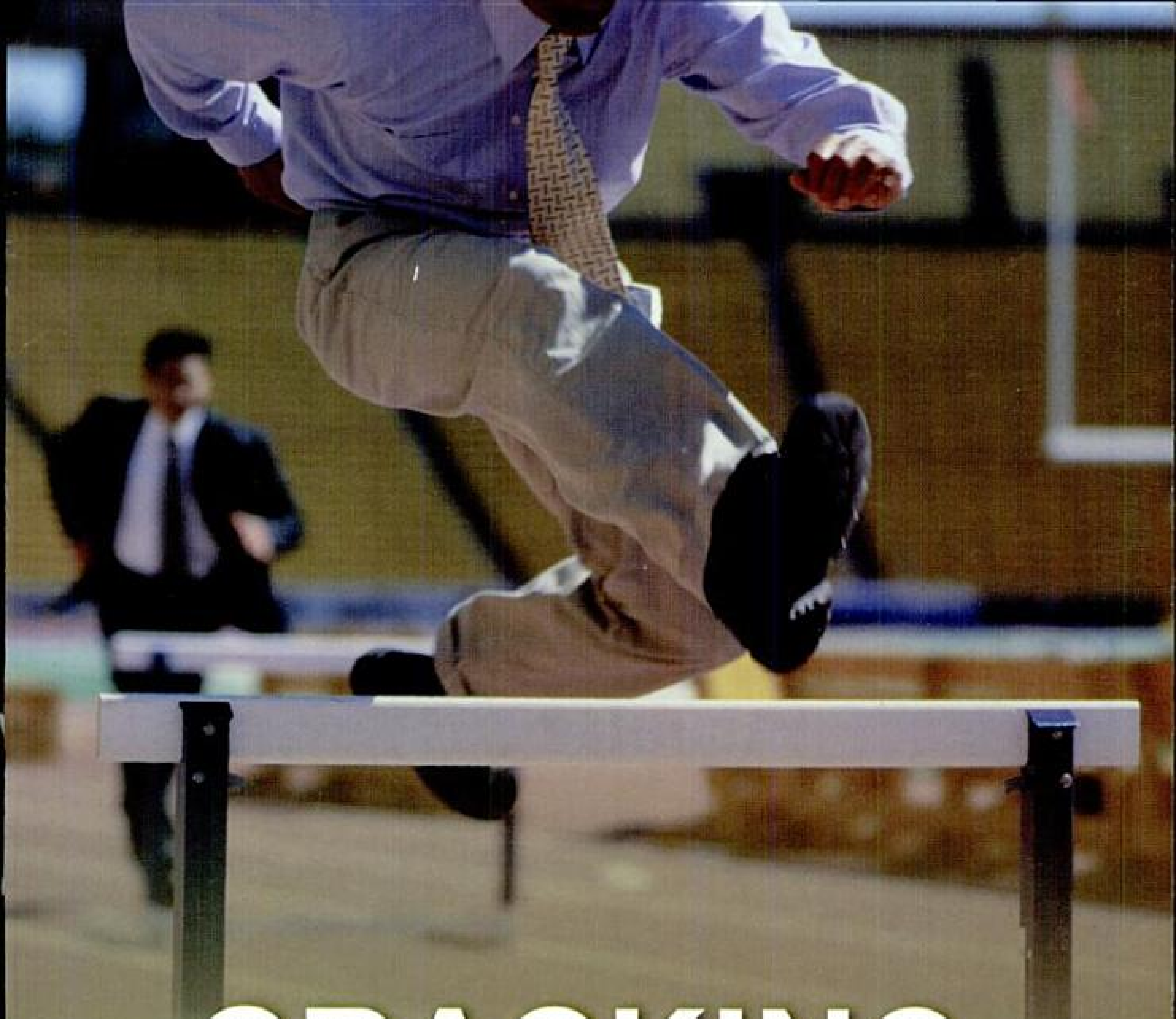

CRACKING *the* C, C++, *and* Java INTERVIEW

S G GANESH

Tata McGraw Hill

Published by Tata McGraw Hill Education Private Limited,
7 West Patel Nagar, New Delhi 110 008.

ISBN (13): 978-0-07-007792-8
ISBN (10): 0-07-007792-4

Typeset at The Composers, 260, C.A. Apt., Paschim Vihar, New Delhi 110 063 and printed at Rashtriya Printers, M-135, Panchsheel Garden, Naveen Shahdara, Delhi-110 032.

Cover Design: Kapil Gupta

Cover Printer: Rashtriya Printers

The McGraw-Hill Companies

For entry-level jobs, the IT industry requires students and young professionals with good programming and problem solving skills. For jobs that require programming skills, most of the IT companies (both products and services companies) look for candidates with good C, C++, Java skills (also C# skills, but to a lesser extent) in addition to skills in writing and using various data-structures and algorithms. There is no doubt that technical knowledge and skills in various specific domains are very important (such as Oracle, Linux, distributed computing, network programming, etc.); the key point is that programming skills are essential for getting almost any software related job in IT companies (particularly product-based companies such as IBM and HP).

A lot of programming books are already available in the market, but most of the books are tutorials or textbooks on programming languages. It is difficult to crack an IT interview by just reading any of these books or by writing a few programs. IT interviews focus on problem solving skills from a programming perspective and hence, the questions asked are considerably different from those covered in tutorials or textbooks. For example, textbooks usually cover pedagogical questions and solutions that are meant for teaching how to write programs.

