# The Program Planning Process and Making Decisions

- Documentation, Advantages of Modularization and Program Modularization
- Local and Global Variables and Constants
- Mainline Logic and Procedural Programs
- Hierarchy Chart and Features of Program Design
- Boolean Expressions, Relational Operators, Logical Operators, Precedence
- Case Structure and Decision Tables

- **Comment**
  - It is helpful if the person who writes the code places some comments in the code to help the reader.
  - Such comments are merely internal program documentation.
  - The compiler ignores these comments when it translates the program into executable code.
  - To identify a comment, C uses two different formats: **block comment and line comment.**
- Block comment:

```
/*
Write a program to find largest number from the given three numbers.
And also write algorithm and draw a flowchart for the same.
*/
```

- Line comment:

```
//Write a program to find largest number from the given three
//numbers. And also write algorithm and draw a flowchart for the
//same.
```

# Documentation

- Documentation refers to all of the supporting material that goes with a program.
- Documentation intended for users
- Documentation intended for programmers
- Program documentation falls into two categories: internal and external.

# Internal documentation

- This documentation consists of program comments, which are non-executing statements that programmers place within their code to explain program statement in English.
- Comment serves only to clarify code, they do not affect the running of a program.

# Output documentation

- This documentation includes all the supporting paperwork that programmers develop before they write a program.
- Because most programs have input, processing, and output, usually there is documentation for each of these functions.

# Modularization

- Programmers seldom write programs as one long series of steps.
- Instead, they break down the programming problem into reasonable units, and tackle one small task at a time.
- These reasonable units are called modules.
- Programmers also refer to them as subroutines, procedures, functions, or methods.
- The process of breaking down a large program into modules is called modularization.

# Advantages of Modularization

- Modularization provides abstraction.
- Modularization allows multiple programmers to work on a problem.
- Modularization allows you to reuse your work.

# Modularization provides abstraction.

- It enable a programmer to see the big picture.
- Abstraction is the process of paying attention to important properties while ignoring nonessential details.
- Abstraction is selective ignorance.
- With abstraction: Do laundry
- Without abstraction:
  - Pick up laundry basket
  - Put laundry basket in car
  - Drive to Laundromat
  - Get out of car with basket
  - Walk into Laundromat
  - Set basket down

# Modularization allows multiple programmers to work on a problem.

- When you dissect any large task into modules, you gain the ability to divide the task among various people.
- Example: word-processing

# Modularization allows you to reuse your work.

- If a subroutine or function is useful and well written, you may want to use it more than once within a program or in other programs.
- Reusability
- Example: a routine that checks the current date to make sure it is valid is useful in many programs written for a business.
- The month is not lower than 1 or higher than 12, the day is not lower than 1 or higher than 31 if the month is 1, and so on.
- A program that uses a personnel file containing each employee's , hire date, last promotion date and termination date can use the date-validation module four times with each employee record.

# Local and Global Variables

# Constants

- Constants are data values that cannot be changed during the execution of a program.
- Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.
- Boolean constants
- Character constants
- Integer constants
- Real constants

# Constant (cont.)

- **Boolean Constants**
- A Boolean data type can take only two values.
- The values are true and false.
- 0(false) and 1(true)

# Constants (cont.)

- **Character Constants**
- Character constants are enclosed between two single quotes.
- We can also use \ (backslash) before the character. The backslash is known as the escape character.
- The escape character says that what follows is not the normal character but something else.

# Constants (cont.)

- The character in character constant comes from the character set – ASCII

| ASCII Character | Symbolic Name |
|---|---|
| Null character | '\0' |
| Alert (bell) | '\a' |
| Backspace | '\b' |
| Horizontal tab | '\t' |
| Newline | '\n' |
| Vertical tab | '\v' |
| Form feed | '\f' |
| Carriage return | '\r' |
| Single quote | '\'' |
| Double quote | '\"' |
| Backslash | '\\' |

# Constant (cont.)

- String constants
- String constants are the constants which are enclosed in a pair of double-quote marks.
- Example
- "good" //string constant
- " " //empty string constant
- "x" //string constant having single character.
- "Earth is round\n" //prints string with newline

# Constants (cont.)

- **Integer constants**
- A integer constant is a numeric constant (associated with number) without any fractional or exponential part.
- Example
- 0, -9, 22

# Constants (cont.)

- **Real constants**
- A floating point or real constant is a numeric constant that has either a fractional form or an exponent form.
- Example
- -2.0 , 0.0000234
- -0.22E-5
- Note: E-5 = $10^{-5}$

# Defining Constant

- Two ways
  - Using **#define** pre-processor.
  - Using **const** keyword.

