VIKAS®

2nd Edition

TIPS AND TRICKS ON C PROGRAMMING

A QUICK GUIDE ON
HOW TO CRACK INTERVIEWS

SUSANT K ROUT

Tips & Tricks on C Programming

A Quick Guide on
How to Crack Interviews

This Page has been intentionally left blank

Tips & Tricks on C Programming

A Quick Guide on
How to Crack Interviews

Second Edition

Susant K Rout
Founder
LIT Susant K Rout's Center of Excellence
Bhubaneswar

VIKAS®

VIKAS® PUBLISHING HOUSE PVT LTD

VIKAS® PUBLISHING HOUSE PVT LTD
E-28, Sector-8, **Noida**-201301 (UP)
Phone: 0120-4078900 • Fax: 0120-4078999
VIKAS® Registered Office: 576, Masjid Road, Jangpura, **New Delhi**-110 014

E-mail: ***helpline@vikaspublishing.com*** • Website: ***www.vikaspublishing.com***

• **Ahmedabad**	: 305, Grand Monarch, 100 ft Shyamal Road, Near Seema Hall, Ahmedabad-380 051 • Ph. 079-65254204
• **Bengaluru**	: First Floor, N.S. Bhawan, 4th Cross, 4th Main, Gandhi Nagar, Bengaluru-560009 • Ph. +91-80-22204639, 22281254
• **Chennai**	: E-12, Nelson Chambers, 115, Nelson Manickam Road, Aminjikarai Chennai -600029 • Ph. +91-44-23744547, 23746090
• **Hyderabad**	: Aashray Mansion, Flat-G (GF), 3-6-361/8, Street No. 20, Himayath Nagar, Hyderabad • Ph. +91-40-23269992, Fax. +91-40-23269993
• **Kolkata**	: 82, Park Street, Kolkata-700017 • Ph. 033-22837880
• **Mumbai**	: 67/68, 3rd Floor, Aditya Industrial Estate, Chincholi Bunder, Behind Balaji International School & Evershine Mall, Malad (West), Mumbai-400064 • Ph. +91-22-28772545, 28768301
• **Patna**	: Flat No. 204, Temple Tower, Sabjibagh, Birla Mandir Road, Patna-800004 • Ph. 94310 15579

Tips and Tricks on C Programming
ISBN: 978-93259-7158-5

First Published 2011
Second Edition 2014

Vikas® is the registered trademark of Vikas Publishing House Pvt Ltd
Copyright © Author

All rights reserved. No part of this publication which is material protected by this copyright notice may be reproduced or transmitted or utilized or stored in any form or by any means now known or hereinafter invented, electronic, digital or mechanical, including photocopying, scanning, recording or by any information storage or retrieval system, without prior written permission from the publisher.

Information contained in this book has been published by VIKAS® Publishing House Pvt Ltd and has been obtained by its Authors from sources believed to be reliable and are correct to the best of their knowledge. However, the Publisher and its Authors shall in no event be liable for any errors, omissions or damages arising out of use of this information and specifically disclaim any implied warranties or merchantability or fitness for any particular use. Disputes if any are subject to Delhi Jurisdiction only.

Objective of Writing This Book

Why Tips and Tricks?

- Every chapter in C contains some rules and regulations. If we know the rules of a chapter, it becomes very easy to understand the topics. So all the rules and regulations of C have been taught in this book in the form of "tips and tricks" which help the student in:
  - ➢ understanding the topics smoothly,
  - ➢ preparing him for the written examination,
  - ➢ preparing him for campus placement.
- The book contains some tricky questions which are commonly asked in the classroom, during viva voce and the interviews. Some questions focus at bringing out the subtle difference between similar looking concepts. These are "differentiate between" type. The objective is to bring in crystal clarity in the subject.
- "Secret" problems in the book are intended to help the student in:
  - ➢ exploring and developing the inherent skills of a ready programmer,
  - ➢ motivating as a sesoned programmer to explore unknown concepts of C programming.
- This new edition also contains 100 short-answer questions in the form of Chapter 4.

As you read the book on, you realize that C is truly a versatile language, and perhaps even after years of experience, we all are beginners in C. There is so much more to explore, learn and master!

This Page has been intentionally left blank

Contents

14. What is the major difference between printf and sprintf?
15. What is the major difference between precedence and associativity?
16. What is the major difference between reference and dereference in pointer?
17. What is the major difference between writing prototype inside and outside of a function?
18. What is the difference between if else and switch case?
19. What is the major difference between sizeof and strlen?
20. What is the major difference between break and continue?
21. What is the major difference between macro and typedef?
22. What is the major difference between lvalue and rvalue?
23. What is the major difference between = and == ?
24. What is the major difference between scanf and sscanf?
25. What is the major difference between process and program?
26. What is the major difference between memcpy and memmov?
27. What is the major difference between text file and binary file?
28. What is the major difference between NUL and NULL?
29. What is the major difference between while and do while?
30. What is the major difference between array and string?
31. What is the major difference between & and &&?
32. What is the major difference between '.' and '->'?
33. What is the major difference between i++ and ++i?
34. What is the major difference between signed and unsigned integer?
35. What is the major difference between getch and getche?
36. What is the major difference between void main and int main?
37. What is the major difference between external variable and global variable?
38. What is the major difference between array and linked list?
39. What is the major difference between const and static?
40. What is major difference between strcpy and strncpy?
41. What is major difference between stdout and stderr?
42. What is the difference between function call and macro calls?
43. What is the difference between writing static to a local variable and a global variable?
44. What is the difference between static and external variable?

11. How to change the byte order of an integer?
12. How do we write a general swap macro, which can swap any types of data?
13. How to check the Endianness of the system?
14. How to find the maximum number of elements present in any array?
15. Given are two arrays A and B. Array 'A' contains all the elements of array 'B' along with an extra element. Write the program to find the extra element.
16. Write a program to display "Hello" without any semicolon.
17. Given is an array of numbers. Except one number all the numbers occur twice. Write a program to find which numbers occur only once.
18. Write a one-line statement to check the number is power of 2 or not.
19. Write a C program to find the smallest of three integers without using any comparison operators.
20. Write a program to multiply any two numbers without using multiplication '*' operators.
21. Write a program to compare between any two integer without using '==' and '!=' operator.
22. Find the maximum and minimum of any two numbers without using any condition or loop.
23. Display all the numbers from 1 to 100 without any relational operators.
24. Write a program and print "Hello world" repeatedly without using loop, recursion, and any control structure.
25. Write a program to count the number of vowels and consonants in a given string without using loop and `if` statement.
26. Write a program that will display spiral of `n*n` numbers, using constant space without using array.
27. Convert a floating point value to a string and format its textual representation.
28. Write a function in a different way that will return f (7) = 4 or f (4) = 7.
29. Write a program to check whether given unsigned number is a multiple of 3 or not without using division or modulo operators.
30. Write a program to form a maximum value number from the digits of a given number. For example, for 374 it is 743.

31. Write a program to sort an array without using any condition.
32. How to return the largest sum of two contiguous integers in the array?
33. Write an efficient code for extracting unique elements from a sorted list of an array.
34. Write a C program to reverse the words in the given string, such as “man is mortal” to “mortal is man” without using extra space.
35. You have an unsorted array of some million random positive integers with no duplicates. Find two integers from this array which add up to 50.
36. Write a program to find out the square root of a number without using sqrt library function.
37. Write a program which produces its source code as output.
38. Write a program to read n lines from a file.
39. Write a program to implement circular left shift by specified number of rotations.
40. How to generate floating point random numbers?
41. How would you find the last nth element of a linked list?
42. How to delete a node from a linked list if address is given, but its previous address is not given?
43. How to write a function that can reverse a linked list?
44. How to check whether a given linked list is circular or not? (Use only one pointer to increase efficiency)
45. Write a recursive function to find the maximum number from an arbitrary array.
46. Write a program to find the largest palindrome of a given string.
47. Write a method to find out the k^{th} largest element in an array having n number of distinct elements.
48. What is the efficient way to find the middle element of a linked list?
49. Write a program to find whether a loop is present in a linked list or not.
50. Write a program without using any control structure, to check whether the given number is EVEN or ODD.
51. How to write a program to check whether the given name is a macro or function?
52. What can you do in C but not in C++?
53. Calculate the generic root of a number in a one-line statement.
54. Write a one-line statement to find (-1)^n.

Pointers

25. Which concept is called “dereference”?
26. What does maximum address by a pointer refer in TC and GCC compilers?
27. What is null pointer?
28. What is wild pointer?
29. What is dangling pointer?
30. What are the different reasons for segmentation fault in C?
31. What are near and far pointers in C?
32. What are the applications of pointers in C?
33. When does core dump occur in C ?
34. What is the difference between malloc and calloc functions?
35. What is meant by static and dynamic memory allocation in C?
36. Why is pointer arithmetic required?
37. Define “memory leak” in a one-line statement.

Arrays

38. What is the relation between arrays and pointers?
39. What is the difference between array and linked list?
40. What is the purpose of double-dimensional array?
41. Which of the two is a faster searching mechanism—“binary search” or “linear search”?
42. Traversing double-dimensional array using row order or column order, which is one is the faster mechanism?
43. If base address of a double-dimensional array is given, how do we get the address of a particular element in an array?
44. What will be size of any array in C?
45. What is the application of an array?
46. What is the difference between character array and string?
47. What is major difference between strcpy and strncpy?
48. What is the difference between char s[] and char *s?

Functions

49. What is a function?
50. What is the use of library function?
51. What is the difference between library function and system calls?
52. What is the disadvantage of writing the function body below or above the main function?
53. How can we find the size of the function?

54. What is function recursion?
55. Which data structure is implemented in function recursion?
56. What is the difference between return and exit?
57. What is the major difference between stdout and stderr?
58. What is the major difference between stream and buffer?
59. When is a "copy of the variable" created?
60. How do we access a variable in other functions without parameter technique?
61. What is the difference between 'call by value' and 'call by address'?
62. What is the difference between static library and dynamic library?
63. What is the difference between dynamic library and dynamic loaded library?
64. What is the difference between void main and int main?

Storage Classes

65. What is a storage class?
66. What is the difference between program scoping and file scoping?
67. What is the difference between writing static as local variable and global variable?
68. Why does a pointer not refer to a register variable?
69. What is the use of static storage class in C?
70. What are the different scope rules in C?

Preprocessors

71. What are the different preprocessor directives in C?
72. Which of the following produce the object file:
    (a) Preprocessor (b) Compiler (c) Assembler (d) Linker?
73. What is macro?
74. What is the difference between macro call and function call?
75. What is the difference between writing header file using " " and < > ?
76. What is the difference between startup and exit pragma?
77. What is the use of C preprocessor?

Structures and Unions

78. What is the self-referential structure?
79. What is slack byte of the structure?
80. What is the difference between a structure and a union?
81. Why cannot two structure variables be compared?
82. What do you mean by "active data member" of a union?

83. What is the application of structures?
84. What is the application of unions?
85. What is the use of bit-field?

Files

86. What is the difference between EOF and End Of File?
87. What is the difference between “w” and “w+”?
88. What are the different types of files?
89. What are the different modes of file opening?
90. What is the main difference between fseek and lseek?
91. What is the difference between file and directory?
92. What is file system?
93. What are the active file pointers in C?
94. What is the maximum size of a file?
95. What is the difference between file and FILE?

Command Line Arguments

96. What is the use of a command line argument?
97. How are “main(void)” and “main(int)” different?
98. Explain c and v in argc and argv with their purpose.
99. How can we redirect the output of one program as input of another program?
100. What is the application of environment vector in command line argument?

1. Tips & Tricks

DATA TYPES / CONSTANTS AND VARIABLES

1. Data will be stored in memory in ABCD or DCBA format that depends on Endianness of the system.
2. In little-endian systems data is stored in DCBA order.
3. In big-endian data is stored in ABCD order.
4. Cycle is present in integer and character data type but is not present in float or double types.
5. Increment the value of a float or double variable beyond its maximum range that is +INF and beyond it minimum range is -INF.
6. The minimum octal character constant is '\000' and maximum octal character constant is '\377'.
7. Float data is always stored in memory in mantissa and exponent format.
8. Enum data type creates a sequence set of integral constants.
9. There is no cycle present in enum data type.
10. BCPL is a type-less language.
11. When a language is able to produce a new data type it is called extensibility.
12. `Tyepdef` creates a new name but does not create a new type.
13. The process of byte ordering is known as endianness.
14. 32 bits recurring binary of a float is always lesser than 64 bits recurring binary of a float.

15. Unsigned modifier always makes double the range of a type.

16. All constants in C are R value category of objects.

17. Constant variables, array name and function name; enum constants are Rvalue category of objects and they should always be placed in the right side of an equal-to operator.

18. All escape sequence characters are octal character constants.

19. The size of the null string constant is 1 byte.

20. '\0' is null character constant whose ASCII value is 0.

21. Void type is used in three different places, such as:

    No argument to a function
    No Return type
    Generic Pointer

22. The number of constants and variables in C are equal.

23. Size of integer type depends on compiler or wordlength of the processor.

24. Float follows 23 bits mantissa, 8 bits exponent and 1 bit sign, but double follows 52 bits mantissa, 11 bits exponent and 1 bit sign.

25. C Language supports 256 characters. Out of them 127 are ASCII (26+26 alphabets, 10 digits, 32 special characters, 33 control characters) and 129 are Extended ASCII characters.

26. Every variable and function allocates memory in load time and run time, but how much memory will be allocated and where will be allocated that is decided during compile and linking time.