Most of the programming textbooks (C/C++/Java) explain how to create a singly linked list or how to write a program for adding or deleting nodes in a binary tree. However, in IT interviews, different kinds of questions are asked from the same topics. For example, "given a singly-linked list, find out the midpoint of a single-linked list in a single parse of the list" or "given a binary tree, how will you verify it is a binary search tree or not". It is difficult to answer such questions by reading or trying out programs given in textbooks or tutorials or by having only theoretical knowledge about technical topics. This book precisely covers such questions.

Target Audience

The book is designed specifically for students and programmers attending campus placements/interviews for software companies with the objective of helping them clear written tests and interviews.

The campus placements in IT companies in India typically consist of written tests and interviews. In written tests, the objective is to assess programming aptitude and technical ability of the candidates. Programming aptitude tests usually have multiple choice questions on C programming and sometimes on Java and C++. They are designed to check if a student knows the fundamental programming concepts and has basic programming and problem solving skills. Aptitude tests cover both theoretical and practical components to test the breadth of a student's knowledge. Interviews complement written tests and typically have questions on problem solving skills and programming puzzles to gauge the depth of understanding. As a student, by reading this book you can easily crack both the written tests and interviews in campus placements.

The job interviews for young programmers (with 1 to 4 years experience) are quite different from the written tests/interviews conducted for students.

Except a very few ones, good IT companies do not conduct written tests for programmers with prior work experience. The interviews are designed to assess a candidate's practical experience in programming and his/her ability to solve the problems faced in day-to-day programming. The programming logic questions and puzzles are also much more complex than those that students are asked. Hence, a programmer can benefit from this book while preparing for such job interviews.

Organization

The book is organized in a question-answer (some chapters in question-answer-explanation) format. Questions covered in this book are of varying levels of difficulty since the questions posed to experienced programmers are expected to be more difficult than the ones posed to students. For instance, each question is marked with a level number ranging from 1 to 5 (implying increasing level of difficulty). Level 1 to 3 questions are simpler ones meant for students and novice programmers; level 3 to 5 questions are meant for programmers with more than a year's experience.

The book starts with FAQs followed by five sections; and three appendices with additional material.

The introductory chapter is organized as FAQs (Frequently Asked Questions) on cracking an IT interview. It has two parts. First part covers general questions about written tests and interviews and clarifies the doubts that the readers might have. The second part has a list of common questions asked in HR interviews and has suggestions and hints on how to answer them.

The core of the book begins with a section on general programming topics. It has three chapters: general problem solving questions, data structures and algorithms and object oriented programming. Most of the interviews focus on these three topics; so this section is important to all the readers.

The next three sections cover C, C++, and Java respectively. Depending on the job interview you are attending, you can choose a section (on the specific language) to read. These three sections have a uniform chapter content. Each consists of three chapters: multiple choice, programming aptitude and theoretical questions. Multiple choice questions are meant to help you clear the written tests. Programming aptitude and theoretical questions help you to prepare for interviews (to some extent, they are useful for written tests also). Multiple choice questions have four options; each question has a unique answer. The chapter on programming aptitude questions has self-contained, compilable programs: after reading the program, you need to determine the output (or behavior) of the program. Answer and explanation are provided directly after the question; but it is recommended that you read the program and try answering instead of just reading the answer and explanation. Theoretical questions focus on important topics in a given language; you can increase or refresh your knowledge by reading the questions and answers in this chapter which will be very helpful in attending interviews.

The last section has two sample question papers. A student can evaluate herself by attempting these timed written tests. The first question paper

Contents

1

Cracking the IT Interview—FAQ

This introductory chapter is in the form of Frequently Asked Questions (FAQ). It answers many of the basic questions and doubts most students and young programmers have. This FAQ also clears a few misconceptions about attending interviews.

Part I: General Questions

1. What skills do IT companies look for in prospective candidates?

Answer: IT companies look out for various technical skills and soft skills in candidates. In soft skills, communication skills are very important. Other soft skills include presentation skills, team work, writing skills, etc. In technical skills, companies expect the candidates to have good expertise in their area of graduation. For computer science students, evergreen technical skills are C, Unix, operating systems, and networking. Candidates having these skills are likely to have better chances of getting a job.

2. When should I start preparing for placements?

Answer: For soft skills, it is better to start preparing one year before the placements start. For technical skills, it is beneficial to focus from the first year itself. Otherwise, you can start revising important subjects—from the placements point of view—from one year before placements.

3. What is the difference between a CV and a résumé?

Answer: A CV (Curriculum Vitae) is a document prepared by a student or a fresher searching a job. It provides the academic details of the student. A résumé is prepared by a person having work experience, which emphasizes job history and on-the-job skills and experience.

4. What should or should not be there in my CV?

Answer: A good CV will have the following sections: objective, personal strengths, academic background, academic achievements (if any), extracurricular activities, project details, areas of interest, and personal and contact details. It should have neat and simple formatting. Ideally, it should be of 2–3 pages.

Some characteristics of a bad CV can be as follows: overly complex formatting or styles; typos, grammatical mistakes; either too short (1 page), or too long (> 3 pages); too many personal details (e.g., parent's occupation, 3rd consolation prize in Rangoli competition); list of weaknesses, etc.

5. How detailed should the CV or résumé be?

Answer: The CV or résumé should ideally have adequate details about one's job history or academic details, project details, and achievements.

The job history should be from current work experience to the details of the first job (i.e., in reverse chronological order) giving the name of the company, role/position and number of years (preferably with joining and leaving dates). The academic details should cover the college/university, course, year of passing, percentages/CGPA, etc. Project details should include the title, where it was done, softwares used, and a short description of the project. Other sections can be very brief and to-the-point.