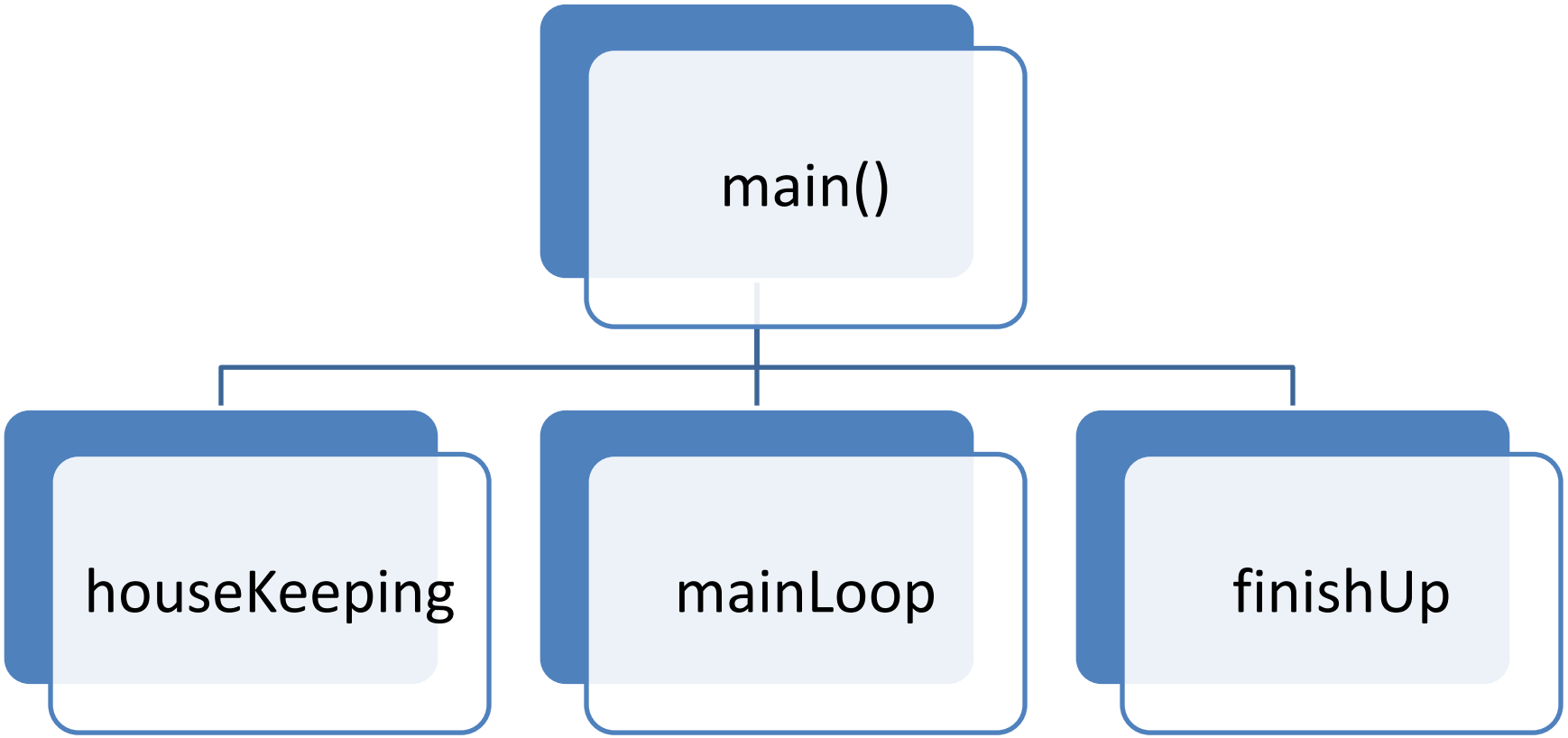
# #define

```
#include <stdio.h>
#define LENGTH 10
#define WIDTH 5
#define NEWLINE '\n'
int main()
{
int area;
area = LENGTH * WIDTH;
printf("value of area : %d", area);
printf("%c", NEWLINE);
return 0;
}
```

# const keyword

```
#include <stdio.h>
int main()
{
const int LENGTH = 10;
const int WIDTH = 5;
const char NEWLINE = '\n';
int area;
area = LENGTH * WIDTH;
printf("value of area : %d", area);
printf("%c", NEWLINE);
return 0;
}
```

# Hierarchy Chart



# Features of Program Design

- You should follow expected and clear naming conventions.
- You should consider storing program components in separate files.
- You should strive to design clear statements within your programs and modules.
- You should continue to maintain good programming habits as you progress in your programming skills.

# Expression

- An **expression** is a sequence of operands and operators that reduces to a single value.
- An **operator** is a syntactical token that requires an action be taken.
- An **operand** is an object on which an operation is performed, it receives an operator's action.

