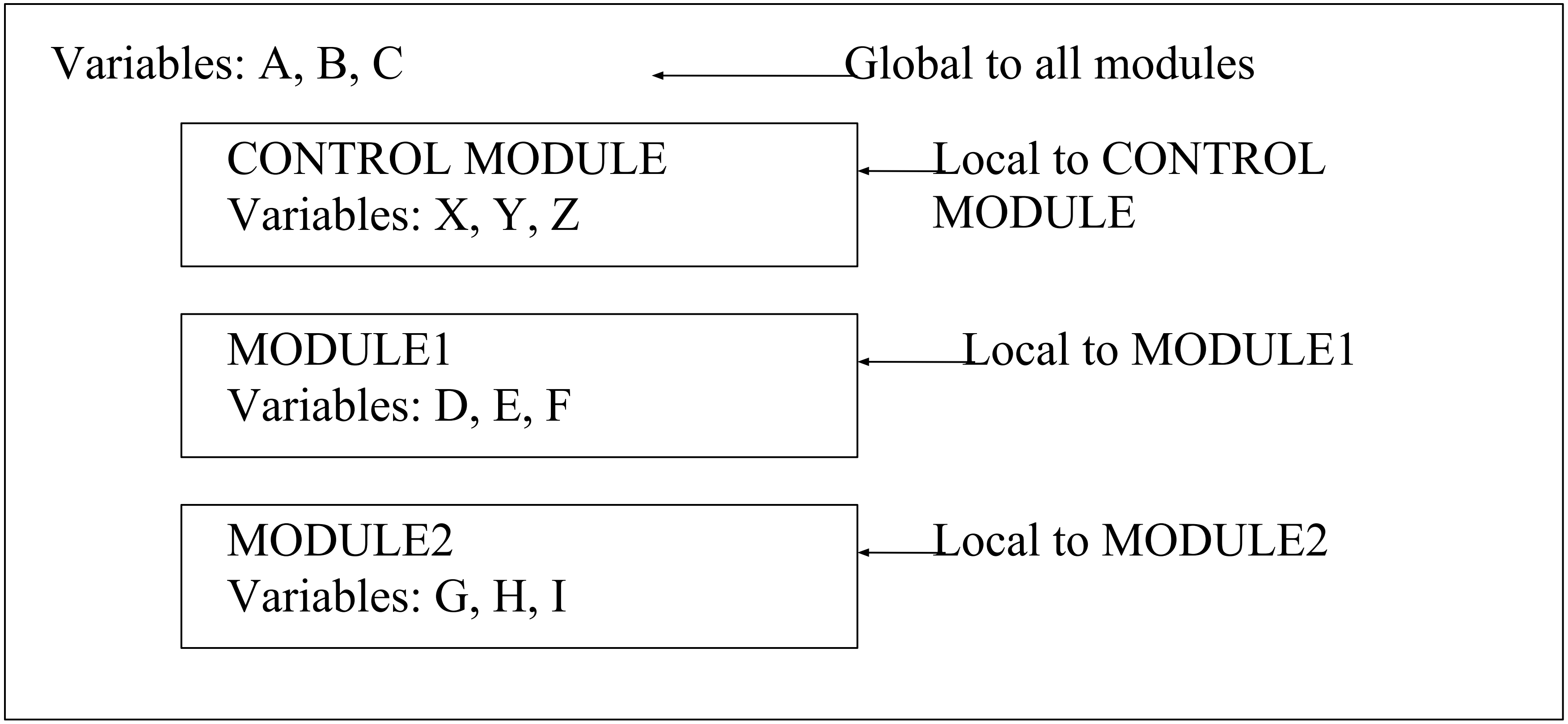
2. Print Modules
   - Print output lines.
3. Read and Data validation modules
   - Read or input data, validate data
   - Validation modules separate from read modules

5. Wrap-up module
   - Execute only once at the end.
   - Include closing file and printing totals.
6. Event module
   - Such as mouse down, mouse up, key entry.