6. How should I prepare for written tests?

Answer: To prepare for written tests, following should be taken care of:

- Know the general format of the question papers from IT companies. Typically, most of the IT companies cover some or all of the following topics:
  1. programming aptitude (C, data structures, algorithms, etc.)
  2. quantitative aptitude
  3. analytical ability
  4. reasoning (logical, critical, etc.)
  5. verbal skills (synonyms, grammar, composition, etc.)
  6. puzzles
- Try solving previous years' sample question papers

  Attempting to solve previous years' sample question papers is very important. This helps you know where you stand, get experience in answering questions quickly in the actual written test, and, in general, become confident of clearing the written test. For sample question papers, please see Chapters 14 and 15.

- Read relevant books
  - GRE (Barron's guide)
  - R.S. Aggarwal's aptitude books (quantitative aptitude, reasoning, etc.)
  - Shakuntala Devi's puzzle books (*Puzzles to Puzzle You, More Puzzles to Puzzle You,* etc.)
  - Technical books (*Let us C,* etc.)

A few important and relevant job search sites, technical sites, and list of books are provided in Appendix I and II in this book.

7. How should I prepare for attending an interview?

Answer: At the first place, know the basic details about the company—its main business, size, etc. If possible, visit the company's website and get to know the general details about the company. This helps to show that you are interested in the company when related questions are asked in the interview.

An experienced person looking for a job change needs to know about the current position he is applying for (the job profile), what is expected from a person in that position, and how he can fit into that position.

An important aspect in cracking the interview is your attitude (i.e., how you present yourself). Show keen interest, be attentive and listen to the interviewer. Other aspects to look out for are eye contact, body language, appearance, way of speaking, showing respect, etc. Also, be in time for the interview (e.g., start early if traffic jams are common in your city).

Knowing the latest advances in technology and other happenings in your technical domain would be an added advantage.

8. How many interviews do I have to clear to get a job?

Answer: It is usually a minimum of two interviews: a technical interview and an HR interview. In some cases (e.g., if the company you've applied to is in another city or country), a telephonic interview is done to screen the candidates before calling them for face-to-face interviews. In major Indian IT companies and MNCs that are product-based, there will be multiple technical interviews. Unless the hiring company is satisfied with your technical skills (and communication skills) and you clear the HR interview, you cannot get the job. Your social status makes no impact in the interviews.

9. **Why do companies have an HR interview in addition to the technical interview(s)?**

   Answer: The HR interview has two objectives—(1) to check if you're fit for the organization; and (2) to check if your requirements match that of the organization. If you have good communication skills, a friendly personality, a positive attitude, and keen interest in learning and contributing, and in addition, if you're an effective team player and it is likely that you'll stay for at least a few years, it's most likely that you will be beneficial to the organization. An HR interview is intended to check all these aspects. In other words, HR interview checks your soft skills, attitude, and if you're the "right fit" for the company.

10. **Why is an HR interview important?**

    Answer: Simple—only if you clear this interview, you'll get a job! Also, if you get the job, the HR department (of course, after consulting with the manager of the team for which you're recruited) decides the pay you'll get, your roles, and responsibilities, etc.

11. **I am a class topper. Should I attend only selective companies in our campus interviews because I am sure to get a job?**

    Answer: You're overconfident and this can spoil your chances of getting a job. Being a class topper obviously gives you a better chance of getting a job. But remember, the skills required for getting a job are different from getting high marks. It is better to get a job first and then start being selective about your 'dream company', which you may want to join as a second job. Also, if you're selective, you lose opportunities to attempt written tests and interviews and hence when your 'dream company' comes for placement, you'll be ill-prepared for it.

12. **I got a job, but I do not get a call to join the company. Should I keep on waiting indefinitely?**

    Answer: Don't worry. If the job environment is bad, it is natural that your joining date can be very late (in some cases, delay can be more than a year!). Keep in touch with the HR of the concerned company and get to know the status from other candidates who have got placed in that company. Explore other alternatives: search for another job (who knows you might be destined for a better job!); join some job-oriented courses; do certifications in the areas of your interest ....

13. That brings me to another question. Do certifications help in getting a job?

Answer: Yes. Today certifications are an effective way to demonstrate your expertise in a particular technical domain. Getting relevant, valuable certifications such as CCNA, MCSD, SCJP etc. can significantly improve your chances of getting a well-paid job. A list of entry level certification courses is provided in Appendix III.

14. I have been searching for a job for more than a year and I still have not got a job. What are the options that I have?

Answer: Don't lose hope. There are examples of candidates who have good jobs, successful careers, who couldn't get any job initially. Continue to search for jobs, but also explore other options in this situation:

1. Go in for higher studies and improve your academic qualifications.
2. If you have a Bachelor's degree, consider enrolling for Master's degrees such as MBA, M.Tech, or other courses, depending on your interest.
3. Consider joining advanced courses, such as a postgraduate diploma program from C-DAC.
4. Do some certification courses (CCNA, etc.) in your areas of interest, which can improve your chances of getting a good job.
5. Join some evergreen job-oriented courses such as software testing and technical writing.
6. Depending on your interest, take some specialized courses such as advanced animation, or CAD-CAM, from established institutes like Aptech, NIIT, etc.
7. Learn any new computer skills: new programming languages C, C++, Java, operating systems (Unix, Linux), applications (Tally, etc.). Such job skills significantly increase your chances of getting a job.
8. Network with your college seniors, relatives, or friends who are already working and ask them to forward your resume to their HR departments.
9. Send your CV to companies both directly and through consultants.
10. Improve soft skills: communication skills, presentation skills, learning foreign languages (Japanese, French, German, etc.).