OPERATORS

1. Precedence and associativity of an operator is only implemented if expression is represented in INFIX notation.
2. Precedence decides which operation will be performed first in an expression.
3. Associativity decides the order of evaluation (left to right or right to left) if more than one operator enjoys same precedence.
4. The operator which requires three operands is called ternary operator.
5. ? : is a conditional ternary operator.
6. Associativity of a comma (,) operator is left to right.
7. # is a string forming operator that converts a non-string data to string and is used only in preprocessor directives.
8. There is no operator in C to calculate the power of a number.
9. Bit-wise operators only work in integral type but does not work in float type data.
10. Return value of relational and logical operators is 0 or 1.
11. x++ and ++x both are always same.
12. If ++ or - - operators occur more than once, then first two make a pair and are evaluated, then the result is added in next, then next, and so on

4

CONTROL STRUCTURES

1. Hanging `if` is not allowed in C.
2. The `if` statement is used for checking range and the yes/no problem.
3. `else` is the optional part of `if`.
4. Execution of `switch case` starts from the `match case`.
5. Use `break` to prevent the falling of control from one case to another case.
6. If none of the cases are satisfied then default case will be satisfied.
7. Case character only can be integral constant.
8. `Switch case` only checks integers and not float.
9. `Continue` can not be the part of the `switch case`.
10. `Switch` without `case` is of no use (no output no error).
11. Duplicate `case` is not allowed in C.
12. `Switch case` is the replacement of `if...else` statement.
13. `Switch case` is used for selection control statement.
14. Loop counter can be `int`, `char` and `float`.
15. If loop counter is not initialized in a loop, that is called odd loop.
16. Loop is an alternate of function recursion and function recursion is an alternate of loop.
17. `for` loop is suitable for finite loop, `while` loop is

suitable for unknown loop and `do while` loop is suitable for looping at least once.

18. loop counter has its own block scoping.
19. Drop all the parts of `for` loop makes it an infinite loop but drops all part of `while` and `do while` loop and it gives an error.
20. `break` in loop transfers control outside the loop.
21. `continue` transfers the control to the beginning of loop.
22. If loop counter exceeds its range, it will be an infinite loop.
23. Every loop has a start value, step value and stop value.
24. If loop loses its step value, it will be an infinite loop.
25. Loop is used for software delay and repeating some part of the C program.
26. Loop is faster than function recursion.
27. Only one statement falls under the default scope of the control structure.

POINTERS

1. When a pointer refers to 0 (zero) address, it is called null pointer.
2. Generic pointer can not be dereference.
3. Arithmetic is not allowed in generic pointer.
4. Return type of `malloc` and `calloc` function is generic pointer.
5. Address + 1 = next address of its type.

6. When a pointer refers an invalid address, then it is called wild pointer.
7. When a pointer loses reference, it is called memory leak.
8. Size of a pointer in gcc is 4 bytes.
9. In TC, when a pointer works within 64KB memory, then it is called near pointer. But when the pointer works beyond 64KB memory, then it is called far pointer.
10. Dereferencing to a wild pointer is called core dump.
11. When the pointer address becomes invalid in a program, then it is called dangling pointer.
12. Function name and array name are known as constant pointers.
13. When more than one pointers refer to the same location and one of them modifies the resource, at the same time others are waiting, then that pointer must be a restrict pointer.
14. Dynamic memory allocation is possible using pointer.
15. Random Access of any memory cell of a variable will use pointer.
16. A function can be called or invoke using a function pointer.
17. Dereferencing to a null pointer is known as null pointer assignment.

ARRAYS

1. An array is a variable which only holds similar types of data elements.
2. Array is used to prepare a list.

3. A list can be prepared using an array and linked list. If the list size is known in advance then you can take an array, but if it is unknown you can take a linked list.
4. Limitation of an array is:
   - Unused memory of an array cannot be used.
   - Array size cannot be changed in run time.
   - One array cannot be assigned to another array, because both are constant pointer.
5. The size of the zero length array is 0.
6. `malloc` memory allocation is single dimensional array equivalent and `calloc` memory allocation is double dimensional array equivalent.
7. The major difference between `strcpy` and `strncpy` is `strcpy` copies the whole string including null character but `strncpy` does not copy null character.
8. Double dimensional array is a collection of single dimensional arrays, where each single dimensional array must be used for different purpose.
9. Visiting double dimensional array in row major is always faster than column major.
10. Array name + 1 = Address of next element if it is single dimensional array.
11. Array name + 1 = Address of next array if it is double dimensional array.
12. To get the address of any element of a double array, use

    `Address = base address + (row number* column size + col number) + size of element`

DYNAMIC MEMORY ALLOCATION

1. Dynamic Memory Allocation in C is a part of heap area.
2. Memory allocation in C is possible through variables and functions.
3. Basically, dynamic memory allocation is done by a family of 3 functions in C, such as `malloc( )`, `calloc( )` and `realloc( )`.
4. `malloc`, `calloc` and `realloc` always allocate memory above the break pointer.
5. Break pointer is the initial entry point in heap.
6. `brk` and `sbrk` functions are used to set and retrieve the break pointer.
7. Memory allocated in heap remains in existence for the duration of a program.
8. When memory is allocated from heap their addresses grow upwards and heap size decreases, but when memory is free, the heap size increases.
9. Free means break pointer is decreasing in the form of last break pointer – malloc size – header size.
10. Every program allocates memory only at load time and run time, but how much memory will be allocated in load time and run time.
    * If it is decided at compile time it is called static allocation.
    * If it is decided at run time it is called dynamic memory allocation.
11. Static allocation is a part of Data or BSS area.
12. Automatic allocation: When you declare automatic variables, such as function, actual arguments and local

variables in C program they reserve memory when control is entered in scope and are automatically freed when a control leaves the scope.

13. Automatic memory allocation is a part of stack area.
14. Dynamic memory allocation is not supported by C variables; there is no storage class called "dynamic" to allocate memory dynamically from the process memory in run time.
15. `realloc()` is used to resize the memory block either allocated by `malloc` and `calloc`.
16. `malloc` memory allocation is a single dimensional array equivalent and `calloc` memory allocation is double dimensional array equivalent.
17. Free memory should not be free again.

FUNCTIONS

1. Function returns value, function parameters, local automatic variables of a function stores it in stack area.
2. If function return type is non integer, a prototype is required.
3. Function at a time returns only one value.
4. Copy of the variable is created when it is 'pass by value'.
5. When a function calls to itself directly or indirectly, that is called a function recursion.
6. Function recursion is an alternate of loop.
7. Function recursion always follows LIFO data structure.
8. Library is a special file which hides details about the functionalities of a function.

9. The size of the function is (called Function Frame) manually calculated in the following way:
   - Size of the local auto variables,
   - Size of function actual parameter,
   - Size of the return address,
   - Size of base address of the next function.
10. When a program is linking with static library, all symbols present in static library will be copied to program.
11. A function can be called in two different ways, such as
    - Function name followed by ( ),
    - Function pointer.
12. Function is used to modularize the program.
13. Functions are created in stack memory in a form of doubly linked list, which follow LIFO data structure.
14. Every library function is finally converted into system calls.
15. Library functions are user space implementation but system calls are kernel space implementation.

STORAGE CLASSES

1. Storage class decides the scope, life, storage and default initial value of variables, and functions.
2. Storage Class follows four kinds of scoping rules such as:
   - Block scoping,
   - Function scoping,
   - File scoping,
   - Program scoping.

3. Static storage class is used in three different places, like:
   - File scoping,
   - Sharing the same memory location in different frames of function recursion.
   - To avoid reinitialization of a variables
4. External storage class is used for program scoping.
5. If a variable is defined in one file and that can be accessed in other file in the same program, we can say variable has a program scoping.
6. If a variable is defined in one file and that can't be accessed in other file in the same program, we can say variable has a file scoping.
7. Auto storage class allocates and reallocates memory automatically from stack area.
8. Register variable has no memory address, so pointer cannot work with register variables.
9. Access time of register variable is 1 nanosecond but auto, static and external variable is 200 nanosecond.

PREPROCESSORS

1. When a macro takes a parameter it is called a parameterized macro or macro call.
2. Startup pragma is executed at first before control goes to main and exit pragma is executed at last once control leaves main.
3. The C preprocessor CPP only process source file such as .c and .cpp but does not process object and executable file.

4. The file inclusion preprocessor directive includes file using “ ” or < > , where “ ” searches the file from the current directory path and std. include path but < > searches the file only from the std.include path.

5. # is used as string forming where as ## is used as token pasting operator.

STRUCTURES / UNIONS

1. A Structure always allocates memory in a form of block and block size depends on the longest data member of the structure.

2. If a structure contains a slack byte, that is called an imbalance structure.

3. Word alignment is required to make imbalance structure to balance structure.

4. A structure can not be nested within same structure, because it is a restriction in C.

5. A structure can be nested within the same structure if nested structure is a pointer that is called self-referential structure.

6. Structure is used for data encapsulation, memory link and bit-fields.

7. Two structure variables can not be compared because it is the restriction in C.

8. One structure variable can be assigned to another structure variable if both belong to same structure.

9. Bit field is a unique feature in C which can be applied to a structure for memory optimization.

10. Each data member of a structure begins at different places.

11. Bit field range must be between 1 and size of its data member.
12. Bit field is not allowed to float data members.
13. The very first member of the union is an active data member and it must be the longest type.
14. Only first member of the union can be initialized at the time of definition and rest cannot be initialized.
15. Union provides a technique to share a common memory in different ways.
16. Union size depends on the longest data member of the union.
17. Union is used for memory optimization, creating message formats and share memory or locking mechanism.
18. Each data member of a union begins at same place, but each data member of a structure begins at different places.
19. Byte offset of the union data member is always 0.
20. To create a memory link or linked list we need a structure and that structure must be a self-referential structure.
21. Data member can be accessed using . or -> notation.

FILES AND COMMAND LINES

1. The flow of data from buffer to device or device to buffer is called stream. When data moves from device to buffer it is called an input stream and when it moves from buffer to device it is called output stream.
2. When operating system allocates memory from a kernel space for special use, that memory is called buffer; In Linux buffer size is 4K.

3. The contents of a file is stored in data block of disk.
4. The active file pointer `write_ptr` and `read_ptr` move automatically to next character when any read/write operation is perform.
5. When a stream is not associated in memory its file descriptor value is -1.
6. `getc` returns -1 when it reads `ctrl+z` character.
7. EOF is a macro whose expansion is -1 which is End of File indicator.
8. File handling is possible using `std.library` and system calls.
9. Read/Write operation using `std.library` and system call are possible in buffer instead of file.
10. When a program is loaded by default `stdin`, `stdout`, `stderr` are also loaded.
11. `fwrite` and `fread` only works with low level file.
12. There are three file opening modes, such as read, write and append.
13. Program takes input from different input devices, such as keyboard, scanner, and mouse. Where as command line is used as another input media to a program.
14. In command line argument process use argument counter, argument vector and environment vector.
15. Space is used as a separator from one parameter to another parameter in command line argument.
16. Every parameter in command line argument is of string types.

17. Argument vector and Environment vector are array of string types.

18. `argc`, `argv` and `env` are used generally in command line argument.

MISCELLANEOUS

1. Execution of C program begins at main function but ends at null statement.

2. Width of the Data Bus is known as Word length of the processor.

3. If the programming language uses 16 bits word length then it is called 16 bits programming; if it uses 32 bits word length it is called 32 bits programming.

4. The running of a program in a computer is called process.

5. The process memory of a process is divided into different segments such as code, data, bss, heap and stack segment.

6. How much memory is accessible by the program that depends on the width of the address bus of microprocessor.

7. Page table is used to translate virtual address to physical address in LINUX operating system.

8. Segmentation unit is used to translate logical address to virtual address.

9. To translate a 16 bits offset address and 32 bits far address into 20 bits real address, MSDOS uses segment address *16 + offset address.

10. All automatic variables are created in stack area, so that their address is always growing down the order.

11. All Intel processors are little endian processors, but Motorola, Power pc and Macintosh are big endian processors.

12. Virtual memory is introduced in a system when the memory required is more than the memory available in the system. Operating system implements a data structure called "segmentation technique" by which it assignes virtual memory for each process.

13. If a microprocessor permits access to 16MB RAM, then the width of address bus must be 24 bits.

14. The half of the address `0XFFFFFF` is `0X7FFFF`.

2. "Differentiate Between" Questions with Answers

Q. 1. What is the major difference between loop and recursion?

Answer

It is said that any problem which can be solved by recursion can also be solved by iteration (looping). It is true, but sometimes the iterative way to do something is much more complex to set up and implement.

Looping is simply executing one piece of code over and over. Recursion also loops through the same code but in recursion a subroutine actually calls itself. Neither can be defined as "best" as it just depends on the problem you are trying to solve. Recursion is not used anywhere near as often as looping, especially since it is harder to use properly.

Q. 2. What is the major difference between `for` and `while`?

Answer

The `for` loop is suitable for situations when we know in advance how many times the loop will continue, but the `while` loop is suitable for situations when we do not know how many times the loop will iterate. For example:

Display all the numbers from 1 to 10. Here, we know in advance the loop will continue 10 times, so `for` is suitable.

Count number of bits set to 1 in an integer 375. Here, in advance we do not know what is the binary of 375 and how many bits are set to 1, so `while` loop is suitable here.

Q. 3. What is the major difference between | and || ?

Answer

The bitwise | operator is used to make a particular bit 1 (ON). But the logical || operator is used to check whether any one of the given expressions is true or not. And it always returns either 1 or 0.

Q. 4. What is the major difference between static library and dynamic library?

Answer

When a program is linked with static library, all the symbols present in the static library, whether or not they are used, are copied to the program and all symbols are resolved at the linking time. But when the program is linked with dynamic library, all symbols present in the dynamic library are not copied; only those being used are copied and they are resolved at run time of the program.

Q. 5. What is the major difference between single dimensional array and double dimensional array?

Answer

A single dimensional array is an array of items. A double dimensional array is an array of arrays of items. Double dimensional array is used for matrix and non linear data structure.

Q. 6. What is the major difference between `std.library` and system calls?

Answer

System call can be done from either kernel space or user space (in this case include switch to privilege mode) and executed in kernel space while library can only be called from user program and may do a system call to perform its function like `printf` that needs `write` system call while others do not need kernel at all like sin(), cos()..... . System calls are faster than std.libraries.

System call examples: `write, read, close, mmap, open`

Library examples: `fopen, fclose, printf, scanf, fscanf`

Q. 7. What is the difference between < > and " " ?

Answer

If the file is included using < > such as `<stdio.h>`, the preprocessor searches the include file from the standard include path, but if the file is included using " " such as `"stdio.h"`, the preprocessor searches the include file from standard include path and current working directory

Q. 8. What is the major difference between `startup` pragma and `exit` pragma?

Answer

If any function is included in startup pragma, it is executed first before the control goes inside the main function, but if any function is included in the exit pragma, it is executed once the control leaves the scope of the main function.