# Expression (cont.)

- Six categories:
  - Primary
  - Postfix
  - Prefix
  - Unary
  - Binary
  - Ternary

# Primary Expression

- The primary expression consists of only one operand with no operator.
- The operand in primary expression can be a variable name, a constant, or a parenthesized expression.
- Example
- B12
- Calc
- 5
- ‘A’
- “welcome”
- (2 * 3 + 4)

# Postfix Expressions

- The postfix expression consists of one operand followed by one operator.
- Example
- a++

# Prefix Expression

- In prefix expressions, the operator comes before the operand.
- Example
- ++a

# Unary Expression

- A unary expression, consist of one operator and one operand.
- Example
- sizeof()
- +a
- -a

# Binary Expression

- Binary expression are formed by a operand-operator-operand combination.
- Example
- 10 * 3
- 10 / 3
- 10 % 3
- 3 + 7
- b = x + 1
- x *= y + 3  same as  x = x * (y + 3)

# Ternary Expression

- ? :
- If Condition is true ? then value X : otherwise value Y
- (condition)? true : false
- Example
- if(5>3) ? “print 5” : “print 3”

# Operator

- An operator is a symbol that tells the compiler to perform specific mathematical or logical functions.
- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators

# Arithmetic operator

- Assume variable **A** holds 10 and variable **B** holds 20

| Operator | Description | Example |
|---|---|---|
| + | Adds two operands. | A + B = 30 |
| - | Subtracts second operand from the first. | A – B = -10 |
| * | Multiplies both operands. | A * B = 200 |
| / | Divides numerator by de-numerator. | B / A = 2 |
| % | Modulus Operator and remainder of after an integer division. | B % A = 0 |
| ++ | Increment operator increases the integer value by one. | A++ = 11 |
| -- | Decrement operator decreases the integer value by one. | A-- = 9 |

# Relational Operators

| Operator | Description | Example A=10 , B=20 |
|---|---|---|
| == | Checks if the values of two operands are equal or not. If yes, then the condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true. | (A <= B) is true. |

# Logical Operators

| Operator | Description | Example<br>A=0 , B=1 |
|---|---|---|
| && | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. | (A && B) is false. |
| \|\| | Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true. | (A \|\| B) is true. |
| ! | Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false. | !(A && B) is true. |

# Assignment Operator

| Operator | Description | Example |
|---|---|---|
| = | Simple assignment operator. Assigns values from right side operands to left side operand | C = A + B will assign the value of A + B to C |
| += | Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand. | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand. | C -= A is equivalent to C = C - A |
| *= | Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand. | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand. | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand. | C %= A is equivalent to C = C % A |

# Conditional / Ternary Operator

- ? :
- If Condition is true ? then value X : otherwise value Y

# sizeof() Operator

- Returns the size of a variable.
- int a;
- sizeof(a);
- where a is integer, will return 4.

# Precedence

- **Precedence** is used to determine the order in which different operators in a complex expression are evaluated.
- **Associativity** is used to determine the order in which operators with the same precedence are evalued in a complex expression.
- Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated.
- Certain operators have higher precedence than others.
- For example, the multiplication operator has a higher precedence than the addition operator.
- For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

# Precedence (cont.)

- Operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.
- Within an expression, higher precedence operators will be evaluated first.

| **Category** | **Operator** | **Associativity** |
|---|---|---|
| Postfix | () [] -- ++ | Left to right |
| Unary | + - ++ -- sizeof | Right to left |
| Multiplicative | * / % | Left to right |
| Additive | + - | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Logical AND | && | Left to right |
| Logical OR | \|\| | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= *= /= %= | Left to right |

# Case Structure

- The **case structure** simply provides a convenient alternative to using a series of decisions when you must make choices based on value stored in a single variable.

# Decision Tables

- Some applications require multiple decisions to produce useful results. Managing all the possible outcomes of multiple decisions can be a difficult task, so programmers use a tool called a **decision table** to help organize the possible combinations of decision outcome.
- A decision table is a problem analysis tool that consists of four parts:
  - Conditions
  - Possible combinations of Boolean values for the conditions
  - Possible actions based on the outcomes
  - The specific action that corresponds to each Boolean value of each condition.

# C - Program Structure

- A C program basically consists of the following parts −
  - Preprocessor Commands
  - Functions
  - Variables
  - Statements & Expressions
  - Comments

```
#include <stdio.h>
int main()
{
        // my first program in C
        printf("Hello, World! \n");
        printf(“FY-B”);
        return 0;
}
```

- The first line of the program ***#include <stdio.h>*** is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.
- The next line ***int main()*** is the main function where the program execution begins.
- The next line **//** will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.

- The next line ***printf(...)*** is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
- The next line **return 0;** terminates the main() function and returns the value 0.

# Compile and Execute C Program

- Open a text editor and add the above-mentioned code.
- Save the file as *hello.c*
- Open a command prompt and go to the directory where you have saved the file.
- Type *gcc hello.c* and press enter to compile your code.
- If there are no errors in your code, the command prompt will take you to the next line and would generate *a.out* executable file.
- Now, type ./*a.out* to execute your program.
- You will see the output *"Hello World"* printed on the screen.