11. Do software projects: it adds value to your CV. Try doing a project from a reputed organization (MNCs, PSUs, Govt. Organizations, etc.). Don't pay for doing projects. Rather, try working as a trainee without getting paid or with a minimum stipend (to gain experience).

15. Is it necessary to change jobs frequently to get better pay and position?

Answer: No, it is not a good idea to change jobs frequently. Try to stick to a company and work there for at least 3 to 5 years.

There are good reasons why one would change a job for professional reasons (better pay, career advancement, new work environment, new area of work, overseas work assignments, etc.), or personal reasons (getting married, want to live with parents, etc.). It is perfectly acceptable to change a job for such reasons. However, don't change your job frequently. There are many reasons why we should avoid 'job hopping.'

Potential employers look at the job history of candidates before selecting them. If a person has changed jobs often (say, 5 jobs in 5 years!), it is very likely that the person will do so in future as well, so employers prefer not recruiting such candidates. Typically, it requires around 6 months to become productive in a new organization and start contributing. If you leave the job within a short period—say within a year—it is a loss to the company because of many reasons: the company has ramped you in your new job and that effort is lost, the company has to spend again to recruit a new person for your position, the work gets pending till the time the new person on board becomes productive, etc. So it becomes difficult to get a new job if you are a 'job hopper.'

It is better to take a long-term view about your career. It takes at least 5 years to learn enough about the job, the company, the technology, become highly productive, and make significant contributions to the company. The pay and position we get in the company depends on the level of contribution we make to the company. If you find 'your kind of job and company' and stick to it, and focus on contributing to the company, you'll naturally grow and earn better than if you keep shifting jobs. 'Focus on learning than earning'—that's what all the successful people have done!

Part II: Some FAQs to Candidates and How to Answer Them

16. Tell me about yourself.

Answer: This is an open-ended question that interviewers ask at the beginning of the interview to know more about you. They also use this question to get an idea of how you look at yourself and your achievements.

Briefly explain your professional background, the projects you've done, significant contributions you've made in your previous jobs, and conclude with a note about your personal background and a few points on your positive personal characteristics. Don't talk for an hour—make it short and to-the-point. Also, don't overemphasize your personal details.

17. What are your strengths and weaknesses?

Answer: This is a question asked to check how you look at yourself and also how your strengths can contribute to the team.

Be honest and tell what you consider as your strengths ('I learn new skills fast,' 'I am an effective team player, ' 'I have good leadership skills', etc.). Provide supporting details for your strengths ('I learn new skills fast. In my previous job, I had to learn scripting. I started to do shell programming from the next day itself, and I did it!'). For weaknesses, don't elaborate too much; some weaknesses can cost you your job ('I can't resist stealing if I see costly mobiles!').

18. What do you know about our company?

Answer: This question is to check if you would be interested in working in that company (if you're a kernel hacker, it is unlikely that you'll be interested in web programming, assuming that the company develops web-based software). It is also to check if you're keenly interested in joining the company—if you're going higher studies and attending interviews 'just-like-that,' then you would not have shown much interest in knowing more about that company, right!

To answer this question you should prepare before attending the interview. Visit the company's website to know about the company. If you know anyone working in the company, contact them and get an idea of what the company works on, which countries (or states) it has presence in, the kind of projects or products they are working on, etc. An overview of the company is more than enough.

19. What do you think of your previous boss?

Answer: This question is to check how well you can work with, or relate to your new boss if you get the job.

Speak about a few good things you found while working with your previous boss. However, you can't be too open in answering this question!

20. Why are you planning to leave your current job?

Answer: Be careful in answering this question. Usually acceptable answers are: 'looking for better pay'; 'looking for better role and growth opportunities'; 'got married and had to shift to this city. 'Bad answers: 'I didn't like my old boss!' (you're too frank to get this job!); 'the project is nearing the deadline and I don't want to work in that hectic schedule' (you can't desert your project when it is in critical situation!); 'I *had* to work!' (come on, you're paid for doing your work!).

21. Tell us about some challenges you faced in your previous job and how you overcame them.

Answer: This question is asked to check how confident you are in handling your day-to-day work and also your confidence in sticky situations.

You can briefly explain some of the challenges that you faced in your earlier jobs, how you dealt with them, how your team or managers helped you, and how you successfully overcame the problems finally. Avoid talking about bad experiences. Also avoid blaming anyone or the team for any problem. It is better to talk about technical challenges and problems.

22. Are you a team player?

Answer: Sometimes the interviewer asks you this question directly. This question is also asked indirectly, as in: 'do you prefer working in teams or working alone?' or 'how comfortable are you in working as a member in a large team?' This question is to check how good and comfortable you are in working as a team player (particularly in a few types of jobs where team work is very important).

Obviously we need to say yes, but support your answer with more details or by giving a few instances in the past where you worked very well as a team player. If you are a fresher, you can talk about your participation or organizing team sport events, get-togethers, etc.

Focus more about team strength than about individual abilities. This question could also lead to questions like how you handled conflicts within the team. So be prepared!

23. How much salary hike are you looking for?

Answer: Obviously this is one of the most difficult questions to answer!

If you're honest and say 'double the current salary,' ' you won't get the job. If you say, 'I am fine getting even the old salary, ' you might actually end up getting it! A safe answer is 'same as the industry average hike one gets while moving to a new job' (whatever that 'industry average' means!). If you've done enough analysis about the salary structure in the new company and know that you'll get more for same level of experience and skill set, you can say: 'same as the salary that a person with similar experience and skills will get in your company,' and throw the ball back in the interviewer's court.