Q. 9. What is the major difference between `return` and `exit`?

Answer

`Return` and `exit` both are used to terminate the child process and send a signal to its parent process. But the `return` statement also returns a value from a calling function to its caller function.

Q. 10. What is the major difference between process and threads?

Answer

1. Threads share the address space of the process that created it; processes have their own addressess.
2. Threads have direct access to the data segment of its process; processes have their own copy of the data segment of the parent process.
3. Threads can directly communicate with other threads of its process; processes must use interprocess communication to communicate with sibling processes.
4. Threads have almost no overhead; processes have considerable overhead.
5. New threads can be easily created; new processes require duplication of the parent process.

Q. 11. What is the major difference between share locking and exclusive locking?

Answer

Exclusive lock is required when a process is writing, but shared lock is required when the process is reading. If the process is only reading then no lock is required.

Q. 12. What is the major difference between `open` and `fopen`?

Answer

`Open` and `fopen` both are used to open a file. However, `open` function is a system call and `fopen` is a library function.

3. System calls to perform its function but std.libraries are on user space implementation.
4. Return type of `fopen` is `FILE*`, but `open` return type is an integer.

Q. 13. What is the major difference between `putc` and `fputc`?

Answer

The major difference between `putc` and `fputc` is, `putc` is macro call implementation and `fputc` is `std.library` function implementation. But job of both is to write a byte at a time into stream buffer.

Q. 14. What is the major difference between `printf` and `sprintf`?

Answer

The only difference between `sprintf()` and `printf()` is that `sprintf()` writes data into a character array, while `printf()` writes data to standard output device. For example

`printf("my age is %d",25)` //output in monitor my age is 25

`sprintf(x,"my age is %d",25);` //output is written into x;

Q. 15. What is the major difference between precedence and associativity?

Answer

Precedence decides which operation will be performed first in an expression, but associativity decides the order of evaluation, when more than one operators enjoy the same precedence in an expression.

Q. 16. What is the major difference between reference and dereference in pointer?

Answer

The term reference means the address of any memory location. Pointer is the only variable which refers to that location. If `read` and `write` operation is done using the pointer where the pointer refers that is called dereference.

Q. 17. What is the major difference between writing prototype inside and outside of a function?

Answer

While writing a prototype inside a function only that function will be able to call but other functions cannot call. But if we write prototype above the function all functions can call.

Q. 18. What is the difference between `if else` and `switch case`?

Answer

Simply saying, `if else` can have values based on constraints where as `switch case` can have values based on user choice. In `if else`, first the condition is verified, then it comes to else, whereas in the `switch case` first it checks the cases and then it switches to that particular case.

Q. 19. What is the major difference between `sizeof` and `strlen`?

Answer

`sizeof` statement is used to measure the size of any variable or any data types, but `strlen` measures only the size of a string.

`sizeof` command measures the size of string including null character, i.e., '\0', but `strlen` measures the length of string excluding the null character.

Q. 20. What is the major difference between `break` and `continue`?

Answer

`break` statement is used to transfer the control to outside the loop and `switch…case`, but `continue` statement is used to transfer the control to beginning of the loop.

Q. 21. What is the major difference between `macro` and `typedef`?

Answer

# defined directive creates `macro` which is handled by a preprocessor (a program run before the actual C compiler) which works like 'replace all' in your Editor. `Typedef` is handled by the C compiler itself, and is an actual definition of a new name of the type.

Q. 22. What is the major difference between `lvalue` and `rvalue`?

Answer

An expression that can appear only on the right-hand side of an expression is an `rvalue`.

The following examples are instances of `rvalues`:

- Array name
- Function name
- All symbolic constants
- Constants and constant variables

Other than the above, constraints like variable, pointer and structure are lvalue objects which can appear in both left and right side of an assignment operator.

Q. 23. What is the major difference between = and == ?

Answer

The '=' is an assignment operator which is evaluated from right to left and its left side must always be a variable. But the operator '==' is a logical comparison operator which checks the equality of two expressions.

Q. 24. What is the major difference between `scanf` and `sscanf`?

Answer

The `scanf` function reads data from standard input stream `stdin` into the locations given by each entry in the argument list. The argument list, if it exists, follows the format string.

The `sscanf` function reads data from buffer into the locations given by argument list. Reaching the end of the string pointed to by buffer is equivalent to the `fscanf` function reaching the end-of-file (EOF). If the strings pointed to by buffer and format overlap, the behavior is not defined.

Q. 25. What is the major difference between process and program?

Answer

Process is program under execution. Process is an active entity while a program is in the form of programming language and also it is a passive entity. Moreover, a program is stored in secondary storage but process is stored in main memory of the system.

Q. 26. What is the major difference between `memcpy` and `memmove`?

Answer

`Memcpy()` copies the bytes of data between memory blocks. If the block of memory overlaps, the function might not work properly. Use `memmove()` to deal with overlapping memory blocks. `Memmove()` is very much like `memcpy()` but very flexible as it handles overlapping of memory blocks.

Example:

```
str = "0123456789"
memmove(str+2, str+1, 3);
```

returns: 0112356789

```
memcpy(str+2, str+1, 3);
```

returns: 0111356789

Q. 27. What is the major difference between text file and binary file?

Answer

A text file is human readable.

A binary file is not human readable

Text file is an ASCII file which is human readable, But binary file contains the text data in encoded format which a human cannot read. In a text file each character is a 7 data bits signed character, but in binary file each character is 8 data bits or unsigned character.

Q. 28. What is the major difference between `NUL` and `NULL`?

Answer

`NULL` is a macro defined in `"stddef.h"` for the null pointer, such as `((void*)0)`.

NUL is the name of the first character in the ASCII character set. It corresponds to a zero value.

Q. 29. What is the major difference between `while` and `do while`?

Answer

In while loop expression is tested first and then the body is executed. If the expression is evaluated to be true it is executed, otherwise it is not.

In do while loop, body is executed first and then the expression is tested. If test expression is evaluated to be true then the body of loop is again executed.

Thus it is clear that the body of do while loop gets executed at least once, where as, it is not necessary in while loop

While loop is used for unknown times, but do while loop is used at least once.

Q. 30. What is the major difference between array and string?

Answer

When we declare an array of characters it has to be terminated by the NULL, but termination by NULL in case of string is automatic.

Q. 31. What is the major difference between & and &&?

Answer

The bitwise '&' operator is used to check a particular bit is 0 or 1 (ON/OFF) of an integral value. But the logical '&&' operator is used to check which of the two expression is true or false and its return value is either 0 or 1.

Q. 32. What is the major difference between '.' and '->'?

Answer

'.' and '->' both are used as data member access operator of a structure or union. The notation '.' is used when structure and union variable is of value type and '->' notation is used when variable is a pointer type.

Q. 33. What is the major difference between `i++` and `++i` ?

Answer

++ operator is a unary operator which is used as post and pre increment in an expression. In the expression if ++ is used as a post, then expression uses the variable first before its increments. When it is used as pre, the expression first increments the variable before using it.

Q. 34. What is the major difference between signed integer and unsigned integer?

Answer

Both, signed and unsigned integers, are modifiers in C. Unsigned modifiers can hold a larger positive value (almost double of the signed moodifier), and no negative value. Signed integers can hold both positive and negative numbers, but unsigned can operate only in positive data.

Q. 35. What is the major difference between `getch` and `getche`?

Answer

Both `getch()` and `getche()` are used to read single character and there is very little difference.

`getch()` does not display output to screen if used without lvalue.

`getche()` displays output to screen even if used without lvalue.

Q. 36. What is the major difference between `void main` and `int main`?

Answer

Every function has a default return type as `int`. `void main()` ensures that the main function does not return anything.

Q. 37. What is the major difference between external variable and global variable?

Answer

A global variable in C/C++ is a variable which can be accessed from any module in your program. You can define a global variable with a statement like this:

```
int  myGlobalVariable;
```

This allocates storage for the data, and tells the compiler that you want to access that storage with the name `'myGlobalVariable'`.

But what do you do if you want to access that variable from another module in the program? You cannot use the same statement given above, because then you'll have 2 variables named `'myGlobalVariable'`, which is not allowed. So, the solution is to let your other modules DECLARE the variable without DEFINING it using extern:

```
extern  int  myGlobalVariable;
```

This tells the compiler "there's a variable defined in another module called myGlobalVariable, of type integer. I want you to accept my attempts to access it, but do not allocate storage for it because another module has already done that".

Q. 38. What is the major difference between array and linked list?

Answer

The difference between arrays and linked lists are:

1. Arrays are linear data structures. Linked lists are linear and non-linear data structures.
2. Linked lists are linear for accessing, and non-linear for storing in memory.
3. Array has homogenous values and each element is independent of each others positions. Each node in the linked list is connected with its previous node which is a pointer to the node.
4. Array elements can be modified easily by identifying the index value. It is a complex process for modifying the node in a linked list.
5. Array elements can not be added, deleted once it is declared. The nodes in the linked list can be added and deleted from the list.
6. Both array and linked list are used to prepare a list. If the list size is known in advance, use an array, but if the list size is unknown in advance, then we use linked list. Thus, array creates static list and linked list creates dynamic list.

Q. 39. What is the major difference between const and static?

Answer

A constant is a variable whose value cannot be changed. A static is a variable that cannot be used outside the scope of its declaration. That is, if it is a global variable then it can be used only in the file that declares it. If it is a variable inside a function, then it can be used only inside that function.

Q. 40. What is the major difference between `strcpy` and `strncpy` ?

Answer

`strcpy` copies the whole string including null character '\0', but `strncpy` copies only fixed number of character, and does not copy the null character.

Q. 41. What is the major difference between `stdout` and `stderr` ?

Answer

`stdout` and `stderr` both are output streams and associated with monitor. But `stdout` data can be redirected where as `stderr` data cannot be redirected.

Q. 42. What is the major difference between function call and macro calls?

Answer

Macro and functions are entirely different. A Macro in principle is text processing like 'Find and Replace' in a text editor. It only happens automatically before compilation process. But functions are handled in all states of the compilation process, right up to linking.

Q. 43. What is the difference between writing static to a local variable and global variable?

Answer

Static storage class is used in two different places, which are:

(a) To avoid reinitialization of local variable during recursive or iterative calls of function,

(b) As a file scoping, which means that if write is static to a global variable it cannot be accessed in other files of the program.

Q. 44. What is the difference between static and external variable?

Answer

Storage classes are used to indicate duration and scope of a variable or identifier. Duration indicates the life span of a variable. Scope indicates the visibility of the variable.

The static storage class is used to declare an identifier that is a local variable either to a function or a file but extern storage exists and retains its value after control passes from where it was declared. This storage class has a duration that is permanent.

Q. 45. What is the difference between declaring a variable and defining a variable?

Answer

Declaring a variable means describing its type to the compiler but not allocating any space for it. Defining a variable means declaring it and also allocating space to hold the variable

Example

```
int  num;
```

this statement is declaration and also definition since it also allocates memory for variable `num`. But when you use the following statement `extern int num;`

it is only a declaration since you are not allocating memory here.

Q. 46. What is the major difference between `malloc` and `calloc`?

Answer

Both the `malloc()` and the `calloc()` functions are used to allocate dynamic memory. Malloc memory allocation is single dimensional array equivalent but calloc memory allocation is double dimensional array equivalent.

Q. 47. What is the difference between "call by value" and "call by address" in a function?

Answer

Call by value and call by address both are used for function calling convention. But the major difference between call by value and call by address is,

1. *Call by value:* If any modification is done in calling function it does not affect the caller function,
2. *Call by address:* If any modification is done in calling function it affects the caller function.

Q. 48. What is the difference between a structure and a union?

Answer

Structure:

1. Each data members begin at different location.

2. Size of structure is size of all data members.

Union:

1. Each data members begin at same location.
2. Size of union is longest data member size.

Both structure and union are completely different.

Q. 49. What is the difference between "w" and "w+" modes in C?

Answer

"w" mode opens file for write only purpose, such as `O_CREAT|O_TRUNC|O_WRONLY` mode, but "w+" opens the file both in read and write purpose, such as `O_CREAT|O_TRUNC|O_RDWR` modes.

Q. 50. What is the difference between `typedef` and `#define` ?

Answer

`typedef` obeys the scope rules of the compiler but `#define` does not obey the scope rules.

3. "Secret" Problems with Solution and Explanation

P. 1. Call a function if function name is passed as a string.

Solution

```
#include "dlfcn.h"
main()
{
    int (*p)();
    char x[]="printf";
    int k=dlopen("/lib/libc.so.6",
    RTLD_LAZY);
    p=dlsym(k,x);
    (*p)("Good Morning\n");
    dlclose(k);
}
```

Explanation

A function name is also known as a symbol. All symbols are generally present in std.library. The standard library function `dlsym` is used to search a symbol from a library and return its address if present. Using function pointer that function can be called if function name is passed as a string to `dlsym` function.

P. 2. Execute two functions such as sum and product according to user's requirement without using any control structures.

Solution

```
int sum(int ,int );
int product(int,int);
main()
{
    int a,b,c;
    int (*p[])()={sum, product};
    int i;
    printf("Press 0-SUM,1-PRODUCT :");
    scanf("%d",&i);
    printf("Enter any two numbers :");
    scanf("%d%d",&a,&b);

    c=(*p[i])(a,b);
    printf("%d",c);
}
    int sum(int a,int  b)
{
    return a+b;
}
    int product(int a,int b)
{
    return a*b;
}
```

Explanation

A function can be called in two different ways, such as ‘function name followed by parenthesis ‘( )’ and function pointer’. Here in the above program any one function is called using function pointer, looking at user’s choice without implementing any control structure.