Local and Global Variables

- The concept of local and global variables to allow cohesion and coupling to occur.
- Local variables
    - defined within a module
    - used only by the module itself
- Global variables
    - defined outside of the individual modules
    - can be used by all modules

Scope of Local and Global Variables



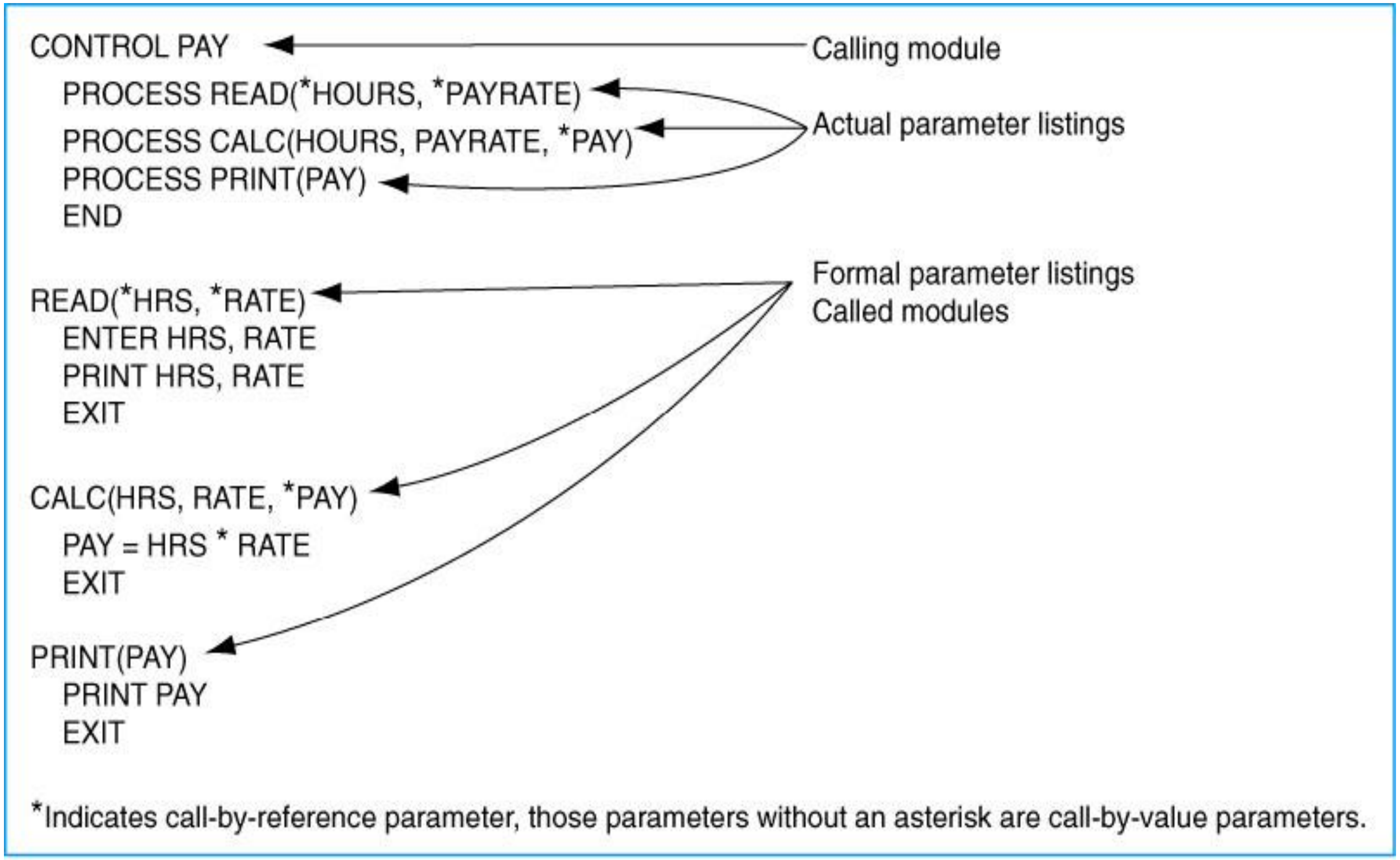
Parameters

- Parameters are local variables that are passed or sent from one module to another.
- Parameters are another way of facilitating coupling that allows the communication of data between modules.
- Eg: Read(A, B, C) – A, B & C are parameters
- There are two types of parameters:
  - Actual Parameter

    list of parameters that follow the module name being processed in the calling module
  - Formal Parameter

    list of parameters that follow the module name at the beginning of the module.

