1 introduction

1 INTRODUCTION C is a general-purpose programming language, which features economy of expression, modern control flow and data structures and a rich set of operations. It has been called as “ system programming language" because it is useful for writing compilers and operating systems. Brief History of C ALGOL (Algorithmic Language) - 1960 CPL (Combined Programming Language) - 1963 BCPL (Basic Combined Programming Language) - 1967 developed by Martin Richards this language in turn strongly influenced the development of the next language B. B - 1970 written and developed by Ken Thompson for the first UNIX system on the DEC PDP-7. Both BCPL and B are “ type less" languages. C - 1972 an expansion of B. a programming language designed by Dennis Ritchie in 1972 at AT and T Bell Laboratories. C was originally designed for and implemented on the UNIX operating system on the DEC PDP — II. the new C added something that B do not have: data types Turbo C — 1987 a particular version of C developed by Borland International Corporation. This version is designed to run on various microcomputer systems, namely those which use the operating system MS-DOS, the operating system for the IBM Personal Computers and Compatibles. Definition of terms Interpreters â†’ reads the source code of your program one line at a time and performs the specific instructions contained in that line. Compilers â†’ reads the entire program and converts it into object code, which is a translation of the program source code into a form that can be directly executed by the computer. Compile Time â†’ refers to the events that occur during the compilation process. 1 Object Code â†’ is also referred to as binary or machine code â†’ a translation of the source code of a program into machine code, which the computer can read and execute directly. â†’ Object code is the input to the

linker 2 Source Code â†’ the text of a program that a user can read, commonly thought of as program. â†’ The source code is the input into the C compiler. Run Time â†’ refers to the events that occur while the program is actually executing. Library â†→ collection of pre-written. Syntax â†→ errors that are detected during compiled time. Semantic/ Run-time â†→ errors that are detected during execution time. The Programming Process (implemented using C Language) Components of turbo C Turbo C is more than just a version of the C language. Rather, it includes a complete environment in which to create, test, and run program. This programming environment consists of a number of components: 1. Editor â†→ used to create program source code. 2. Extended C Language â†→ this version o C is significantly extended from the “ base bones" language of Ritchie’s specifications. The extension includes enhancement which make the Turbo C compatible with the new proposed and ANSI Standard. 3. Compiler â†→ used to convert source code into machine code or binary code. 4. Debugger â†→ used for testing program and locating programming errors. 5. Run-Time Environmentâ†→ the capability for running programs within the Turbo C system. 6. User Interface â†→ the various of Turbo C are integrated into a single program which allows you to smoothly from source code entry to compilation to debugging to running without ever leaving the Turbo C environment. Features and Characteristics of C Middle Level Language combines elements of high-level language with the functionalism of assembly language Portable Program it is possible to adapt software written for one type of computer for use on another Ex. Program written for an Apple II+ can be easily moved to an IBM PC. C allows almost all meaningful type conversions. Ex. Char and integer types may be freely intermixed in most expressions. Turbo C has 43

keywords (32 as defined by the ANSI standard and 11 added by BORLAND to allow you to make better use of some special aspects of the PC environment), which are the commands that make-up the Turbo C language C allows manipulation of bit, bytes and addresses (basic elements which the computer functions). Can manipulate data using arrays and pointers. It has its own standard library. Uses of C C was used for systems programming. A system program is part of a large class of programs that form a portion of the operating system of the computer or its support utilities. For example the following are commonly called systems programs Operating System Assemblers Interpreters Compilers Editors Database Managers 1 C PROGRAM STRUCTURE

_____ main( ) { /* refers to the beginning of the program */ ... statement; } /* refers to the end of the program */

_____ 1. /* ... */ Comment 2. Preprocessor Directive â†’ Contains information needed by the program to ensure the correct operation of Turbo C's standard library functions. #include directive â†'(commonly know as macro include) is a preprocessing directive that causes a copy of the file to included at this point in the file when compilation occurs. â†' a #include line can occur anywhere in a file, though it is typically at the head of the file. The quotes surrounding the name of the file are necessary. â†' an include file, is also called " header file", can contain # define lines and other lines. By convention, the names of header files end in . h. Examples: #include " stdio. h" #include #include #include The C system provides a number of standard header files. These files contain the declarations of functions in the standard library, macros,