P. 3. Add any two numbers without using any arithmetic operators.

Solution

```
main()
{
    char *a=10,b=20;
    printf("%d",&a[b]);
}
```

Explanation

A single-dimensional array can be represented in different ways, such as `a[2]` can be defined as `2[a]`, `*(a+2)` and `*(2+a)`. See the expression `a[2]` as `*(a+2)`. Here, `a+2` means `a` is pointer type and 2 is value type and in between + is an operator which adds `a` and 2. So, to overcome our problem of adding between any two numbers using any arithmetic operators we refer array expression such as `&a[b]` means `&*(a+b)`. Here, `a` and `b` both are added to each other and finally printf `prints` the values as addition.

P. 4. Reverse a string without using array and any string reverse function.

Solution

```
main()
{
    rev_str();
}
int rev_str()
{
    char ch;
    if((ch=getchar())!=10)
    {
        rev_str();
    }
    else
        return;

    putchar(ch);
}
```

Explanation

A function always allocates memory from stack in a LIFO fashion. So the above program is written in function recursion and every time `rev_str` function is pushed into stack the `getchar()` reads a character when the function is pushed; and `putchar()` prints a character when a function is popped, thus the output is in reverse order.

P. 5. Count the number of bits set to 1 in a short integer without using any condition.

Solution

```
main()
{
    int c=0;
    int n;
    printf("Enter any number :");
    scanf("%d",&n);
    while(n>0)
    {
        c++;
        n=n&n-1;
    }
    printf("%d ",c);
}
```

Explanation

The bitwise '&' operator is used to check whether a particular bit is 1 or 0. When the number is ended with that number end 1 is subtracted from it then, the resulting number is number -1 or 0. So based on the same conditions here in the above program it counts how much time `n` and `n-1` is anded and returns the desired result.

P. 6. Convert the following program into function recursion.

Example 1

```
main()
{
    int i,j;
    for(i=0;i<=5;i++)
    {
        for(j=i+1;j<=5;j++)
        {
            printf("*");
        }
        printf("\n");
    }
}
```

Example 2

```
main()
```

```
{
    rec(5);
}
    int rec(int n)
{
    static int i=0,j=0;
    if(i<n)
    {
        int k=n-(j+i);
        j++;
        if(k>0)
        {
            printf("* ");

        }
        else
        {
            printf("\n");
            i++;
            j=0;
        }
        rec(n);
    }
    else
        return n;
}
```

Any problem that can be solved using function recursion can also be solved using loop. So, function recursion is an alternate of loop statement. Here, in Example 1 the loop is nested within a loop which is replaced in Example 2, using function recursion.

P. 7. Add any two numbers using function recursion.

Solution

```
main()
{
    int x;
    x=add(5,6);
    printf("%d",x);
}
int add(int a,int b)
{
    static int i=0;
    if(a>0)
    {
        b++;
        a- -;
        add(a,b);
    }
    else
    {
        i=b;
        return;
    }
    return i;
}
```

Explanation

Any problem solved in loop can also be solved using function recursion. So if we are to add any two numbers using recursion means it can be solved using loop. So here `a` is subtracted by 1 and `b` is incremented by 1 as long as a value is greater than zero.

P. 8. If the base address of a double dimensional array is given, how can we calculate the address of any other element?

Solution

```
    int x[10][15];
```

`if &x[0][0] is 500,` then what will be the address of `&x[3][7];`

```
    New address = base address + (row number *
col size + col number) * size of array element.
```

Therefore,

```
    New address = 500 + (3 * 15 + 7) * 4.
```

P. 9. How to measure the size of a variable or data types without using `size of` command?

Solution

```
#define SIZE_OF(X) ((char *)(&x+1) - \
  (char\*)&x)
main()
{
    short int x;
    printf("%d",SIZE_OF(x));
}
```

Explanation

Let us see how the answer is coming without `sizeof()`. First declare a `short int x` variable. By using `&x` we get the base address of the variable `x` and by adding 1 to it we get the base address of next short int type. Hence the resulting address of `(&x+1)` will be 2 bytes more than the base address of the variable `x`. But if we just display the difference between the base address of `x` and the incremented address, then the difference will be 1 means "1 block of `short int` type has been added" but we need the result of size of variable in terms of bytes not in terms of blocks. This can be achieved if we typecast the address into `char*`, because address of `char` datatype will always be in blocks of 1 byte, hence if the difference between the base address of `x` and the incremented address is displayed in terms of `char` type, then the difference will be displayed as 2, because the difference is actually 2 blocks or 2 bytes in terms of `char` type representation.

P. 10. What are the different algorithms that we can use to swap two variables?

Solution

Logic 1:

```
int a=5,b=10;
int c;      //using extra memory space

    c=a;
    a=b;
    b=c;
```

The above example shows the swapping of two variables `a` and `b` by using extra memory space for variable `c`. In the first step the value of `a` is assigned to `c`, thus the value of `c` now becomes 5, after that `b` is assigned to `a` and the value of a now becomes 10 and finally value of `c` is assigned to `b`. Thus the values of `a` and `b` have been swapped.

Logic 2:

```
    a=a+b; // Using arithmetic operators
    b=a-b;
    a=a-b;
```

In the above code snippet an interchange in the values of `a` and `b` is done but without using any extra memory space. In the first statement `a` is assigned the value of (a+b), i.e., 15. In the 2nd statement `b` is assigned the value of `(a-b)`, i.e., (15-10) equal to 5 and in the 3rd statement `a` is replaced with `(a-b)`, i.e. 10.

Logic 3:

```
a=a^b;  // Using Bit wise operators
b=b^a;
a=a^b;
```

In line 1, `a=a^b` results in the value of `a` becoming 15 and `b` becoming 10. In the 2nd line `b=b^a`, means that `a` has the same 15 but the value of `b` is now 5. In the subsequent line 3, the expression `a=a^b` results in varible `a` becoming 10 and `b` becoming 5, thereby completing the swap.

line	operation	value of `a`	value of `b`
1	`a=a^b`	15	10
2	`b=b^a`	15	5
3	`a=a^b`	10	5

Logic 4:

```
asm mov ax,a;
asm mov bx,b;
     asm mov cx,bx;
     asm mov bx,ax;
     asm mov ax,cx;
     asm mov a,ax;
     asm mov b,bx;
```

The above example shows the swapping of two variables using assembly programming and without using assignment operator.

Logic 5:

```
    a^=b^=a^=b; //Using one line statement
```

In the above example, `a` and `b` are interchanged in a single line statement using the compound assignment operator, where its order of evaluation is from right to left. So, the value b is XORed with `a` and the result is assigned to `a` (i.e., `a=15`). In the second compound assignment operator `5` is XORed with `b` and the result is assigned to `b` (i.e., `b=5`). Finally, `b` is again XORed with `a` and the result is assigned to `a` and `a` becomes 10. So, both are swapped.

Logic 6:

```
   a=(a+b)-(b=a); // Using one line statement
```

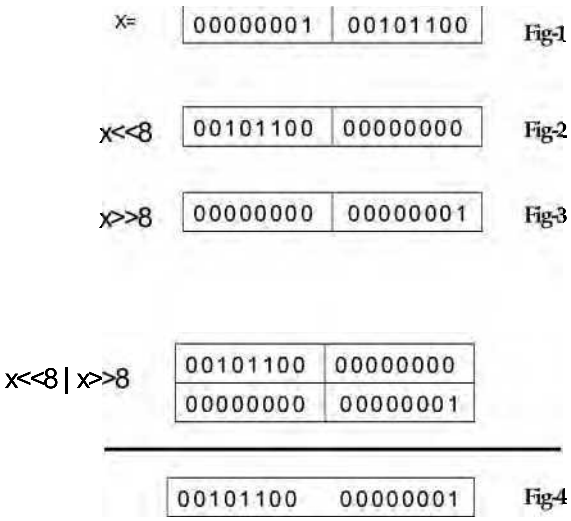
In the above example, the operator in parenthesis enjoys the highest priority and the order of evaluation is from left to right. Hence `a+b` is evaluated first and replaced with 15. Then `(b=a)` is evaluated and the value of `a` is assigned to `b`, which is 5. Finally, `a` is replaced with 15–5 , i.e., 10.

P. 11. How to change the byte order of an integer?

Solution

```
main()
{
    short int x=10;
    x=x<<8|x>>8;
    printf("%hd",x);
}

Output: 11265
```



In Figure 1, we have the binary representation of x, depicting the value 300. In Figure 2, the value of the 1st byte of x is shifted to the 2nd byte. In Figure 3, the value of the 2nd byte of x is shifted to the 1st byte. Figure 4 is the result of byte ordering of x that is `x<<8| x>>8`. So, in the above program before byte order the value of x is 300 and after byte ordering the value of x becomes 11265.

P. 12. How do we write a general swap macro, which can swap any type of data?

Solution

```
#define SWAP(a,b,type) {type t = \ a;a=b;b=t;}
main()
```

```
{
    float f1 = 10.3, f2 = 12.3;
    SWAP(f1,f2,float);
    printf("f1=%.2f\tf2=%.2f",f1,f2);
}
```

Explanation

In the above program SWAP is a generic macro which swaps between two variables of any data types. The macro SWAP takes three parameters where the first two parameters are variable which are to be swapped and the third parameter is a type of variable. The following expanded source code is produced by preprocessor before compilation for the above program.

P. 13. How to check the Endianness of the system?

Solution

Logic 1

```
main()
{
    int x = 300;
    if(((*(unsigned char *)&x)==1) &&
    ((*(unsigned char *)&x+1)==44))
        printf("BIG ENDIAN\n");
    else
        printf("LITTLE ENDIAN");
}
```

Logic 2

```
void main()
{
    union xxx
    {
        unsigned int x;
        unsigned char ch[2];
    };
    union xxx check={300};

if((check.ch[0]= =1)&&(check.ch[1] = = 44))
        printf("BIG ENDIAN");
    else
        printf("LITTLE ENDIAN");
}
```

Explanation

Addresses are always created in memory in ABCD format, but the data can be stored in memory either in DCBA format, which is Little-Endean system or in the ABCD format in case of a Big-Endean system.

Let's store 300 in the memory location of an integer. Now if we visit the memory location byte by byte we will find that the content of first byte will be 1 and the content of the other will be 44. In a Big-Endean System the 1st byte will contain 1 and the 2nd byte will be 44 according to the ABCD data storage format. Whereas in the Little-Endean System the 1st will contain 44 and the 2nd byte will be 1 according to the DCBA data storage format. Please refer to Figures 1 & 2.

Logic 1

First we have to extract the base address of x by typecasting it, i.e., unsigned char *. Then by using de-referential operator *, we can get the value of the low or 1st byte. If that is found to be equal to 1 and the value of the high / 2nd byte obtained in the same method by adding 1 to the base address of x is equal to 44, then the system is `Big-Endian` or else it is `Little-Endian`.

Logic 2

In this logic we have taken the union `xxx` with 2 members – one is an integer `x` and the other is an array of 2 unsigned character, that is “ch[2]”. As in a union all data members share the same memory location, hence in this case, “ch[0]” will be accessing the 1st or low byte of “x” and “ch[1]” will be accessing the 2nd or high byte of “x”. So we can compare p.ch[0] with 1 and p.ch[1] with 44. If it matches it will be “Big-Endean”, else it will be “Little-Endean”.

P. 14. How to find maximum number of elements present in any array?

Solution

```
#define MAX 50
void main()
{
    float arr[MAX]={4.0,8.9,9.6,4.2};
    printf("THE MAXMIMUM NO OF ELEMENT THAT
CAN BE STORED = %d",sizeof(arr)/sizeof(arr[0]));
}
```

Explanation

The size of an array is nothing but the product of the total number of elements that it can store, with the size of the data type it belongs to. Keeping this in mind we just reverse the process by dividing the size of the total array with size of the data type which gives us the total number of elements.

The `sizeof(arr)` gives us the total size of the array and simultaneously `sizeof(arr[0])` gives the size of the 1st element. So size of the total array, when divided with size of the first element will give us the total number of elements that we can store in that particular array.

P. 15. Given are two arrays A and B. Array 'A' contains all the elements of array 'B' along with an extra element. Write the program to find the extra element.

Solution

```
int find_extra(int [],int [],int);
main()
{
    int first[]={1,2,3,67,90};
    int second[]={2,3,67,90};
    int n2=sizeof(second)/sizeof
    (second[0]);
    int extra = find_extra(first,
    second,n2);
    printf("THE EXTRA ELEMENT PRESENT IS
    = %d",extra);
}
    int find_extra(int first[],int
    second[],int n2)
{
    int loop,sum1=0,sum2=0;
    for(loop=0;loop<no2;loop++)
    {
        sum1 +=first[loop];
        sum2 +=second[loop];
    }
    return (sum1+first[loop]-sum2);
}
```

Explanation

We can get the extra element by subtracting the summation of the elements of first array from the summation of the elements of second array.

Here `find_extra` is a function receiving 3 parameters, two of them are arrays and the 3rd one is the number of elements present in the 2nd array. Here `n2` represents the number of elements present in 2nd (second) array. This means the 1st array contains `n2+1` number of elements. In order to find the extra element present in the 1st array, we compute the summation of the elements of the 2nd array and subtract it from the summation of the elements of the 1st array.

As the loop will continue for `n2` times, `sum2` will contain summation of `n2` number of elements present in the **second** array where as `sum1` also contains summation of `n2` number of elements but not of the all elements as it has `n2+1` number of elements. Now in the **return** statement before subtracting `sum2` from `sum1`, we add first[i] (i.e the last element present in **first** array) to make it the summation of all the elements.

The same logic is implemented in the next solution but this time its coded into the main function.

P. 16. Write a program to display "Hello" without any semicolon.

Solution

```
main()
{
    while(!printf("Hello"))
    {
    }
}
```

Explanation

A semicolon is used to terminate a C statement. But the control structure `if`, while or `switch` do not require a semicolon to terminate. So if we use the `printf` statement inside these constructs, then we don't require any semicolon to terminate the `printf` statement. The `printf` function returns non-zero when it prints anything on screen. So here in the above program `printf` function returns 5 which is nonzero and `!printf` statement return 0, which makes the while loop false. So it prints only once `Hello` on screen.

P. 17. Given is an array of numbers. Except one number all the numbers occur twice. Write a program to find which numbers occur only once.