Parameter Terminology



```
CONTROL PAY
   PROCESS READ(*HOURS, *PAYRATE)
   PROCESS CALC(HOURS, PAYRATE, *PAY)
   PROCESS PRINT(PAY)
   END

READ(*HRS, *RATE)
   ENTER HRS, RATE
   PRINT HRS, RATE
   EXIT

CALC(HRS, RATE, *PAY)
   PAY = HRS * RATE
   EXIT

PRINT(PAY)
   PRINT PAY
   EXIT
```

*Indicates call-by-reference parameter, those parameters without an asterisk are call-by-value parameters.

Beginning Problem-Solving Concepts for the Computer

- Data used in processing
- **Constant** is a value that never changes during the processing of all instructions in a solution
- Constant is given a location in memory and a name.
- The value of a **variable** may change during processing

Rules for naming and using variables:

- Name according to what it represented
- Do not use spaces in a variable name
- Do not a dash
- Consistent in the use of variable names
- Consistent when using upper and lowercase characters

Data Types

- Data are unorganized facts
- Goes as input and is processed to produce output, information
- Computers must be told the data type of each variable and constant
- 3 common types: numeric, character, and logical
- Other data types: date data type and user-defined data types

Data Type	Data Set	Examples
Numeric: INTEGER	All whole numbers	3580 −46
Numeric: REAL	All real numbers (whole + decimal)	3792.91 4739416.0 0.00246
CHARACTER (surrounded by quotation marks)	All letters, numbers, and special symbols	“A” “a” “M” “z” “k” “1” “5” “7” “8” “0” “+” “=” “(” “%” “$”
STRING (surrounded by quotation marks)	Combinations of more than one character	“Arcata” “95521” “707-444-5555”
LOGICAL	TRUE FALSE	TRUE FALSE

Rules of Data Types

- Programmer designates the data type during the programming process
- Data types cannot be mixed
- Each data types used data set
- Calculations involved only numeric data type.

Functions

- Small sets of instructions that perform specific task and return values
- Used as part of instruction in a solution
- Divided into classes:
  - Mathematical functions – used in science and business
  - String functions – used to manipulate string variables
  - Conversion functions – used to convert data from one data type to another
  - Statistical functions – used to calculate things such as maximum value
  - Utility functions – access information outside the program, i.e. date and time function
- Functions use data, called parameters. Functions normally do not alter parameters

  Functionname(parameters list)

Operators

- They are data connectors within expressions and equations.
- They tell computer how to process the data
- **Operands** are data the operator connects and processes
- **Resultant** is the answer
- Data type of the operand and the resultant depends on the operator:
  - Mathematical operators
  - Relational operators
  - Logical operators
- Operators have a **hierarchy** or precedence

Order of Operations	Operand Data Type	Resultant Data Type
( ) Reorders the hierarchy; all operations are completed within the parentheses using the same hierarchy.		
1. Functions		
Mathematical Operators		
2. Power	Numeric	Numeric
3. \, MOD	Numeric	Numeric
4. *, /	Numeric	Numeric
5. +, −	Numeric	Numeric
Relational Operators		
6. =, <, >, <=, >=, <>	Numeric or string or character	Logical
Logical Operators		
7. NOT	Logical	Logical
8. AND	Logical	Logical
9. OR	Logical	Logical

Expressions and Equations

- Expression processes data, the operands, through the use of operators
- Equation stores the resultant of an expression in a memory location in the computer through the equal (=) sign.
- Equations are often called assignment statements

Writing C Program

- A programmer uses a **text editor** to create or modify files containing C code.
- Code is also known as **source code**.
- A file containing source code is called a **source file**.
- After a C source file has been created, the programmer must **invoke the C compiler** before the program can be **executed** (**run**).