24. Why should we hire you?

Answer: This is a question that every interviewer has, while interviewing a candidate. They want a justification on why they should select you. The interviewer just bounces this ball to you and checks how you give the reason for hiring you!

Tell them about your professional and personal strengths, relevant job experience, or academic background, your suitability for the current job requirements, etc., and give your view on why they should hire you. Bad answers: 'Because I am desperate for a job;' 'I have searched for jobs for more than a year and I didn't get any—you should help me!'

25. Do you have any questions for us?

Answer: Typically, an interviewer will ask this question just before the end of the interview. This is to check if you have any important questions that you want to get clarified. Instead of saying 'I have no questions,' it is better to ask relevant questions to show your keen interest in getting the job.

Do show enthusiasm about the new job and ask about the new team, opportunities, company, etc. Good examples: 'What are the current problems that the team is facing now and how can I possibly help?', 'What are the career growth opportunities available in the company'. Bad examples: 'Did I do the interview well?', 'Will I get this job?' (Both are in the list of Frequently Asked Wrong Questions—never ask these questions in the interview! But yes, you can ask 'When can I expect to hear from you?'.)

Section I
General Programming

2

C Problem-Solving Questions

One of the important aspects in technical interviews is testing the problem-solving aptitude of the candidate.

There are multiple ways in which problems can be asked for assessing candidates' problem-solving skills such as bit manipulation programming questions in C, finding what went wrong in data-structure manipulation programs, solving a particular programming problem, etc. We will cover these in this chapter, along with a few tricky questions asked in interviews.

The code segments or functions given in this chapter outline the solution and are not fully compilable or runnable programs.

1. **Write a simple implementation of C `strlen` function. (Level 1)**

   Answer: Copy the address of a passed string to a temporary variable and traverse the string till the end-of-string (a null character) is reached. Then return one less than the difference between the temporary variable and the address of the string passed. This will return the length of the string.

   ```
   int my_strlen(char *str){
     char *temp = str;
     while (*temp++)
       ;
     return (temp - str - 1);
   }
   ```

2. **How would you find if the given string is a palindrome or not? (A palindrome string reads the same if we read it backwards. For example, "malayalam" and "madam"). (Level 1)**

   Answer: We can use two pointers, one pointing to the beginning of the string and the other to the end. A loop is a setup that compares the characters pointed by these two pointers—if the characters don't match, it's not a palindrome. Else, move the pointers one character each toward the other. The loop continues till the pointers cross each other in the middle of the string. At that point we can conclude that the given string is a palindrome.

The code for this is as follows:

```
int is_palindrome(char * str) {
  char *p1 = str, *p2 = str + strlen(str) - 1;
  while (p1 < p2) {
  if (*p1++ != *p2——)
    return 0;  // false; chars not equal, so not a
    palindrome
  }
  return 1; // yes, it's a palindrome
}
```

3. **Which data structure can always be used for rewriting a recursive program to a non-recursive one? Why? (Level 1)**

   Answer: The required data structure is stack. Recursion internally uses stack, so stack can always be used for implementing a non-recursive equivalent for a recursive program.

4. **How can we print the contents of a singly linked list in a reverse order? (Level 1)**

   Answer: We can write a recursive function that traverses to the end of the list. While returning from the function, the contents of the node can be printed, which will be in reverse order now.

5. **Write simple functions/macros to set and clear (unset) a specific bit in a given variable. (Level 1)**

   Answer: The bit-wise `or` operation can set a bit and bit-wise `and` can reset a bit.

```
#define BIT(x)   (0x1 << x)

int set(int val, int pos) {
  return (val |= BIT(pos));
}

int unset(int val, int pos) {
  return (val &= ~BIT(pos));
}
```

6. **Rewrite the statement "int k = i * 8 + j / 4;" using bit-wise operators (where i and j are integers). (Level 2)**

   Answer: Multiplication and division by a number that is a power of 2 can be replaced by equivalent << and >> operators. So the expression can be rewritten as:

```
int k = (i << 3) + (j >> 2);
```

7. **What does the expression (j + ((i – j) & –(i < j))) do where i and j are integers? (Level 2)**

   Answer: This code results in the minimum of two integers, i and j. Note that, with slight modification, the expression (j – ((i – j) & –(i<j))) results in the maximum of two integers.

8. **Provide different solutions for finding the complement of an integer (instead of directly applying the complement of an integer). (Level 2)**

   Answer: This solution considers the fact that applying XOR operator with all ones will result in a complement and ~i is the same as one's complement + 1:

```
int complement(int i){
  printf("Solution 1::%x\n", i^(~0));
  printf("Solution 2::%x\n", -(i+1));
}
```

9. **There is a list of numbers from 1 to (N – 1). How would you find a repeated number (duplicate number) in that list? (Level 2)**

   Answer: Add all the values in the list and subtract the value (N * (N – 1))/2 from the sum. The resulting number is the repeated number.

   The formula to find the sum of numbers from 1 to N is (N * (N + 1))/2. Since there are (N – 1) numbers in the list, the formula becomes ((N – 1) (N – 1 + 1))/2, which is (N * (N – 1))/2. The list of numbers is from 1 to (N – 1), with one extra value. So, when we subtract the sum of numbers 1 to (N – 1) from the sum of numbers in that list, we get that extra value (i.e., the repeated number).