Solution

```
int find_unique(int [], int);
main()
{
    int arr[]={1,1,2,3,4};
    int duplicate=find_unique(arr,
    sizeof(arr)/sizeof(arr[0]));
    printf("The Unique Element is =
    %d",duplicate);
}
    int find_unique(int arr[],int n)
{
    int loop,sum;
    for(loop=0;loop<n;loop++)
        sum +=arr[loop];
    return (sum - n * (n-1)/2);
}
```

Explanation

There are 2 basic pieces of logic being applied in the solution, which are –

1. The array must have odd number of elements, because the numbers are repeated twice except for 1 number.
2. If we sort the elements in the array, then all the similar pairs will come together and the unmatched number will be occurring singularly.

Based on the above logic we have a function `find_unique`, which takes the array of integers as parameter, in which every element, except one, it occurs twice. Hence the array must contain odd number of elements, that's why our function doesn't bother to check any boundary condition.

To find the unique number, we first sort the list so that all the repeated numbers will appear in two conjugative places. Then we search for a place starting from the beginning (i.e., 0th index) by checking two conjugative places at a time until a mismatch occurs. If there is no mismatch, then the last element must be the unique one.

P. 18. Write a one-line statement to check whether the number is power of 2 or not.

Solution

```
main()
{
    int n;
    printf("\nEnter Any number:");
    scanf("%d",&n);
    if(n && (n & n-1)==0)
        printf("\nIt is Power of 2");
    else
```

```
        printf("\nIt is Not power of 2");
}
```

Explanation

The logic says that if a number is a power of 2 then in the binary representation only one bit of the number is 1 and rest are 0. For example, 32 has a binary representation of 00100000. Now the expression `n&n-1` will reset only one set of bit (1) from the number to 0. After this operation if the number is the power of 2, the set bit(1) will reset to 0. The resulting value will be 0 and the expression `n && 0` gives 0. Hence, the condition evaluates as true. In this example, the `if` condition looks like `(32 && (32 & 31))`. This will evaluate as 0 and the condition will be true. For odd numbers the bitwise & operation will not result in 0 and hence the condition will be false.

P. 19. Write a C program to find the smallest of three integers without using any comparison operators.

Solution

```
main()
{
    int a,b,c;
    printf("Enter three Number:");
    scanf("%d%d%d",&a,&b,&c);
    if((a-b) & 32768)
    {
        if((a-c) & 32768)
            printf("The Smallest Number
                is %d",a);
        else
            printf("The Smallest Number
            is %d",c);
    }
    else
    {
        if((b-c) & 32768)
            printf("The Smallest Number
                is %d",b);
        else
            printf("The Smallest Number
                is %d",c);
    }

}
```

Explanation

When we subtract a number from another number we can find out which one is greater than other. If the result is positive then the first number is greater else the second one is greater. This is known by doing a bitwise & operation with the sign bit (i.e., 32768 or 15th bit).

P. 20. Write a program to multiply any two numbers without using te multiplication '*' operator.

Solution

```
long int mul(int,int);
main()
{
    int a,b;
    long int result;
    printf("Enter Two Number  :");
    scanf("%d%d",&a,&b);
    result = mul(a,b);
    printf("The Multiplication Of %d *
        %d = %d",a,b,result);
}
long int mul(int a,int b)
{
    if(b==0)
        return;
    return (b>0 ? (a + mul(a,b-1)):
      (-a+mul(a,b+1)));
}
/*long int mul(int a,int b)
{
    long int res = 0;
    if(b<0)
    {
        b = -b;
        a = -a;
    }
    while(b)
    {
        res +=a;
        b—;
    }
    return res;
}*/
```

Explanation

It is evident that the multiplication of two numbers is nothing but addition of one number with itself second number of times. For example, 3 × 4 is nothing but addition of 3 with itself 4 times. The same process is represented in two different ways first one is done through iteration and the second one is done through recursive function.

P. 21. Write a program to compare between any two integers without using '= =' and '!=' operator.

Solution

```
main()
{
    unsigned int a,b,x;
    printf("Enter two No.");
    scanf("%u%u",&a,&b);
    if(a<b||a>b)
        printf("Not Equal");
    else
        printf("Equal");
}
```

Explanation

If a number is equal to another then both values must be same. So, here we have checked for the same in a reverse manner. Its evident that a number will be equal to another when that number is neither less than nor greater than the other. If any one condition in the given `if` condition `if (a<b||a>b)` fails then they are not equal.

P. 22. Find the maximum and minimum of any two numbers without using any condition or loop.

Solution

```
main()
{
    int a=10,b=25;
    printf("Maximum %d Minimum %d",
        ((a+b)+abs(a-b))/2,((a+b)-abs(a-
         b))/2);
}
```

Explanation

The logic behind the solution is that if we add the numbers and then add the result to the absolute difference of the same 2 numbers and then we divide the resultant value by 2 we will get the greater or the maximum of 2 numbers. Here we have done the same by dividing the result of the addition of the sum of two numbers and their absolute difference by 2. Similarly minimum can be found by doing a subtraction between the summation of the 2 numbers and their absolute difference and dividing the result by 2.

The results of abs(a-b) and abs(b-a) are same (here that is 5).

a	=	10	
b	=	25	
a+b	=	35	
abs(a-b)	=	15	
(a+b) + abs(a–b))/2	=	25	(Max value)
(a+b) – abs(a–b))/2	=	10	(Min value)

P. 23. Display all the numbers from 1 to 100 without any relational operators.

Solution

```
main()
{
    int i=0;
    while(100 - i++)
        printf("%d\n",i);
}
```

Explanation

The most important thing to understand here is the `while` condition expression. The loop will terminate when the value inside the `while` condition will become 0. In order to achieve that we have initialized the value of a variable `i` to zero. Then we go on incrementing the value of `i` and printing its value until it becomes 100. This condition is checked by subtracting `i` from 100 and when the value is zero then the loop terminates. The subtraction value will become 0 when the value of `i` will be 100, by which time the program would have displayed 1 – 100.

P. 24. Write a program to print "Hello world" repeatedly without using loop, recursion, and any control structure.

Solution

```
#include "stdlib.h"
main()
{
    printf("Hello World\n");
    system("test");
}
```

Explanation

To repeat some part of a C program, we need repetition control structure, such as `for`, `while` and `do while` statement. In advance we know that any problem can be solved in loop, if it can be solved in function recursion, but it does not happen. Because it is restricted in the above question. So to overcome this problem write the above code in a file, such as `test.c`. Then compile and link the code to create `test` as executable file using the command `gcc -otest test.c`. Then, pass the executable file name in the `system` function, such as system(`test`) which will execute the same program repeatedly, because system is a function which executes extern commands or executable files.

P. 25. Write a program to count the number of vowels and consonants in a given string without using loop and if statement.

Solution

```
int v=0,c=0, space =0;
int calc(char *strPtr)
{
    (*strPtr = = ‘a’ ||*strPtr = = ‘e’
    ||*strPtr = = ‘i’ || *strPtr = = ‘o’
    || *strPtr = = ‘u’)?v++ : (*strPtr
    = = ‘ ‘)? space++ : c++;
    (*(strPtr+1)!=0) ? calc(strPtr+1) : 0;
    return *(strPtr+1)==0 ? 0 : 1;
}
main()
{
    char str[20];
    printf(“Enter a string”);
    gets(str);
    calc(str);
    printf(“Vowels = %d consonant =
    %d”,v,c);
}
```

Explanation

In order to count the number of vowels and consonants in a string we have used a recursive function called `calc`. The function takes one string argument and updates two global variables v & c for vowels and consonants respectively. Within the function the character present in the base address, that is supplied as an argument to calc, is checked for the five vowels & if it matches we increment the value of v otherwise increment the value of c as it will be a consonant. We call the same function by incrementing the base address of the string, this continues till it gets a null character. Once the function passes the entire string, the variables would have been checked for vowels or constants and the respective numbers of both would be displayed.

Q. 26. Write a program that will display spiral of n*n numbers, using constant space without using array.

Solution

```
void spiral_print(int, int);
main()
{
    int x,y;
    printf("Enter The Dimension");
    scanf("%d",&x);
    for(y=x;y>0;y—)
    {
        spiral_print(x,y);
        printf("\n");
    }
}
void spiral_print(int n, int i)
{
    int k;
    if(i<=0)
        return;
    else if(n==i)
```

```
        for(k=1;k<=n;k++)
            printf("%3d",n * n-k);
    else if(i==1)
    {
        for(k=0;k<n;k++)
            printf("%3d",(n*n-1)-3*
              (n-1)+k);
    }
    else
    {
        printf("%3d",(n*n)-3 * (n-1)-i);
        spiral_print(n-2,i-1);
        printf("%3d",(n *(n-1))-(n-i));
    }
    i– –;
}
```

Explanation

To print the numbers in spiral order starting from 0 to `n*n-1` in an anti-clockwise direction we have used the help of the spiral function. For this we call this function spiral `n` times starting the value from `n` and go on decreasing to 1 which takes two arguments, dimension `n` and parts `i` and `n=i`. Each time, we print the required numbers in a line by using the recursive call. In each recursive call we check three conditions,

1. Condition for first line (i.e., if dimension `n` is equal to `i`)

   Here, the function prints the numbers in decreasing order starting from `n*n-1` to `n*(n-1)`.

2. Condition for last line (i.e., if part `i` is equal to 1).

   Here the function prints the numbers in increasing order from `(n*n-1)-3*(n-1) to (n*n-1)-2*(n-1)`.

3. Condition for middle lines

   Here, the function prints the first number to its dimension and then again calls the function from its lower level (i.e., `n-2 & i-1`) and then prints the last number to its dimension.

P. 27. Convert a floating point value to a string and format its textual representation.

Solution

```
#include "stdio.h"
#define DIGITS 10000;
void float_string(float,char []);
main()
{
    float x;
    char str[15];
    printf("Enter The floating Point
    Number :");
    scanf("%f",&x);
    float_string(x,str);
    puts(str);
}
void float_string(float x, char str[])
{
    int i=0,j;
    long integral, fractional, k=1;
    if(x<0)
    {
        str[i++] = '-';
        x = -x;
    }
    integral = (long)x;
    fractional = (x-integral+1)*DIGITS;
    if(!integral)
        str[i++] = ‘0’;
    while(integral)
    {
        k = k * 10 + integral % 10;
        integral = integral/10;
    }
    integral = k;
    k = 0;
    while(fractional>1)
    {
        k = k * 10 + fractional % 10;
        fractional = fractional / 10;
    }
    fractional = k;
    while(integral>1)
    {
        str[i++] = integral % 10 + 48;
        integral = integral/10;
    }
    str[i++] = ‘.’;
    if(!fractional)
        str[i++] = ‘0’;
    while(fractional)
    {
        str[i++] = fractional % 10 + 48;
        fractional = fractional / 10;
    }
    str[i] = 0;
}
```

Explanation

In the above program the function float string takes two parameters, one is the float and another is a character array. It converts the float into a string and stores it in the character array.

The floating type value is divided into two parts

(i) Integral value

(ii) Fractional value

Each part is converted into string and stored in the character array one after another separated by "`.`". At first we are checking whether the value is negative or not. If the value is negative, then we convert it into positive and put a '`-`' character in the 1st position of the character array.

P. 28. Write a function in different way that will return f(7) = 4 or f(4)=7.

Solution

```
main()
{
    int n;
    n = show(7);
    printf("%d\n",n);
    n = show(4);
    printf("%d\n",n);
}
int show(int x)
{
    return x^3;
}
```

Explanation

The program basically asks for a solution which can produce 4 if the parameter is 7 and vice-versa, i.e., can produce 7 if the parameter is 4. In the solution we have a function which is defined in such a way that if we pass 4 then it returns 7 and if 7 is passed then it will return 4. The concept behind this is when we XOR 7 with 3 then the result is 4, similarly XOR 4 with 3 to get 7. So the function will return the value by doing an XOR operation with the number passed to it. This program is only applicable for the values of 7 & 4.

P. 29. Write a program to check whether the given unsigned number is a multiple of 3 or not without using division or modulo operators.

Solution

```
main()
{
    unsigned int number;
    int ret;
    printf("Enter a Number");
    scanf("%d",&number);
    ret = check(number);
    if(ret == 1)
        printf("%d is a Multiple of 3",
          number);
    else
        printf("%d is not a Multiple of
          3",number);
}
int check(unsigned int number)
{
    while(number > 3)
```

```
        number -=3;
    return (number==3)? 1 : 0;
}
```

Explanation

The logic behind the solution is that if we want to find whether any number is divisible by 3 then we must keep on subtracting 3 from the number as long as it is a value greater than 3. If the last value becomes less than 3, then the original number is not divisible by 3. If it is equal to 3, then the number is divisible by 3. In doing so we are only using the minus (–) operator and not the divide or modulo operator. The same logic can be used to find the divisibility of any number by any other number.

P. 30. Write a program to form the maximum number from the digits of a given number. For example, for 374 it is 743.

Solution

```
main()
{
    long int number, copy;
    int arr[8],x,y,digit,j=0;
    printf("Enter a Number\n");
    scanf("%ld",&number);
    copy=number;
    x=0;
    while(copy)
    {
        digit =copy % 10;
        if(x == 0)
            arr[x] = digit;
        else
        {
          for(j=x-1;j>=0 && digit        >
            arr [j];j— —)
                arr[j+1] = arr[j];
        }
        arr[j+1]=digit;
        x++;
        copy /= 10;
    }
    number = 0;
    for(j = 0;j<x;j++)
        number = number * 10 + arr[j];

    printf("The Required  number is %ld",
      number);
}
```

Explanation

Here we store each digit in the array in ascending order and then calculating the number through the 2nd loop

P. 31. Write a program to sort an array without using any condition.

Solution

```
main()
{
    int a[]={21,13,5,9,8};
    int size = 5;
    int i,j,temp;
    int avg, meandiff,eval;
    for(i=0;size-i;i++)
    {
        for(j=i;size-j;j++)
        {
            avg = a[j]+a[i];
            meandiff=a[j]-a[i];
            eval = 2* a[j]-abs(meandiff)-
              avg;
            if(eval)
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }
    for(i=0;size - i; i++)
    {
        printf("%d\t",a[i]);
    }
}
```

Explanation

Here we have swapped the array using insertion sort but while checking the condition we have used it in a bit different way. That is to check `i<size` we have used `size-i`, so that if value of `i` equals the size then their difference is zero and which means that the condition is false. Similarly, to check `a[i]>a[j]` we use three statements. That is we find the sum of both `avg` then we calculate the difference `meandiff` then calculate `eval` by substracting avg and the absolute value of `meandiff` from 2 multiplied with `a[j]`, then the condition value is stored in `eval`.