structure templates and other programming elements that are commonly used. b. #define directive â†’ line can occur anywhere in a program. It affects only the lines in the file that come after it. â†’ Normally, all # define line are placed at the beginning of the file. By convention, all identifiers that are to be changed by the preprocessor are written in capital letters. Examples: #define LIMIT 100 #define PI 3. 1416 Explanation: If the above lines occur in a file that is being compiled, the preprocessor first changes all occurrences of the identifier LIMIT to 100 all occurrences of PI to 3. 14159. The identifier LIMIT and PI are called symbolic constants. The use of symbolic constants in a program make it more readable. More importantly, if a constant has been define symbolically by means of #define facility and used throughout a program, it is easy to change later, if necessary. 3. Declaration Section syntax: ex. int x, y, z; float f= 5. 67; char g; char name[10]; int x= 10; A. Data Types int â†’ contain integral values only, that is values that do not contain decimal places. â†’ variables of type int, can hold integer quantities that do not require a fractional component. Variables of this type are often used for controlling loops and conditional statements. â†’ a whole number consisting of an optional sign (+ or -) followed by a sequence of digit. â†’ occupies two bytes in the address. â†’ can contain no decimal point or fractional part. â†’ cannot contain commas. â†’ ranges from —32768 to +32767 Types of int short int â†’ identical with int â†’ ranges from —32768 to +32767 â†’ occupies two bytes long int â†’ takes up more space and can store large number â†’ ranges from —2147483648 to +2147483647 â†’ occupies 4 bytes unsigned int â†’ cannot be negative from 0 to 65536 unsigned long int float â†’ consists of an optional sign (+ or -), followed by one or more digits , a decimal point, and one or more further digits. â†’

occupies 4 bytes. â† it can include an optional exponent ranging from 3. 4 E —38 to 3. 4 E +38 double â† is a special float which can store more significant digits and have longer exponent. â† occupies 8 bytes in the memory â† ranges from 1. 7E-308 to 1. 7E+308 with 15 digits accuracy. char â† can be used to contain a single letter, digit, punctuation mark or control symbol recognized by the computer. Written enclosed within single quotation marks, but literals strings are enclosed in double quotation marks. a character may be assigned an integer value between —128 and +127. â† unsigned char data type may be assigned an integer value from 0 to 255. Ex.: char c; char B = '*'; char a[30]; char a = " apple"; void â† valueless Three uses of void: to declare explicitly a function as returning no value; to declare explicitly a function having no parameters; to create generic pointers. B. Identifiers The names that are used to reference variables, functions, labels and various other user-defined objects. Sequence of letters, digits, and the special characters, which is called an underscore An identifier in C can vary from one to several characters The first character must be a letter or an underscore with subsequent characters being letters, numbers or underscore. In Turbo C, the first 32 characters of an identifier name are significant In C, upper and lower case are treated as different and distinct from one another. Ex.: studname, Studname, StudName are three separate identifiers. An identifier may not be the same as a Turbo C keyword, and it should not have the same name as a function. Correct Incorrect count 1count test123 hi! there 4. The Body of the Program The body of the program starts with the reserved word main( ) and is enclosed with { and }. block of code is logically connected group of program statements that is treated as a unit. is created by placing a sequence of statements between

opening and closing curly braces. NOTES: All C keywords must be lowercase. It may not be used for any other purpose in a C program. All program statements in C must be terminated by a semicolon (;). Comments may be placed anywhere in a program and are enclosed between two markers. The starting comment marker is /* and the ending comment marker is */. Keywords/Reserved Words Words that have a special meaning in C and cannot be used for other purposes. All upper in lowercase. auto break case char const continue default do double else enum extern float for goto if int long register return short signed sizeof static struct switch typedef union unsigned void volatile while The Turbo C Extended Keywords asm interrupt cdecl _ds _ss _cs near far pascal _es huge Variables are identifiers which can be assigned a value within a program 32 characters are valid like letters, digits, and underscore. Declaring Variables: Syntax: Data — type variable list; â†’ list of variables separated by commas Example: int a, b, c; char x, y; double p; float average; char name[30]; Variable initialization â†’ the process of assigning starting values to the variables. Several Ways to Initialize a Variables: By using an assignment statement â†’ maybe used to assign to variables of any type. Syntax: variable-name = expression Example: x = -10; ch = ‘ A’; The symbol = C’s assignment operator By using function; (scanf) Example: scanf(“%d", &x); scanf(“%lf", &y); By assigning value during declaration Example: int x = 13; char y = ‘ a’; double a, b = 120. 00; Global and Local Variables Global Variables â†’ are initialized only at the start of the program. All global variables are initialized to zero if no other initializer is specified. Local Variables â†’ are initialized each time the function in which they are declared is entered. Constants refers to fixed values that may not be altered by the program. Syntax: type variable_name = constant;