   Here is the function dup, which returns the duplicate element in the array of size N.

```
int dup(int arr[], int N) {
  int sum = 0;
  for(int i = 0; i < N; i++)
    sum += arr[i];
  int sigma_sum = (N * (N - 1))/2;
  return sum - sigma_sum;
}
```

10. **What is wrong with the following code segment? (Level 2)**

    Answer: `// free the nodes in a singly linked list`

```
for(ptr = head; ptr != NULL; ptr = ptr -> next) {
  free(ptr);
}
```

In the expression `ptr = ptr->next,` `ptr` is accessed after doing `free(ptr)`. So the behavior of this code segment is undefined. The solution is to introduce a temporary variable to hold the address to be pointed next and then free the ptr:

```
for( ptr = head ; ptr != NULL ; ptr = temp) {
  temp = ptr->next;
  free(ptr);
}
```

11. The following loop is intended to copy the data in a linked list to an array. What went wrong with this code? (Level 2)

Answer:
```
int i = 0;
for(ptr = head; ptr != NULL; ptr = ptr->next)
arr[++i] = p->data;
```

The operation `++i` leaves the first element `arr[0]` uninitialized. Moreover, since the first element is not assigned, and if the size of the array is the same as the number of elements in the linked list, the loop will attempt to write past the end of the array, which is wrong.

12. What is wrong with the following code segment? (Level 2)

Answer:
```
// code for inorder traversal of a binary tree
void inorder_traverse (Node* root) {
  if (root == 0) {
    printf("empty tree");
  }
  else {
    inorder_traverse (root->left);
    printf("%d\n, " root->data);
    inorder_traverse (root->right);
  }
}
```

The termination condition of recursion is wrong. If root is zero, the control has to return, but it instead just prints "empty tree." This will result in printing "empty tree" (N + 1) times because for a binary tree with N nodes, there are N + 1 empty nodes.

13. It is a common practice to assign null to the pointer after calling free. The following macro does it for convenient use:

```
#define free_null(p) free(p); p = NULL;
```

What can go wrong with this macro and how would you provide a better version of `free_null`? (Level 2)

Answer: Consider the following example of how `free_null` can be used:

```
if(foo(p))    // consider that foo is some function
free_null(p);

// this expands to:

if(foo(p))
  free(p);
  p = NULL; ;
```

Note that the assignment of p to null is outside if statement and spurious null statement (;). As it is evident, this macro can lead to problems when used as a statement. A better option is to make it possible to be used as a single statement.

```
#define free_null(p) do { free(p); p = NULL; } while(0)
```

This alternative implementation can now be used as a standalone statement or inside blocks as well. The spurious semicolon is also ignored.

14. You are given a `#define X` for some data type in C. How would you find the size of the data type, without using the `sizeof` operator, declaring a variable or a pointer variable of that type? (Level 2)

Answer: Since `sizeof`, or declaring variables or pointers of that type are not allowed, we are left with only pointer arithmetic to get the size of the data type. The only known valid value for any pointer to a data type is 0 (null). And adding 1 results in adding `sizeof`(X) bytes. Hence the solution is the following expression: `(((X*)0)+1)`

15. How would you swap two nibbles in a byte using a macro? (Level 2)

Answer: Using bit-wise and shift operations, separate the two nibbles and do `or` to interchange them by doing bit-wise `or` of the result:

```
#define swap(x) (((x & 0xF) << 4)  | ((x & 0xF0)>> 4))
// or
#define swap(x) (((x % 16) << 4)  | ((x >> 4) % 16)
```

16. How would you find the number of bits set in an unsigned integer? (You can use loops). (Level 2)

Answer: Solution I:

This straightforward solution loops over to check if a particular bit is set and increment the count:

```
int count_bits(unsigned int n){
  int c = 0;
  while (n != 0){
    if (n & 01)
      c++;
```

```
        n >>= 1;
    }
    return c;
}
```

Solution II:

This solution loops uses the same logic as that of finding the highest bit set, and loops over to count the number of times it becomes true:

```
int count_bits(unsigned int n){
    int c = 0;
    while (n) {
        n &= n - 1;
        c++;
    }
    return c;
}
```

Note: You can modify the code by adding the following statement to check if the integer is that power of 2 or not:

```
    (ones==1) ?
        printf("The number is a power of 2")
        : printf("The number is NOT a power of 2");
```

17. The parity of an integer refers to even or odd number of bits set in that integer (even-parity or odd-parity). How would you find the parity of an unsigned integer? (Level 2)

Answer: Solution I:

The direct way to find if the integer is of even or odd parity is to count the number of bits set in an integer and check if it is an even or odd number.

Solution II:

In the logic to find the number of bits set in an integer, introduce a parity Boolean value (which can be an integer). If the loop executes even number of times, `par` will be 0, and odd number of times, it will be set to 1. The final result of `par` is the parity of the integer:

```
typedef int bool;

int parity(unsigned int n){
    bool par = 0; // set initial parity to false
    // 0 (false) is even parity and 1 (true) is odd parity
    while (n) {
        par = !par;
        n &= n - 1;
    }
    return par;
}
```

18. How can you swap two variables without using a temporary variable? (Level 2)

Answer: There are a few well-known ways to swap two variables without using a temporary variable. We'll see a solution using arithmetic and bit-wise operators.

Assuming that i and j are integers, here is a trick to do it with + and – operators:

```
i = i + j;
j = i - j;
i = i - j;
```

Here, in the first step, the values of both i and j are added and stored in one variable. In the next step, the added value of the variable i is subtracted from j, which results in value of i (since i + j – j is i) and that value gets stored in j. The value of i is still the original value of i + j and j now contains the original value of i. Now the expression i – j is equivalent to i + j – i, which is j; and that value gets stored in i. So, i and j have their values swapped in the last step.