P. 32. How to return the largest sum of two contiguous integers in the array?

Solution

```
main()
{
    int  array[10]={10,20,30,11,16,1,-
      90,0,89,3};
    int x, m = 0,s=0,val1=array[0], val2=
      array[1];
    for(x=0;x<9;x++)
    {
        if(s<(array[x]+array[x+1]))
        {
            val1 = array[x];
            val2 = array[x+1];
            s = array[x] + array[x+1];
        }
    }
    printf("The Sum is of the memeber  %d and
%d is largest an sum is %d", val1, val2,s);
}
```

Explanation

Here the array contins some numbers in random order. We have to find the sum of two consecutive numbers present in the array. This can be possible by a loop which scans the array from the first element to the last but one element. In each iteration the xth element and the `x+1`th element of the array is added and the result is compared with the variable `s`, if s is small then `s` value is updated with the result.

P. 33. Write an efficient code for extracting unique elements from a sorted list of an array.

Solution

```
void find_unique(int array[],int lenght)
{
    int i,count = 1;
    for(i = 0;i<lenght-1;i++)
    {
        if(array[i]!=array[i+1] && count
          = = 1)
        {
            printf("%d",array[i]);
        }
        else
            count++;
        if(array[i] && count>1)
        {
            count = 1;
            if(i == lenght-2)
                printf("%d",array[i+1]);
        }
    }
}
main()
{
    int a[10] = {0,0,2,2,3,4,5,6,8,8};
    int lenght = sizeof(a)/sizeof(a[0]);
    find unique(a,lenght);
}
```

Explanation

The secret behind this solution is to find out the efficient way of spotting the unique element in a sorted array. Logic says if an array is sorted, then all those elements whose content does not match with the content of the very next element are unique. If they are not unique then the very next element must have the same content as the current one. In order to implement this, we have used a function `find unique` which takes two arguments – one is the array and the second one `n` is the length of the array. A loop will continue for `n` times where `count` counts the frequency of each character. If `count` is equal to 1 and the value is not equal to the next element then a unique element has been found out otherwise `count` is increased by one.

P. 34. Write a C program to reverse the words in the given string, such as "man is mortal"to "mortal is man" without using extra space.

Solution

```
void main()
{
    char a[30],*p;
    int x,k,c,f=0;
    p=a;
    printf("\nEnter a sentence ");
    gets(a);
    k=strlen(a);
    for(f=0;a[f]!=' ';f++);
    while(k>f)
    {
        c=0;
        while(p[k-1]!=' ')
        {
```

```
        for(x=k;x>0;x—)
            p[x]=p[x-1];
        p[0]=p[k];
        c++;
        }
        k-=c;
        p+=c;
        for(x=k;x>0;x—)
        p[x]=p[x-1];
        p[0]=' ';
        k— —;
        p++;
    }
    p[k]=0;
    puts(a);
    getch();
}
```

Explanation

The requirement is to reverse a string word-wise as depicted in the question. The string to be reversed is entered by the user and is stored in a character array `a`. We first calculate the position of the first space and store it in `f` and the string length in `k` and `p` points to `a`. Then we take a loop which continues till the values of `k` and `f` become same. Each time we circularly shift the characters towards the right until a space is found which signals the `end of word` and we update `p` to point to the end of the word. We continue the shift till the next space in the same manner and update the pointer `p` to the next space or `end of word`. When the loop terminates, the string `a` will contain the reversed string in terms of words.

P. 35. You have an unsorted array of some million random positive integers with no duplicates. Find two integers from this array which add up to 50.

Solution

```
void main()
{
    int  a[50],b[100];
    int  x,y,j;
    int  n;
    printf("\nEnter the length of the
  array: ");
    scanf("%d",&n);
    printf("\nEnter the elements : ");
    for(x=0;x<n;x++)
        scanf("%d",&b[x]);
    y=0;
    for(x=0;x<n;x++)
        if(b[x]<=50)
            a[y++]=b[x];
    n=y;
    for(x=0;x<n-1;x++)
        for(y=x+1;y<n;y++)
            if(a[x]>a[y])
                a[x]^=a[y]^=a[x]^=a[y];
    y=n-1;
    for(x=0;x<n && x<y;)
    {
        if(a[x]+a[y]= =50)
        {
            printf("\nThe two element are
        %d & %d",a[x],a[y]);
```

```
            x++;
        }
        if(a[x]+a[y]<50)
            x++;
        else
            y— —;
    }
}
```

Explanation

The secret behind this solution is not the algorithm to check for the summation of 50, the more important factor is how efficient and fast the algorithm is. Thus what we have done first is, we have selected all the numbers less than 50 and stored it in another array. The reason behind this is that all the numbers are positive and 2 numbers whose summation will have to be 50 must definitely be less than or equal to 50. Now our summation algorithm needs to work on a much reduced list of numbers and hence the efficiency increases. Now the numbers are sorted inside the second list. After sorting all the numbers we have set two counters – ‘x’ to the starting index of the array and the other one ‘y’ is set to the last index of the array. We go on adding the x and y elements and check whether the result is equal to 50 or not. If it is equal to 50 then both numbers are printed otherwise if it is less than 50, x will be increased otherwise y will be increased.

P. 36. Write a program to find out the square root of a number without using square root library function.

Solution

```
#include "stdio.h"
#include "math.h"
double sq_root(double n)
```

```
{
    return exp(log(n)/2);
}
main()
{
    double number;
    printf("Enter a number");
    scanf("%lf",&number);
    printf("The Square Root of %.2lf is
      %.2lf",number,sq_root(number));
}
```

Explanation

Instead of using the `sqrt` function we have used two library function which are defined in `math.h` header file. One of the functions is `log` and the other function is `exp`. The line `exp(log(n)/2)` gives the square root of a number.

P. 37. Write a program which produces a source code as output.

Solution

```
#include "stdio.h"
main()
{
    FILE *p;
    char ch;
    p = fopen("q63.c","r");
    while((ch=getc(p))!= -1)
    {
        putchar(ch);
    }
}
```

Explanation

Printing the source code of the program at the time of execution can be done in various ways. A simple method is that just the file (`test.c` in this example) within the same file and reading each character inside it and displaying it in the output.

P. 38. Write a program to read `n` lines from a file.

Solution

```
#include "stdio.h"
main()
{
    FILE *p;
    int f = 0, l=0,n;
    int a[30]={0,0};
    long int sum=0;
    char c;
    printf("Enter the value of n");
    scanf("%d",&n);
    p = fopen("a.txt","r");
    if(!p)
    {
        printf("File Is not found\n");
        printf("Press any key to exit");
        getc(stdin);
        exit(1);
    }
    while((c=getc(p))!=-1)
    {
        a[l]++;
        if((l>=n || l<f)&& c=='\n')
        {
            a[l]+=1;
            sum+=a[f];
            a[f]=0;
            f++;
            l++;
        }
        else if(c == '\n')
        {
            a[l]+=1;
            l++;
        }
    }
    sum+=a[f];
    printf("%ld %d\n",sum,l);
    fseek(p,sum,0);
    while((c=getc(p))!=-1)
    {
        printf("%c",c);
    }
}
```

Explanation

To print the last n lines from a file, we have to find the location of the nth line from the end of the file. Then we can shift the file pointer to that position and we can easily print the contents of those n lines. To find the exact location, we read each character in the file starting from the first to the last. While checking each character, count of the number of new line characters along with storing of number of characters present in each line in an array. When the counter reaches n, from that place onwards we go on calculating the sum from the first location and every time we come across a new line character in the file. When the pointer reaches the end of file then the sum contains the location from where the nth line from the last started.

P. 39. Write a program to implement circular left shift by specified number of rotations.

Solution

```
main()
{
    int rotatebits,number,y,i=1,noofbits;
    printf("Enter the value to be rotated:");
    scanf("%d",&number);
    printf("Enter the no of bits to be
    rotated:     ");
    scanf("%d",&rotatebits);
    noofbits = sizeof(number)*8-1;
    while(i<=rotatebits)
    {
        y = number>>noofbits;
        y = y & 1;
        number = number<<1;
        number = number|y;
        i++;
    }
    printf("The Rotated Value is:%d\n",
    number);
}
```

Explanation

The basic logic of the solution and also of circular left shift is that the MSB is shifted to LSB. In the given program code, when `y=x>>nofbits` executes, assuming that `int` is of two bytes, the value of `x` is right shifted by 15 `(sizeof(x)*8-1 = 2*8-1 = 15)`, which means, in case of positive `x` values the value of `y` will be 0 and in case of negative `x` value, `y` will be 1. With a bitwise and operation with `y` and 1. Subsequently, `x` is left-shifted by 1 and we do a bitwise or with `x` and `y` and store the resulting value in `x`. This will do a circular left shift of 1 bit only, but if we want a variable number of rotations, the same operation is repeated `n` number of times.

P. 40. How to generate floating point random numbers?

Solution

```
#include "stdio.h"
#define F_WIDTH 10
main()
{
    int integral, decimal,no_times;
    float flt_value,random_no;
    printf("Enter Any folating point
    Number");
    scanf("%f",&flt_value);
    do
    {
        printf("Enter how many random U
          want To generate  :");
        scanf("%d",&no_times);
        if(no_times<=0)
        {
            printf("Enter a Positive
              value");
        }
    }
while(no_times<=0);
    while(no_times)
    {
        integral =flt_value;
        decimal = (flt_value-integral) *
          F_WIDTH;
        random_no = (random(integral) +
          (float)random(decimal))/
            F_WIDTH;
        printf("%f\t",random_no);
        no_times− −;
    }
}
```

Explanation

Generally, we have library functions like `random` or `rand` which are capable of generating random numbers for any integer type value but not for floating type values. Keeping this in mind we have first divided the given `float` value into two parts, the integral and the decimal part. It's evident that individually the integral and decimal parts are integral in nature. Now, we can separately find the random numbers corresponding to the integral and decimal parts using the `random` function and then join them to get a randomized `float` value.

P. 41. How would you find the last (`nth`) element of a linked list?

Solution

```
void find_nthlastpos(struct node *base, int
npos)
{
    int i;
    struct node *ptr= base;
    for(i = 1 ; i < npos && ptr; i++)
        ptr = ptr -> next;
    if(npos <= 0 || !ptr)
    {
        printf("\nInvalid Position\n");
        return;
    }
    while(ptr -> next)
    {
        ptr = ptr -> next;
          base = base -> next;
    }
    printf("\n....%d...",base->data);
}
```

Explanation

In the above function `find_nthlastpos`, it takes two parameters, one is the base address of the linked list and the second one is the position from the last that is to be found. At first we move the pointer `ptr` from the beginning to the position given by the user. So now `ptr` is present at the `nposth` position from the beginning. Then we move both the pointer `ptr` and base simultaneously till `ptr` reaches the last node. At the moment `ptr` reaches the last node, the base will be at the nth node from the last.

P. 42. How to delete a node from a linked list if address is given, but its previous address is not given?

Solution

```
void Delete_node(struct node *cur )
{
            struct node *front;
            if( cur -> next = = NULL)
            {
                printf("\n This function
                cant delete the last
                node\n ");
                return;
            }
            front = cur -> next;
            cur -> data = front -> data;
            cur -> next = front  -> next;
            free(front);
}
```

Explanation

This function `delete_node` takes parameter as the address of the node which is to be deleted. Now in order to delete that node we require the address of it's previous node. So that we can store this address into the address part of the node `next` to the one being deleted and then free the node to delete. But here we only have the address of the node to be deleted. So what we have done here is, simply copied the content of the next node and stored it in the current node (of which we have the address and is supposed to be deleted) and then deleted the next node as usual.

P. 43. How to write a function that can reverse a linked list?

Solution

```
    reverse(struct node *base)
    {
        struct node *back, *cur, *front;
        front = base;
        cur = NULL;

        while(front)
        {
            back = cur;
            cur = front;
            front = front -> next;
            cur -> next = back;
        }
    }
```

Explanation

In the solution we have taken 3 pointers – `back`, `cur` and `front`. The main objective of the above function is to store the address of the previous node in the address part of current node so that the list will get reversed. At first we assign the base address of the list passed to this function, into `front` and `cur` to `NULL`. Then we start a loop that runs till the last node has not been processed.

The first line `back = cur` assigns `NULL` to `back`. The second line `cur = front` will assign the base address to `cur` and the line `front = front -> next` makes `front` point to the next node. Now `cur -> next = back` will store NULL in the address part of the first node which indicates that this is the last node. Thus the first node has now become the last node. This process will continue and each time the node pointed by `cur` will progress by storing the address of the previous node pointed by back. Every time the next node address comes from `front-> next` till front points to the last node whose next part contains `null`, thereby assigning `null` to `front`, which results in the loop termination.

P. 44. How to check whether a given linked list is circular or not? (Use only one pointer to increase efficiencies)

Solution

```
iscircular(struct node *base)
{
    while(base -> next && base != base -
    > next)
        base -> next = base -> next ->
        next;
    return(base -> next == base);
}
```

Explanation

We have taken a loop which will continue till the last node. Here we skip the nodes to the next of the base and assign the next to next node of base to base next pointer.

And continue till the condition and after the loop we check if the base next pointer will point to base then there is a cycle otherwise no cycle.