Example: char ch = ‘ a’; int num = 0; float bal = 123. 3; Turbo C constant can be of any of the basic data types: character constants are enclosed between single quotes. Example: ‘ b’, ‘%%’ integer constants are specified as number without fractional components. Example: 90, -600 floating-point constant are required the use of decimal point followed by the numbers fractional components. Example: 12. 134 string constant consists of a phrase contained/enclosed within double quotes. String constants are used in printf statements to introduce text messages into a program. Examples: char ‘ a’ ‘ 9’ int 10. 1 123 float 123. 23 double 123. 23 12323333 string “ Elsa" Declared Constants â†’ constants which are assigned a name declared constant are defined using a # defined declaration at the beginning of the program. This declaration is introduced by the word # define followed by a constant name and its value. Examples: # define total amount 1000 # define grapes 90. 00 # define titik ‘ E’ # define m 14 # define salita “ good day" Access Modifiers C has two type of modifiers that are used to control the way in which variables may be accessed or modified. 1. const â†’ variables declared under this modifier cannot be changed during program execution. However, you may give an initial value before the start of the program. Syntax: const datatype variable = value; Example: const float version = 3. 20; Interpretation: This creates a float variable called version that may not be modified by your program. A constant variable will receive its value either from an explicit initialization or by some means dependent on the hardware. 2. using # define â†’ the # define declaration is used at the beginning of a program. Like const, #define is also used in declaring constants. Syntax: # define variable value Example: # define amount 56. 89 Note: it is necessary to tell a computer the type of a declared constant. C determined the data

type from the assigned value. Remember that although, const and # define defines constant, the two modifiers have different syntax. 1 Type Modifiers A modifier is used to alter the meaning of the base type to more precisely fit the needs of various situation. With the exception of type void, the basic data type may have various modifiers preceding them. Example: signed long unsigned short Possible Combinations of C’s Basic Types and Modifiers | Type | Bit Width | Range | | char | 8 |-128 to 127 | | unsigned char | 8 | 0 to 255 | | signed char | 8 |-128 to 127 | | | | | int | 16 |-32768 to 32767 | | unsigned int | 16 | 0 to 65535 | | signed int | 16 |-32768 to 32767 | | short int | 16 |-32768 to 32767 | | unsigned short int | 16 | 0 to 65535 | | signed short int | 16 |-32768 to 32767 | | long int | 32 |-2147483648 to 2147483649 | | signed long int | 32 |-2147483648 to 2147483649 | | | | | float | 64 | 3. 4E —38 to 3. 4E+38 | | double | 64 | 1. 7E —308 to 1. 7E+308 | | long double | 80 | 3. 4E-4932 to 1. 1E+4932 | Assignment Statement Is the process of assigning a value to a variable. In C, we use the equal sign (=) as the assignment operator. A = C + r * (9 / p); How to Assign Values to Variables The variable must be on the left side of the equal sign. Values must be on the right side of the equal sign. Any expression can be assigned on the variable. Expressions are not allowed on the left side of the assignment operator. 1 OPERATORS Operators is a symbol that tells the compiler to perform specific mathematical, relational or logical manipulations. a symbol which represents an operation to be performed. a. Arithmetic Operators Symbols Functions - subtraction + addition * multiplication / division % modulus -- decrementing x-- same as x= x-1 ++ incrementing x++ same as x= x+1 Important Notes: The operators +, -, *, and / all work the same way in C as they do in most computer languages. When / is applied to an integer or character, any

remainder will be truncated. Example: 13/6 = 2 3 / 4 = 0 10/3 = 3 11/2 = 5 The modulus division operator, % yields the remainder of an integer division. % cannot be float on type or double. Example: 13 % 2 = 1 -25 % 2 = -1 5 % 3 = 2 6 % 2 = 0 When an increment or decrement operator precedes its operand, C performs the increment or decrement operation prior to using the operand’s value. If the operator follows its operand, C uses the operands value before incrementing or decrementing it. Example: X = 12; Y = ++X; â†’ X is 13; Y is 13 X = 12; Y = X++ â†’ X is 13; Y is 12 Increment and Decrement Operators Increment â†’ to add exactly one to the value of a variable Decrement â†’ to subtract exactly one from the value of a variable. ++ increment operator X++ post increment --X pre increment - - decrement operator Example: 1. X = X + 1 or X+= 1 is the same with X++ 2. X = 5, Y = 12 W = Y + X++ W = Y + X; 17 X = X + 1; 6 W = Y + ++ X X = X + 1; 6 W= Y+ X; 18 W = Y + X-- W = Y + X; 17 X = X - 1; 4 W = Y + -- X X = X - 1; 4 W= Y+ X; 16 Precedence of Arithmetic Operators highest ++ - - - (unary) * / % lowest + - b. Relational Operators â†’ used to determine the relationship of one quantity to another. â†’ refers to the relationship values can have with one another. Symbols Functions Examples > greater than x > y >= greater than or equal to num >= 100 < less than x < y 9 ? printf(“ Yes"): printf(“ No"); Interpretation: Exp1, epx2 and exp3 are expressions; exp1 is evaluated, if true then exp2 is evaluated and its value becomes the value of the expression. The value of a ? expression is determined this way: Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression. If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression. 2. , the comma operator is used to string together several expressions. It is used to string together several