If i and j are not big (so that the result does not overflow), and if j is not zero, then the following trick also works, based on the same logic outlined earlier:

```
i = i * j;
j = i / j;
i = i / j;
```

This trick worked using arithmetic operators (+, –, *, /) and can be done using single bit-wise ^ (`ex-or`) operator also:

```
i = i ^ j;
j = i ^ j;
i = i ^ j
```

This works based on the idea that the `ex-or` operator complements itself.

19. Given the pointer to a particular node, say `node_ptr`, in a singly linked list, how would you delete that node? (Level 3)

Answer: Since we don't know the previous node pointer in a singly linked list, we can't actually delete the node and keep the linked list intact. One possible solution is to copy the rest of the nodes while leaving the current node:

```
// save the node in a temporary pointer
node * temp_node_ptr = node_ptr;
// till the end of the list is reached
while (temp_node_ptr != 0) {
  // copy the next node to the previous ones
  //- the first one is discarded
  temp_node_ptr.data = temp_node_ptr->next.data;

  // traverse to the next node to continue copying
  temp_node_ptr = temp_node_ptr->next;
}
```

As a result, the list seems as if the link has moved one place ahead, with the data in the node initially pointed by `node_ptr` getting deleted.

One problem with this solution is that we need to copy the data in the nodes till the end of the list (the time complexity is O(N)). And if there are any pointers pointing to somewhere in the rest of the list, then they will get invalidated. Therefore, a better solution is to copy only the data from the immediate next node and free the next node:

```
// save the node in a temporary pointer
node * temp_node_ptr = node_ptr;

if (temp_node_ptr->next) {
  temp_node_ptr.data = temp_node_ptr->next.data;

  // save the address of next node to free it later
  node *free_ptr = temp_node_ptr->next;

  // copy the next of the next node to the current node
  temp_node_ptr->next = temp_node_ptr->next->next;

  // we've copied both data and next, so free it now
  free(free_ptr);
}
```

Note that these solutions won't work if the node pointer is the last element in the list (in which case, we have no way of setting the previous node pointer to null).

20. How would you find the middle node in a singly linked list? Write a sample C code segment for this. (Level 3)

Answer: Create two pointers, pointing to the starting of the list—one pointer to traverse node-by-node and another to traverse by skipping a node. When the second pointer reaches null, the node pointed by the first pointer is the middle node in the linked list.

```
Node * middle_node (Node *head) {
  Node * ptr1 = head, * ptr2 = head;
  assert(head);
  while (ptr2) {
    ptr1 = ptr1->next;
    ptr2 = (ptr2->next) ? ptr2->next->next : 0;
  }
```

```
    return ptr1;
}
```

21. Consider that the doubly linked list has 5 nodes with data 10, 40, 23, 44, and 50 in the nodes. The recursive version of traversal function to visit the nodes (and print the values in the nodes) is given below:

```
void traverse(Node* node) {
    static int i = 1;
    if(node == NULL)
        return;
    traverse(node->next);
    printf("%d : %d\t", i++, node->data);
}
```

I. What will be the output of this code when the head node is passed to this traverse function?

II. Write an iterative version of this code. (Level 3)

I. Output:

```
1 : 50 2 : 44 3 : 23 4 : 40 5 : 10
```

II. Iteractive version:

```
void traverse(Node* node) {
    static int i = 1;
    while (node->next)
        node = node->next;
    while (node) {
        printf("%d : %d\t", i++, node->data);
        node = node->prev;
    }
}
```

Answer: In the recursive function, the data are printed in the reverse order of the elements in the linked list. In the iterative version, the traversal is done till the end of the list. Then traversal is done back till the beginning, while the data are printed.

22. Given a binary tree, you need to verify if it is a binary search tree or not. How do you do that? (Level 3)

Answer: A binary search tree is a binary tree for which the following condition holds true: left-node val <= mid-node val <= right-node val. So, in the inorder traversal of the binary tree, it is enough to ensure that this condition is true:

```
// code to do inorder traversal of a binary tree and
// assert that it's a binary search tree
```

```
void inorder_BST (Node* node) {
  if(node == 0) {
    return;
  }
  else {
    if(node->left)
      assert(node->left->data <= node->data);
    inorder_traverse (root->left);
    printf("%d\n", root->data);
    if(node->right)
      assert(node->right->data > node->data);
    }
}
```

23. How would you find if your system follows a big-endian or a little-endian order? (Level 3)

Answer: The byte-order in which the variable is stored in a machine is referred to as the endianness of the machine. If the bytes are stored in sequence order it is known as a big-endian scheme, and if the order of bytes is reversed, it is known as little-endian scheme. For example, for the hexa-decimal value 0x0102, if the machine stores the two bytes in increasing order in memory, as 0x01 and 0x02, it is a big-endian machine. if the order is 0x02 and 0x01, then it's a little endian machine—Solution I does this check.

Solution I:

```
int main(){
  union {
    short c;
    char a[sizeof(short)];
  } check;
  check.c=0x0102;
  if ( check.a[0] == 0x01 && check.a[sizeof(short) - 1]
  == 0x02 )
    printf("Big endian\n");
  else
    printf("Little endian\n");
}
```

Solution II:

In this solution, there is an integer with initial value 10 (which can be stored in a byte). In other words, the value is 0x0000000A. If the first byte is nonzero (i.e., 0x10, it is little-endian, otherwise it's big-endian). By casting the pointer to `char *`, and indirecting it results in the value of the first byte in the integer:

```
int main(){
  int x=10;
  if (*((char *)&x) == 0)
    printf("Big Endian\n");
  else
    printf("Little Endian\n");
}
```

24. The macro `offsetof` is used to find the offset of a member inside a structure.

A sample implementation of this macro is as:

```
#define offsetof(a,b) ((int)(&(((a*)(0))->b)))
```

Explain how this macro works. (Level 3)

Answer: The only known valid value for any pointer to a data type is 0 (null), so the expression `(a *)0` gets a pointer of type `a*`. Now, accessing a member—which is `b` here—and taking its address will result in the offset N bytes from 0 for that member `b` in `a`. This is the relative offset of member `b` from the beginning of the object of type `a`.