P. 45. Write a recursive function to find the maximum number from an arbitrary array.

Solution

```
#include "time.h"
#include "stdio.h"
main()
{
    int magnitude,*array,loop,max;
    printf("Enter The magnitude of the
    array  :");
    scanf("%d",&magnitude);
    array=(int *)malloc(magnitude);
    srand((unsigned)time(NULL));
    printf("The generated array is :");
    for(loop = 0;loop<magnitude;loop++)
    {
        *(array+loop) = rand()%256;
        printf("%d\t",*(array+loop));
    }
    max=find_max(array,magnitude);
    printf("\n The Maximum Element from
  the array is :%d\n",max);
}
int find_max(int *array,int n)
{
    static int max;
    if(n==0)
        return;
    if(*array>max)
        max = *array;
    find_max(array+1,—n);
    return max;
}
```

Explanation

To calculate the maximum number from an arbitrary array we can use a recursive function `find_max` as defined above. This function takes two arguments, one is the array name and the second one is the number of elements present in the array. Now we call this function recursively for `n` times and for every time, we check the value present at each index with `max`. In a situation where the number is greater than the value of `max` we will assign it to `max`. At the end we return it to the calling function. The variable `max` has been declared as static for retaining the value till the end of the program.

P. 46. Write a program to find the largest palindrome of a given string.

Solution

```
is_palindrom(char *str, int len)
{
    int l=0;

    for(len—;str[l]==str[len];len—,l++);
    if (l < len ) return 0;
        return 1;
}

int find_lasrgest_palindrom(char *source, char
*big_buf)
{
    int len = strlen(source);
    int ii;
    int pal_len;
    int j;
    int num_substr;
    for(pal_len=len;        pal_len>0;
    pal_len—)
        {
        num_substr = len - pal_len + 1;
        for (j=0; j<num_substr; j++)
            {
            if (is_palindrom(source + j,
            pal_len))
            {
            memcpy(big_buf,    source+j,
            pal_len);
```

```
                big_buf[pal_len] = 0;
                return pal_len;
                }
                }
            }
        return 0;
}

void main()
{
    char st[20],st1[20];
    int x;
    printf("\nEnter the string : ");
    gets(st);
    x=find_lasrgest_palindrom(st,st1);
    printf("\nThe length is %d and string
      %s",x,st1);
}
```

Explanation

Here we are finding the largest string which is a palindrome. We are using two functions, one is `is_palindrome` and `find_largest_palindrome`. The function `is_palindrome` takes two arguments, one is the string and the other is the buffer to store the string, which checks whether the the string passed to it is a palindrome or not. Then the function `find_largest_palindrome` is used to find the largest palindrome. Here in this function we use two loop. The outer loop's counter starts from the length of the string passed to it and the loop continues till counter becomes zero. And in each iteration of the outer loop the inner loop finds all possible strings of length `pal_len` and each string is checked for the palindrome. If a string is found to be palindrome then function returns the length of the string. And the buffer `st1` contains the largest palindrome string.

P. 47. Write a method to find out the `k`th largest element in an array having `n` number of distinct elements.

Solution

```
#include "time.h"
#include "stdio.h"
main()
{
    int magnitude,*array,loop,nthlarge,
      element;
    printf("Enter The magnitude of the
    array  :");
    scanf("%d",&magnitude);
    array=(int *)malloc(magnitude);
    srand((unsigned)time(NULL));
    printf("The generated array is :");
    for(loop = 0;loop<magnitude;loop++)
    {
        *(array+loop) = rand()%256;
        printf("%d\t",*(array+loop));
    }
    printf("Enter Which Largest element
            u Want to search  :");
    scanf("%d",&nthlarge);
    element = find_nthlargest (array,
                  magnitude, nthlarge);
    printf("The %d largest Element is %d",
nthlarge,element);
}
```

```
int  find_nthlargest(int  *array,int  len,  int
large)
{
    int outer, inner,temp;
    for(outer  = 0; outer<len; outer++)
    {
        for(inner=0;  inner<len-outer-1;
          inner++)
        {
            if(*(array+inner)>*  (array  +
              (inner+1)))
            {
                temp  =  *(array+inner);
                *(array+inner)=*(array+
              (inner+1));
                *(array+(inner+1))=  temp;
            }
            if(outer == large)
                break;
        }
    }
    return  *(array+(len-large));
}
```

Explanation

The above function takes 3 parameters.

1. The array having distinct elements
2. No of elements preset in the array`(n)`
3. Which largest number do u want to find`(k)`.

So here we implement the logic of bubble sort. The bubble sort algorithm says that after first iteration the largest element is in its appropriate position. So we are just required to perform `k`th iteration. So that after kth iteration the `k`th largest number is at its original position. So when the value of `j` is equal to `k` we come out from the loop, and return `a[n-k]` which is nothing but the `k`th largest number.

P. 48. What is the efficient way to find the middle element of a linked list?

Solution

```
find_middle(struct node *base)
{
    struct node *ptr , *dptr;
    ptr = dptr = base;
    while(dptr && dptr -> next)
{
    if(dptr -> next -> next)
        ptr = ptr -> next;
        dptr = dptr -> next -> next;
}
return(dptr -> data);
}
```

Explanation

Single linked list is a list which consists of some nodes and there is a link between the nodes. It is a one-way communication hence we can go on visiting node by node through the entire link starting from the base node to the last. The position of the middle node is nothing but the total number of nodes divided by two. In order to get the middle node without traversing the entire list is done by using two pointers at the list.

Set two pointers to the base address of link list. Move one pointer node by node and move the other pointer two nodes at a time. The pointer moving two nodes at a time will reach the end of linked list, while the pointer moving one node at a time will reach the middle of the linked list.

P. 49. Write a program to find whether a loop is present in a linked list or not.

Solution

```
struct xx
{
    int val;
    struct xx * n;
};
int len=0;
struct xx * create()
{
    struct xx *q=(struct xx *)malloc
      (sizeof(struct xx));
    printf("\nEnter the val :");
    scanf("%d",&(q->val));
    q->n=0;
    len++;
    return q;
}
void main()
{
    struct xx *b=0,*p;
    char c=1;
    int x=0;
    p=create();
    b=p;
    printf("Press Esc to terminate ");
    while((c=getch())!=27)
    {
        p->n=create();
        p=p->n;
        printf("Press Esc to terminate");
    }
    p->n=b->n;
    p=b;
    while(p && x<=len)
    {
        p=p->n;
        x++;
    }
    if(x>len)
        printf("\n\n\tThere is a loop ");
    else
    printf("\n\n\tThere is no loop");
}
```

Explanation

Here we are taking a counter which counts the number of nodes present in the linked list at the time of creation. We make a loop by pointing the last node's address to the second node. After that we traverse the linked list and if there we fid more than `len` number of nodes then the list contains loop.

P. 50. Write a program without using any control structure, to check whether the given number is EVEN or ODD.

Solution

```
main()
{
    char p[2][5] = {"EVEN","ODD"};
    int number;
    printf("\nEnter A number");
    scanf("%d",&number);
    printf("%s\n",p[number&1]);
}
```

Explanation

We can know that a number is odd or even by checking the LSB of the number. If the LSB contains 1 then the number is odd, otherwise the number is even. So, to get the value of LSB we have to mask the LSB by bitwise AND with the number 1. The result decides whether the number is even or odd. To avoid conditional statements we have taken a two dimensional character array, where we have stored even and odd as strings and according to answer we print the strings fetched by the index of array.

P. 51. How to write a program to check whether the given name is a macro or function?

Solution

```
include "stdio.h"
#define STR(X) #X
#define STR2(X) STR(X)
#define TEST_IT(X)\
do{\
    if(strcmp(#X "(0)",STR2(X(0)))==0)\
        printf(STR(X)"IS A FUNCTION\n");\
    else\
        printf(STR(X)"IS A MACRO\n");\
}while(0)
#define MAX 5
```

```
main()
{
    TEST_IT(main);
    TEST_IT(printf);
    TEST_IT(putchar);
    TEST_IT(MAX);
}
```

Explanation

A macro name or call is always expanded before compilation or during the generation of intermediate file. But function name or call is never expanded throughout the program. In case of function call the control is transferred from the called program to the function definition and after execution, the control returns to the calling function. Whereas in case of a macro there is no such facility and it executes the expanded codes on the spot without any transfer of control. This forms the secret behind the solution.

Hence, the given solution differentiates between macro and function on the basis of the logic that the macro name gets expanded. On the other hand a function name does not expand or convert into a different expression. In the solution we have used three new macros to show it. One is `STR` which converts the name to string by using # which is a string formation operator. `STR2` just calls the `STR` macro by passing the same value. And the `TEST_IT` uses both the macros to check the condition for function name or macro name. So, the macro `TEST_IT` checks whether the name passed to it expands or not. We use strcmp to compare, where the first parameter always evaluates as a constant string which could either be the function or the macro name. But the second parameter evaluates as a function name to be executed or in case of a macro name it will expand. This means in case of printf, the first and second parameters remain the same, whereas in case of putchar the two parameters differ because the second parameter expands into the macro definition and as the macro name cannot be same as the definition, hence the strcmp will return non-zero. Accordingly, the if statement will evaluate and distinguish between Function and Macro name.

P. 52. What can you do in C but not in C++?

Solution

```
main()
{
    int class = 8;
    int template = 9;
    printf("%d",class + template);
}
```

Explanation

Everything which is possible in C can be used in C++ but C++ also has some extra properties which are not present in C. Hence, C can be considered to be a subset of C++. This means that we can write statements, which can be executed in C but not in C++. For example, as shown in the solution, we declare a variable with a name that is a keyword of C++ but is not present in C. Let us say, we declare a variable called class, which is a keyword in C++ but not in C.

P. 53. Calculate the generic root of a number in a one-line statement.

Solution

```
main()
{
    int num,x;
    printf("\n ENTER A NUMBER");
    scanf("%d",&num);
    printf("\nThe Generic Root of the
number %d is %d",num, find_generic(num));
}
int find_generic(int num)
{
    int rem;
    return (rem = num % 9)?rem:9;
}
```

Explanation

The generic root of a number is the repetitive sum of the digits in the number until the result is a single digit. If the number is 23 then the generic root is 2+3=5, for 79 the generic root is 7+9=16 but 16 is not a single digit number, so we have to find the sum of the digits again, which is 1+6=7. So the generic root of 79 is 7. In the solution we find the remainder of the division of the number by 9 and that is exactly the generic root which is our answer. But there is a special case, where, if the given number is itself a multiple of 9, then the above formula does not work. That's why we are checking whether the number is a multiple of nine or not, if it is then we simply return 9 otherwise the reminder.

P. 54. Write a one-line statement to find (-1)^n.

Solution

```
main()
{
    int x,n;
    printf("Enter the value of n   :");
    scanf("%d",&n);
    x = n & 1 ? -1:1;
    printf("%d",x);
}
```

Explanation

The value of -1 to the power n depends upon the value of `n`. If the value of `n` is odd then the result is `-1` otherwise `1`. So far, we were checking whether `n` is an even number or odd. The statement `n&1` gives the content of LSB bit along with resetting all the remaining bits to zero. So, if the content of LSB bit is zero `n` is an even number, otherwise it is odd. So the result of `n&1` is either `1` or zero. So keeping this thing in mind we have written a line `n&1?-1:1` which results either `1` or `–1`. which is the required result.

4. Short-answer Questions with Answers

DATA TYPES

Q. 1. What are the data types that are missing in C?

Answer

Data types that are available in other programming languages, but not available in C, are missing in C.

Q. 2. Which programming language does not support data type?

Answer

All the scripting programming languages, such as PHP, Perl, Java Script, Shell script, etc., do not support data types.

Q. 3. What is the disadvantage of a programming language not supporting data types?

Answer

If a programming language does not support data types, then new data structure and new implementation is not possible in that programming language.

Q. 4. What is the difference between NUL and NULL?

Answer

NUL is the very first character in the character set in C, but NULL is macro which represents Null the pointer.

Q. 5. Why is negative (-ve) data stored in the memory in the form of 2's complement?

Answer

The negative data stores in the memory in the form of 2's complement, only to overcome the -0 (which is undefined).

Q. 6. How to input null character from the keyboard?

Answer

Input of null character from the keyboard is done by typing CTRL+@.

Q. 7. If float variable is incremented beyond its maximum range, what will be the output?

Answer

Incrementing the float variable beyond its maximum range is: +INF.

Q. 8. What is the difference between data types and modifiers?

Answer

Data types decide the size and modifiers decide the range of the variables.

Q. 9. Why is cycle seated in character data types?

Answer

It is because character type has the cycle, that any kind of files which contain uni-code characters can be processed in C.

Q. 10. What is the difference between little-endian and big-endian system?

Answer

Endianness is the process of byte ordering. In little endian, data is stored in DCBA order, whereas in big endian, data is stored in ABCD order.

OPERATORS

Q. 11. What is the difference between precedence and associativity?

Answer

Precedence decides which operation should be performed first, while associativity decides the order of evaluation if more than one operators enjoy the same priority.

Q. 12. What is return value of relational operators?

Answer

All relational operators return values of 1 or 0.

Q. 13. Which of the POLISH notation precedence is not allowed: (a) Infix, (b) Postfix, (c) Prefix?

Answer

In case of both, Postfix and Prefix expressions, Precedence is not allowed.

Q. 14. Which data structure is used to convert an arithmetic expression?

Answer

Stack.

Q. 15. What is the difference between '&' and '&&' ?

Answer

'&' is bit-wise 'and' operator which only works in integer type to check whether a particular bit is ON or OFF, and return 1 if both the bits are 1, or return 0 if any of the bit is 0. But '&&' operator is a logical operator. It returns 1 if both operands are true value, and returns 0 if any of the operands is false value.

CONTROL STRUCTURES

Q. 16. What are the control structures in C?

Answer

For, while, do while, switch case, if else, goto, break and continue.

Q. 17. How many statements fall under the default scope of a control structure?

Answer

A single statement falls under the default scope of the control structure.

Q. 18. Should goto statement be used in the program or not?

Answer

If "goto" statement is used there is no problem in the program, but it is suggested not to use it because it is violation of ANSI rules.

Q. 19. Which of the following loops is exit control loop: (a) For, (b) While, (c) Do while?

Answer

"Do while" loop is known as exit control loop.

Q. 20. What is the difference between loop and function recursion?

Answer

Loop does not allocate memory, but function recursion allocates memory from the stack.

Q. 21. What is hanging if statement in C?

Answer

The statement which is neither part of "if" nor part of "else", but it is on fly is known as hanging if statement. Hanging if statement is not allowed in C.

Q. 22. Of the "if else" and "switch case" statements, which executes faster?

Answer

'Switch' statement is faster than the 'if else' statement, because in switch statement execution starts from the match case. The if else execution starts from the beginning of if.