expressions. The left side of the comma operator will always be evaluated as void. This means that the expression on the right side will become the value of the total comma separated expression. Example: Y = (X= 4, X++, X *3); X = 4 X = 5 X = 15 Finally Y = 15 x = (y= 3, y+1); The value of x is 4 y = 20; x = (y= y-5, 30/y); The value of x is 2 Hierarchy of C Operators ( ) highest ! + + - - (type) * / % + - < >= == != && || ? = *= /= %= += , lowest Expression Operators, constant and variables are the constituents of expressions Are combinations of operators, constants and variables. Expression Evaluation: v = (6 / 2 * 3) >= ((6/2 % 3) + 6 * 2) v = (5 == (3 + 1)) || ((6 % 3) >= (5 /2)) && (5 != 6) A = 1; B= 2; C= 3; X = !((A > C) && (B >= 3)) || (C < 4) && ((6 = 75) printf(“ Passed"); Interpretation: If condition evaluates to true, then statementT is executed; otherwise statement is skipped and control passes to the next statement. Note: Usually, the condition in an if statement is a relational, logical, equality expression, a condition from any domain is permissible. Where appropriate, compound statement should be used to group of statements under the control of a single if condition. The code be written to be more efficient and more understandable by using a single if statement with a compound statement for its body. 6 IF — ELSE Statement The general form of the if - else statement is Syntax : if (condition) statementT; else statementT; Example: if (grade >= 75) printf(“ Passed"); else printf(“ Failed"); Interpretation: If condition evaluates to true, then statementT is executed; otherwise it is skipped and statementF is executed. In both cases, control passes to the next statement. Note: Block of statements in C is surrounded by curly braces { } else is optional 7 if … else if Statements A common programming constructs is the if … else if statement. It look like this: Syntax : if (condition) Statement1; else if

(condition) statement2; else if (condition) statementn; : else statemente; Example: if (x > 0) printf(“ x is positive"); else if (x < 0) printf(“ x is negative"); else printf(“ x is zero"); Interpretation: The conditions here are evaluated in sequence until a true condition is reached. If a condition is true, the statement following it is executed, and the rest is skipped. If a condition is false, the statement following it is skipped, and the next condition is tested. If all conditions are false, then the statemente following the final else is executed. Note: The last else is optional. 8 Nested if — Else Statements A common programming constructs is the nested if … else statement. It look like this: Syntax : if (condition) if (condition) statement2; else statementn; else statemente; Example: if (x == 50) if (y >= 120) { sum = x+y; printf(“ the sum of x and y is %d", sum"); } else { diff = x - y; printf(“ the difference of x and y is %d", diff"); } printf(“ x is zero"); else printf(“ Next time"); Sample Program No. 1 /* Program to determine if a number is positive or negative*/ #include " stdio. h" main() { int num; clrscr(); printf(" Enter an integer "); scanf("%d",νm); if (num > 0) printf("%d is a positive integer", num); else printf("%d is a negative integer", num); getch(); } Sample Program No. 2 /* Program to determine if a number is odd or even*/ #include " stdio. h" main() { int num, rem; clrscr(); printf(" Enter a number "); scanf("%d",νm); rem = num % 2; if (rem == 0) printf("%d is an even number", num); else if (rem != 0) printf("%d is an odd number", num); getch(); } 1 Sample Program No. 3 /* Program Arithmetic Operations */ #include " stdio. h" main() { int f, s, total= 0; char optr; clrscr(); printf(" Enter 1st number : "); scanf("%i", &f); printf(" Enter the operator : "); scanf("%s", &optr); printf(" Enter 1st number : "); scanf("%i", &s); if (optr=='+') total= f+s; if (optr=='-') total= f-s; if (optr=='*') total= f*s; if