3 Stages of Compilation

Stage 1: **Preprocessing**

- Performed by a program called the **preprocessor**
- Modifies the source code (in RAM) according to **preprocessor directives (preprocessor commands**) embedded in the source code
- skips comments and white space from the code
- The source code as stored on disk is <u>not</u> modified.

3 Stages of Compilation (con’t)

Stage 2: **Compilation**

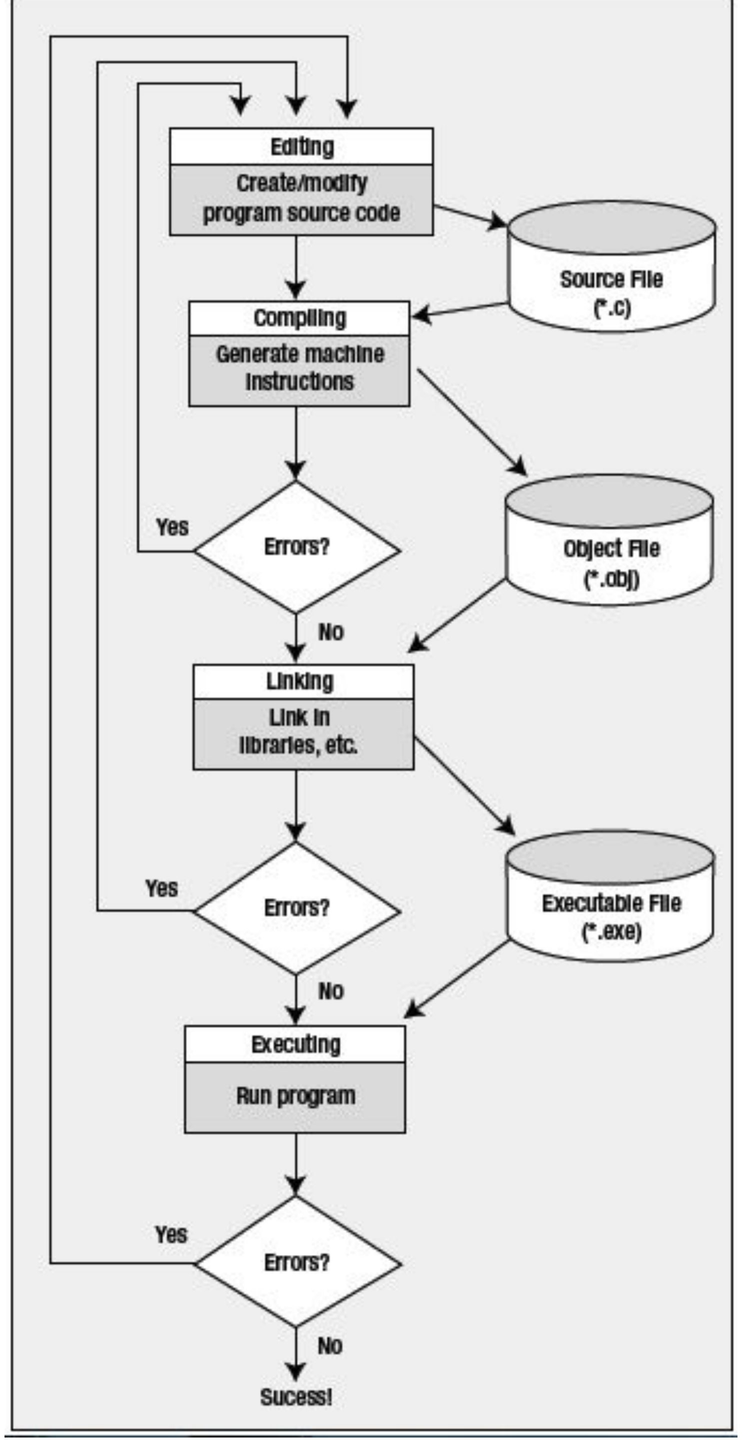
- Performed by a program called the **compiler**
- Translates the preprocessor-modified source code into **object code (machine code)**
- Checks for **syntax errors** and **warnings**
- Saves the object code to a disk file, if instructed to do so (we will not do this).
    - If any compiler errors are received, no object code file will be generated.
    - An object code file will be generated if only warnings, not errors, are received.