Q. 23. What is the difference between break and continue?

Answer

Break transfers the control to outside of the loop but continues to transfer the control to the beginning of the loop.

Q. 24. What is the difference between `for` and `while` loops?

Answer

If the number of iterations is known in advance, then the `for` loop is used. If the number of iterations is not known in advance, then `while` loop is used.

POINTERS

Q. 25. Which concept is called “dereference”?

Answer

Doing read or write operation in memory using the help of pointer is called dereference.

Q. 26. What does maximum address by a pointer refer in TC and GCC compilers?

Answer

In Turbo C pointer maximum refers up to 1 MB of memory, whereas in GCC pointer maximum refers up to 4 GB of memory.

Q. 27. What is null pointer?

Answer

When a pointer refers initial address in memory or 0th address in memory it is called null pointer.

Q. 28. What is wild pointer?

Answer

When a pointer refers unauthenticated address in memory it is called wild pointer.

Q. 29. What is dangling pointer?

Answer

A reference that does not actually lead anywhere.

Q. 30. What are the different reasons for segmentation fault in C?

Answer

When segmentation fault occurs in a program, the program is simply terminated. There are different reasons for segmentation fault, such as:

- Dereferencing a wild pointer
- Working beyond array size
- Stack overflow
- Attempts to free array memory

Q. 31. What are near and far pointers in C?

Answer

Far and near pointer is only introduced in Turbo C compiler. When the pointer refers to an address in the same segment it is called near pointer, but when it refers to an address in another segment it is called far pointer.

Q. 32. What are the applications of pointer in C?

Answer

- Dynamic memory allocation
- Array implementation
- Directly accessing the hardware address
- Calling convention of function

Q. 33. When does core dump occur in C ?

Answer

A process dumps core when it is terminated by the operating system due to a fault in the program. The most typical reason for its occurence is that the program has accessed an invalid pointer value. Given that you have a sporadic dump, it is likely that you are using an uninitialized pointer.

Q. 34. What is the difference between malloc and calloc functions?

Answer

Malloc memory allocation is equivalent to a single-dimensional array but calloc memory allocation is equivalent to a double-dimensional array.

Q. 35. What is meant by static and dynamic memory allocation in C?

Answer

Memory allocation in a program is only possible in load time or rum time of the program. The memory allocated at compile time is known as static allocation. That allocated at run time is called dynamic allocation.

Q. 36. Why is pointer arithmetic required?

Answer

Pointer arithmetic is required only to access a particular memory address.

Q. 37. Define "memory leak" in a one-line statement.

Answer

At any moment if a pointer loses reference of a memory block in the heap it is called memory leak.

ARRAYS

Q. 38. What is the relation between arrays and pointers?

Answer

Array name itself is a pointer which refers the initial address of an array.

Q. 39. What is the difference between an array and a linked list?

Answer

Array and link list both are used.

Q. 40. What is the purpose of double-dimensional array?

Answer

Double-dimensional array is suitable to implement the non-linear data structure, such as graph and tree.

Q. 41. Which of the two is a faster searching mechanism—"Binary search" and "linear search"?

Answer

Linear searching is faster than binary search, if both are implemented using a linked list. But binary searching is faster than linear searching if both are implemented using an array.

Q. 42. Traversing double-dimensional array using row order or column order, which is one is the faster mechanism?

Answer

Row order traversal of double-dimensional is faster than column order traversal, because row order traversal is sequential traversal, whereas column order traversal is not sequential type.

Q. 43. If base address of a double-dimensional array is given, how do we get the address of a particular element in an array?

Answer

(*Note*: If int*[5][5], &*[0][0] is given as 500, what will be address of &*[3[4])

To get the address of a particular element in a double-dimensional array, the following formula is used.

Address = base address + (row number * column size + column number) * size of the array element.

Q. 44. What will be size of any array in C?

Answer

The size of an array depends on the size of the segment divided by the size of the array element.

Q. 45. What is the application of an array?

Answer

Application of an array is to implement different data structures, such as stack, queue, tree, graph and searching, sorting.

Q. 46. What is the difference between character array and string?

Answer

String is a character array terminated with null character '\0'. So, a string is a character array but a character array is not string.

Q. 47. What is the difference between strcpy and strncpy?

Answer

Strcpy copies the entire string with null character, but strncpy copies only the specified number of characters and does not copy the null character.

Q. 48. What is the difference between char s[] and char *s?

Answer

Char s[] is an array. When a string is assigned to 's', the size is automatically taken by the compiler looking at string size. But in char *s, 's' is a pointer. When a string is assigned to 's' the size does not depend on the string. Its size is fixed which is 4 bytes in GCC and 2 bytes in Turboc. That's why char s[] will be dereference, but char *s cannot be dereference.

FUNCTIONS

Q. 49. What is a function?

Answer

A function is set of statements which are encapsulated together to perform a specific task. They are seated separately in a file, which are compiled independently and loaded in memory when they are required.

Q. 50. What is the use of library function?

Answer

Library functions are the wrappers of the system calls, which are seated in a separate file called share object file. They are loaded and unloaded in memory depending on the program.

Q. 51. What is the difference between library function and system calls?

Answer

Library functions are implemented in user space memory, but system calls are implemented in kernel space memory. All the library functions are finally converted into system calls.

Q. 52. What is the disadvantage of writing the function body below or above the main function?

Answer

If we write the function body below or above the main function in the same file, the function body cannot be reutilized in other programs.

Q. 53. How can we find the size of the function?

Answer

Size of the function is calculated as follows:

(a) Size of local automatic variables used in function +

(b) Size of the actual parameters used in the function +

(c) Size of the next address of the function +

(d) Size of the previous address of the function.

Q. 54. What is function recursion?

Answer

When a function is called to itself it is known as function recursion.

Q. 55. What is the difference between return and exit?

Answer

‘Return’ and ‘exit’, both statements are used to terminate the child process and send a signal to parent process. Return is a keyword, whereas exit is a function.

Q. 56. Which data structure is implemented in function recursion?

Answer

Stack data structure is implimented in function recursion.

Q. 57. What is the major difference between stdout and stderr?

Answer

If a program uses stdout, the output of the program can be redirected. But when the program uses stderr, the output of the program cannot be redirected.

Q. 58. What is the major difference between stream and buffer?

Answer

Buffer is a block of memory allocated by operating system in kernel space for IO operations. But stream is a mechanism which establishes the connection between a buffer and a device.

Q. 59. When is a "copy of the variable" created ?

Answer

There are two different types of function calling conventions, such as 'called by value' and 'called by address'. Copy of a variable is created when it is called by value.

Q. 60. How do we access a variable in other functions without parameter technique?

Answer

If a variable is defined in one function, it cannot be accessed in other functions. To access the variable value in other functions we take the help of stack. See the example as follows:

```
Main()
{
   Int x=90;
   Show();
}
Int show()
{
   Int y;
   Printf("%d ",*(&y+3));
}
```

Q. 61. What is the difference between 'call by value' and 'call by address'?

Answer

Both are used for function calling convention. But the major difference between these is:

Call bye value: If any modification done in calling function does not affect caller function, it is call by value.

Call bye address: if any modification done in calling function affects caller function, it is call by address.

Q. 62. What is the difference between static library and dynamic library?

Answer

When the program is linked with static library, all the symbols present in static library may be used or may be not used, but all are copied to the program and all symbols are resolved at linking time. But when the program is linked with dynamic library, all the symbols present in the dynamic library are not copied; only those symbols are used which are copied and they are resolved at run time of the program.

Q. 63. What is the difference between dynamic library and dynamic loaded library?

Answer

Dynamic library: When a program is loaded along with library is not loaded, so program and library both are loaded and unloaded separately.

Dynamic loaded library: When program is loaded along with library is loaded and when program is unloaded along with library is unloaded.

Q. 64. What is the difference between void main and int main?

Answer

In case of void main child process does not send a signal to parent process, but in case of int main child process sends a signal to its parent process.

STORAGE CLASSES

Q. 65. What is a storage class?

Answer

Storage class is a data structure used by every C compiler which decides the scope, life, default initial value and storage of a variable and functions.

Q. 66. What is the difference between program scoping and file scoping?

Answer

Program scoping: If a variable or function is defined in one file, and that can be accessed in another file of a C program, it is know as program scoping.

File scoping: If a variable or function is defined in one file, and that cannot be accessed in another file of a C program, but only can be accessed in the same file, it is know as file scoping.

Q. 67. What is the difference between writing static as local variable and global variable?

Answer

Static is used in two different places, such as:

To avoid the re-initialization: When static is used as the local variable.

File scoping: When static is used as global variable.

Q. 68. Why does a pointer not refer to a register variable?

Answer

Registers are used as memory element of the microprocessor. They have no address. So, pointer does not refer to a register variable.

Q. 69. What is the use of static storage class in C?

Answer

Static is used in two different places, such as

- To avoid the reinitialisation : when static is used as the local variable
- File scoping: when static is used as global variable.

Q. 70. What are the different scope rules in C?

Answer

Different scope rules in C are:

- Program scoping
- File scoping
- Function scoping
- Block scoping

PREPROCESSORS

Q. 71. What are the different preprocessor directives in C?

Answer

The following are preprocessor directives used in C:
#if, #else, #endif, #elif, #define, #undef

**Q. 72. Which of the following produce the object file:
(a) Preprocessor (b) Compiler
(c) Assembler (d) Linker?**

Answer

Assembler.

Q. 73. What is macro?

Answer

Macro is the symbolic name of longer construct.

Q. 74. What is the difference between macro call and function call?

Answer

Macro call is process before compilation , but function is process during run time of the program.

Q. 75. What is the difference between writing header file using " " and < > ?

Answer

If the file is included using <>, such as <stdio.h>, the preprocessor searches the included file from the standard include path, but if the file is included using " ", such as "stdio.h", the preprocessor searches the included file from standard include path and current working directory.

Q. 76. What is the difference between startup and exit pragma?

Answer

Any function is included in startup pragma, that is executed first before control goes inside main function, but any function is included in exit pragma, that is executed once control leaves the scope of the main function.

Q. 77. What is the use of C preprocessor?

Answer

C preprocessor processes some parts of the C program before compilation.

STRUCTURES AND UNIONS

Q. 78. What is the self-referential structure?

Answer

When a structure is nested within the same structure and the nested structure variable is a pointer it is known as self-referential structure.

Q. 79. What is slack byte of the structure?

Answer

Every structure allocates memory in the form of a block. So, when the data member of the structure does not properly accommodate in the block, a few bytes of memory are lost in that block. This is knows as slack byte or undefined byte of the structure.

Q. 80. What is the difference between a structure and a union?

Answer

Structure:

1. Each data member of a structure begins at different location,
2. Size of the structure is the size of all data members.

Union:

1. Each data member of a structure begins at the same location,
2. Size of the union is the size of the longest data member.

Q. 81. Why cannot two structure variables be compared?

Answer

Two structure variable cannot be compared because of slack byte.

Q. 82. What do you mean by "active data member" of a union?

Answer

The first data member of a union is known as the active data member, which should be the longest data member of the union.

Q. 83. What is the application of structures?

Answer

The application of structures lies in:

- Creating memory link
- Data encapsulation
- Bit field

Q. 84. What is the application of unions?

Answer

Application of union is in creating a share memory , so locking mechanism will be implemented.

Q. 85. What is the use of bit field?

Answer

Bit field is created using a structure. It is used to create the protocol header and developing different fonts.

FILES

Q. 86. What is the difference between EOF and End Of File?

Answer

EOF is a macro , whose expansion value is -1, whereas End Of File is the ASCII value of CTRL+Z character (26) , which represents the end of file character.

Q. 87. What is the difference between “w” and “w+”?

Answer

“w” and “w+” both are used as file opening modes. In “w” mode, the program only does write operations, but in “w+” mode the program does both read and write operations.

Q. 88. What are the different types of files?

Answer

Linux operating system supports seven different types of file, such as:

- Regular files
- Directory file
- FIFO file
- Character special file
- Block special file
- Link file
- Socket file

Q. 89. What are the different modes of file opening?

Answer

There are three different modes of opening a file, such as,

- Read mode
- Write mode
- Append mode

Q. 90. What is the difference between fseek and lseek?

Answer

Both are used to transfer the control to any position of a file, whereas the fseek is a standard library function and lseek is a system call.

Q. 91. What is the difference between file and directory?

Answer

File is name reference to an inode number in disk, where it contains data. Directory is also the name reference of inode number in disk, where it contains file and directory.

Q. 92. What is file system?

Answer

File system is the data structure used by the operating system to organize files and directories.

Q. 93. What are the active file pointers in C?

Answer

In linux, the file system supports two different active file pointers in C, such as *__IO __ptr_read and *__IO__ptr_write.

Q. 94. What is the maximum size of a file?

Answer

The size of a file depends on the file system used by the operating system.

Q. 95. What is the difference between file and FILE?

Answer

File is a container of data whereas FILE is the typedef name of a structure which is defined in stdio.h header file.

COMMAND LINE ARGUMENTS

Q. 96. What is the use of a command line argument?

Answer

Every application program needs interface in the form of text or graphic or windows. A command-line interface is a means of interacting with a computer program where the user issues commands in the form of successive lines of text (command lines).

Q. 97. How are " main(void)" and "main(int)" different?

Answer

Writing void inside the main function means the program does not take any input through command line, but writing int inside main means the program can take input through command line.

Q. 98. Explain c and v in argc and argv with their purpose.

Answer

In C, we can supply arguments to 'main' function. These are called command line arguments, and are supplied at the time of invoking the program. Their forms are:

```
main (int argc, char *argv[]) { }
```

'argc' is called 'argument counter'. It represents the number of arguments in the command line. 'argv' is 'argument vector'. It is an array of char type pointers that points to the command line arguments. Size of this array is equal to the value of argc.

Q. 99. How can we redirect the output of one program as input of another program?

Answer

Only by using command line argument.

Q. 100. What is the application of environment vector in command line argument?

Answer

It is useful to communicate with a program in a semi-permanent way, so specifying a command-line option every time is not needed while typing the command to execute the program.