(optr=='/') total= f/s; if (optr=='+' || optr=='-' || optr=='*' || optr=='/') printf("%d %c %d = %d", f, optr, s, total); else printf(" Operator not valid!!!"); getch(); } 9 SWITCH Statement is a multiway conditional statement generalizing the if-else statement. is a built-in multiple branch decision statement. A variable is successively tested against a list of integer or character constants. When a match is found, a statement or block of statement is executed. Syntax: switch(variable/ expression) { case constant1 : statement; break; case constant2 : statement; break; case constant3 : statement; break; : default : statement; } Example: switch(QUIZ) { case 10 : case 9 : printf"A"); break; case 8 : printf"B"); break; case 7 : printf"C"); break; case 6 : printf"D"); break; case 5 : case 4 : case 3 : case 2 : case 1 : case 0 : printf"F"); break; default : printf(“ Input out of Range"); } Interpretation: The switch expression maybe any expression which evaluates to an int, conditional, char. The case list must consists of constants whose matches that of the switch expression. After the statements for a case, a keyword break maybe used. The break keyword means that at that point, execution should jump to the end of the switch statement. The switch statement is terminated by curly bracket (}). This switch constant can be used to specify an action to be taken if the value of the switch expression does not match any of the listed values. Note: The case list cannot include variables or expressions. The optional switch constant default. The statements following a case label may be one or more C statements, so you do not need to make multiple statements into a single compound statement using braces. If no case label value matches the expression, the entire switch statement body is skipped unless it contains a default label. If so, the statements following the default label are executed. There are 3 important things to know about

switch statement: Switch differ from the if in that switch can only test for equality, where as the if, can evaluate relational or logical expression. No two case constant in the same switch can have identical values. A switch statement enclosed by an outer switch may have case constants that are the same. If char constant are used in the switch, they are automatically converted to their integer values. Switch statement is often used to process keyboard commands, such as menu selection. 1 Sample Program No. 1 /* Program Simulation */ #include " stdio. h" int x; main() { clrscr(); puts(" Enter a number: "); scanf("%d",&x); switch(x) { case 1: puts(" now, there's"); break; case 2: puts(" only but a few"); case 3: puts(" good men..."); puts(" it's true!!!"); break; default: puts(" no good men at all..."); } getch(); } 2 Sample Program No. 2 /* Program Simulation */ #include " stdio. h" int x; main() { clrscr(); puts(" Enter a number: "); scanf("%d",&x); switch(x) { case 1: case 2: case 3: printf("%d Indian", x); break; case 4: case 5: case 6: printf(" 4 little 5 little 6 little Indian"); break; } getch(); } 3 Sample Program No. 3 switch(optr) { case ‘+’ : total = f + s; break; case ‘-’ : total = f - s; break; case ‘*’ : total = f * s; break; case ‘/’ : total = f / s; break; default : printf(“ operator not valid"); } 10 ITERATION / REPETITION CONTROL STRUCTURES 11 Loops Allow a set of instructions to be repeated until a certain condition is reached. 12 FOR LOOP The for loop will continue to execute as long as the condition is true. Once the condition becomes false, program execution will resume at the statement following the for. Syntax: for(initialization expression; loop repetition condition; update expression) statement; where: (the for statement allows many variants, but there are three main parts) initialization expression â†’ is an assignment statement that is used to set the loop-control variable. Loop repetition condition â†’ is a relational

expression that determines when the loop will exit by testing the loop-control variable against some value. Update expression â†’ defines how the loop-control variable will change each time the loop is repeated. Example: for (x= 100; x!= 65; x-= 5) { z= sqrt(x); printf(“ The square root of %d is %f", x, z); } Interpretation: First, the initialization expression is executed. Then, the loop repetition condition is tested. If it is true, the statement is executed, and the update expression is evaluated. The loop repetition condition is re-tested. If the loop repetition condition is found to be false, the for loop is exited. Note / Programming style: These three major sections must be separated by semicolons. You may place all three expressions in the for heading in a single line, specially if all three expressions are very short.] The body of a for loop may be a compound statement. 1 Sample Program No. 1 /* Program on Factorial */ #include main() { int f, x, ans; clrscr(); printf(" Enter a number : "); scanf("%d",&f); ans= 1; for(x= f; x>= 1; x--) ans= x*ans; printf(" The factorial of %d is %d", f, ans); getche(); } 2 Sample Program No. 2 /* Program Exponential */ #include main() { int b, e, x; double ans; char sagot; clrscr(); printf(" Enter Base : "); scanf("%d",&b); printf(" Enter Exponent: "); scanf("%d",&e); ans= 1; for(x= 1; x