3 Stages of Compilation (con't)

Stage 3: **Linking**

- Combines the program object code with other object code to produce the executable file.
- The other object code can come from the **Run-Time Library**, other libraries, or object files that you have created.
- Saves the executable code to a disk file.
  - If any linker errors are received, no executable file will be generated.

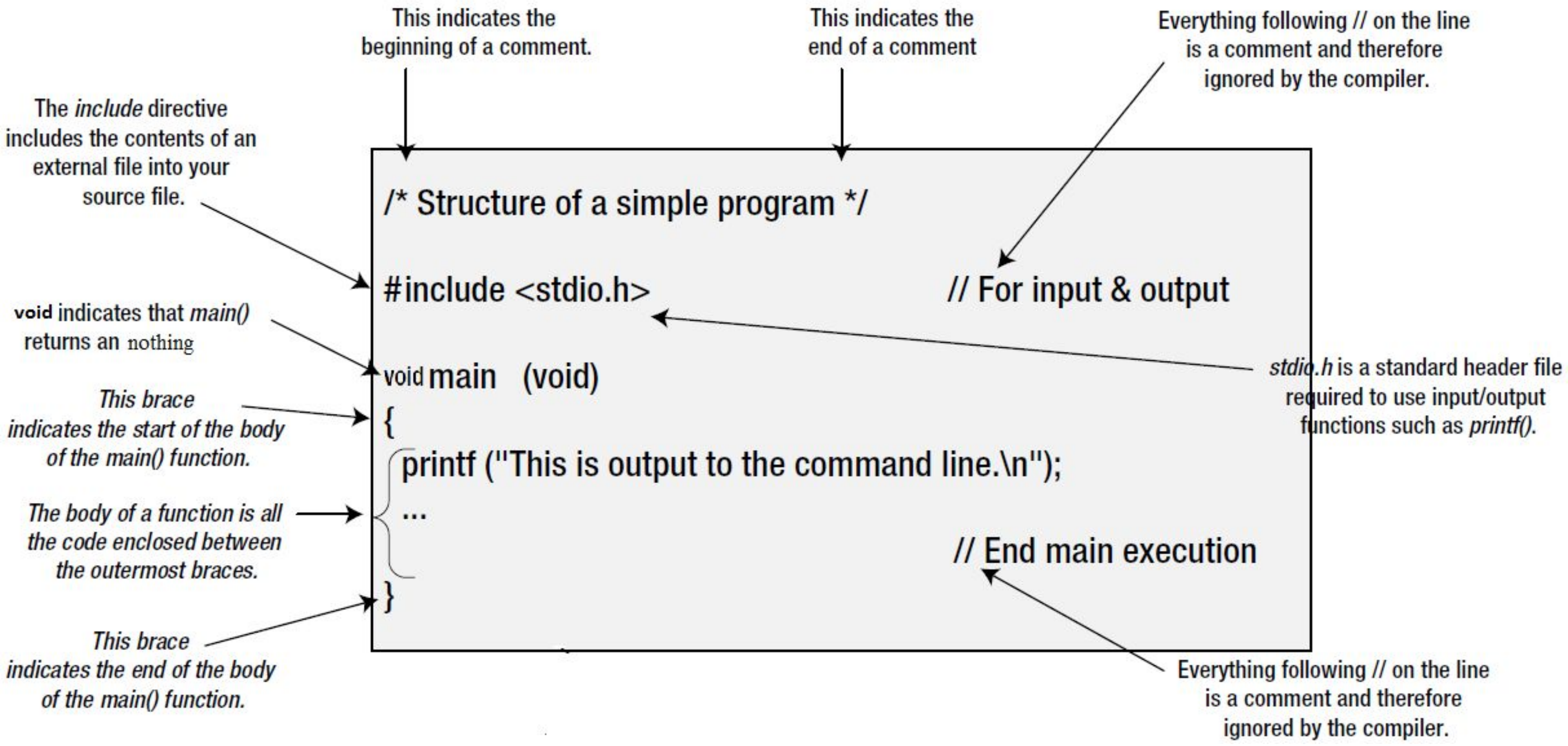
chitectural Diagram



A Simple C Program

- /* Filename: First.c
-    Author: FAST
-    Date written: ?/?/2018
-    Description: This program prints the greeting   “Hello, World!”
- */
- #include <stdio.h>
- void main (void)
- {
-     printf (“This is class CS118!\n”) ;
-     getch( ) ;
- }

Anatomy of a C Program


This indicates the beginning of a comment.
This indicates the end of a comment
Everything following // on the line is a comment and therefore ignored by the compiler.
The *include* directive includes the contents of an external file into your source file.
void indicates that *main()* returns an nothing
This brace indicates the start of the body of the main() function.
The body of a function is all the code enclosed between the outermost braces.
This brace indicates the end of the body of the main() function.
stdio.h is a standard header file required to use input/output functions such as *printf()*.
Everything following // on the line is a comment and therefore ignored by the compiler.

```
/* Structure of a simple program */

#include <stdio.h>                    // For input & output

void main   (void)
{
 printf ("This is output to the command line.\n");
 ...
                                      // End main execution
}
```

Program Header Comment

- A **comment** is descriptive text used to help a reader of the program understand its content.
- All comments must begin with the characters /* and end with the characters */
- These are called **comment delimiters**
- The program header comment always comes first.

Preprocessor Directives

- Lines that begin with a # in column 1 are called **preprocessor directives** (**commands**).
- Example: the **#include <stdio.h>** directive causes the preprocessor to include a copy of the standard input/output header file **stdio.h** at this point in the code.
- This header file was included because it contains information about the printf ( ) function that is used in this program.

stdio.h

- When we write our programs, there are libraries of functions to help us so that we do not have to write the same code over and over.
- Some of the functions are very complex and long. Not having to write them ourselves make it easier and faster to write programs.
- Using the functions will also make it easier to learn to program!

void main (void)

- Every program must have a **function** called **main**. This is where program execution begins.
- main() is placed in the source code file as the first function for readability.
- The **reserved word** “void” indicates that main() **returns** nothing.
- The parentheses following the reserved word “main” indicate that it is a function.
- The reserved word “void” means nothing is there.

The Function Body

- A left brace (curly bracket) -- **{** -- begins the **body** of every function. A corresponding right brace -- **}** -- ends the function body.
- The style is to place these braces on separate lines in column 1 and to indent the entire function body 3 to 5 spaces.

printf (“Hello, World!\n”);

- This line is a C **statement**.
- It is a **call** to the function **printf ( )** with a single **argument (parameter)**, namely the **string** “Hello, World!\n”.
- Even though a string may contain many characters, the string itself should be thought of as a single quantity.
- Notice that this line ends with a semicolon. All statements in C end with a semicolon.

getch();

- getch() is built-in function
- By using this function at end of file, it holds your output screen until you press any character.

Variables

- A variable is a space in the computer’s memory set aside for a certain kind of data and given a name for easy reference.
- Variable are used so that the same space in memory can hold different values at different times.
- All variables must be defined to specify their name and type and set aside storage.

Variables (con’t)

To declare a variable in C, do:

var_type *list variables*;

```
e.g. int i, j, k;
     float x, y, z;
     char ch;
```

Variables (con’t)

Variable Types:

Type	**Memory (byte)**	**Range**
char	1	-128 to 127
int	2	-32,768 to 32,767
long int	4	-2,147,483,648 to 2,147,483,647
float	4	10^{-38} to 10^{38} 7 digits precisions
double	8	10^{-308} to 10^{308} 7 digits precisions

End of Lecture

Any Question ????