25. How would you find if a number is the power of 2 or not by using only operators (no loops)? (Level 3)

Answer: The following solution is simple, so it can be written as a macro as well. If the bit-wise AND of an integer value and that value minus one is zero, then it's a power of 2:

```
int is_power_of_2(int n) {
  return(!(n & (n - 1)));
}
```

26. How would you find the bit-count of an integer without using any loops? (Level 3)

Answer: Solution I:

```
unsigned char init_array[256];
// initialize values from 1 to 256 for this array

int number = 10; // the number here
char *byte_pos = (char *)(&number);
int num = (init_array[byte_pos[0]] + init_array[byte_pos[1]]
        + init_array[byte_pos[2]]
        + init_array[byte_pos[3]]);
```

The idea is to keep an array (here it is init_array) with numbers initialized from 1 to 256 and pass on the bytes in an integer to given an index value. The sum of the result from all the bytes in an integer will give the number of bits set.

Solution II:

This solution is assuming that an integer is of 32-bit size (which can be modified for a given size):

```
int count_bits(unsigned int num) {
  unsigned int c = num;
  c = ((c >> 1) & 0x55555555) + (c & 0x55555555);
  c = ((c >> 2) & 0x33333333) + (c & 0x33333333);
  c = ((c >> 4) & 0x0F0F0F0F) + (c & 0x0F0F0F0F);
  c = ((c >> 8) & 0x00FF00FF) + (c & 0x00FF00FF);
  c = ((c >> 16) & 0x0000FFFF) + (c & 0x0000FFFF);
  return c;
}
```

This logic adds up the bits step-by-step and finally it results in the bit-count of the unsigned integer. The masks are 0101, 0011, 00001111, 00000000 11111111 (repeating) and 0000000000000000 1111111111111111, which is hexadecimal numbers are 5, 3, 0F, and 00FF and 0000FFFF (respectively).

27. Write a recursive function that prints the binary representation of an integer. (Level 3)

Answer: A solution is given here, but it does not print the leading zeros in the integer:

```
void bit(int num){
  if(num==0)
    return;
  bit(num>>1);
  (num&01)? printf("1") : printf("0");
}
```

28. Write a one-line program that will print the binary representation of the integer passed as argument. (Level 3)

Answer: This solution converts the passed string to integer using `atoi` and does looping to print the bits (as zero or one):

```
int main(int argc, char *argv[]){
  for(argc = sizeof(int)*8; argc--; putchar(
    (atoi(argv[1])&(1 << argc)) ? '1' : '0'));
  }
```

29. How would you find the size of an integer without using the sizeof operator? (Level 3)

Answer: Solution I:

This solution finds the number of bits in an integer and divides it by 8 (number of bits in a byte):

```
int size_of_int(){
  unsigned int x=-1;
  // or the value -1, all the bits are one
  int count = 1;
  // count init value is 1 not 0 because of the number of
  // iterations in while loop is (num_of_bits - 1)
  while (x<<=1)
    count++;
  return (count/8);
}
```

Solution II:

This solution finds the difference in addresses of two consecutive elements in an integer array (which is the size of an integer):

```
int size_of_int(){
  int arr[2];
  return (int) ((char *)(arr + 1) - (char *)arr);
}
```

Solution III:

This solution does increment of the value of an integer pointer whose value is 0 (which will result in size of integer bytes moved from 0):

```
int size_of_int(){
  int *p = 0;
  return ++p;
}
```

30. Write a simple code segment that checks if your implementation follows arithmetic or logical right shift (for negative integers). (Level 4)

Answer: In signed numbers, with right shift (>>), the value of the most significant bit (MSB) filled is implementation-dependent: it can be either arithmetic or logical shift.

Arithmetic Shift: The MSB is filled with the copy of sign bit, preserving the sign of the value.

Logical Shift: The MSB is filled with zero, thus modifying the sign of the value.

In other words, if the right-shifted value of a negative integer is less than zero, it is arithmetic shift, or else it is logical shift. Based on this logic, here is a code segment that prints if the implementation follows arithmetic or logical shift:

```
int main() {
  (((-1 >> 1) < 0)) ?
    puts("arithmetic shift\n") :
    puts("logical shift\n");
}
```

31. Consider that there is a loop in a singly linked list (by possibly a bug in the code). Write a sample C code segment that finds out if there is a loop in a given list or not. (Level 4)

Answer: Create two pointers, pointing to the starting of the list. One pointer traverses node-by-node and another should traverse one node at a time. If the pointers reach null, then there is no loop. At any point if the two pointers are equal (except for the initial condition when both point to the head node), there is a loop.

```
int is_there_a_loop(Node *head) {
  Node * ptr1 = head, * ptr2 = head;
  assert(head);
  if(head->next != 0) {
    ptr1 = head->next;
    ptr2 = (head->next) ? head->next->next : 0;
  }
  while ((ptr1 != 0) && (ptr2 != 0)) {
    if (ptr1 == ptr2) {
      return 1; // has loop
    }
    ptr1 = ptr1->next;
    ptr2 = (ptr2->next) ? ptr2->next->next : 0;
  }
  return 0; // no loop in the linked list
}
```

32. What does the expression ((x) & –(x)) do (where x is an integer)? (Level 4)

Answer: It returns the lowest bit-set in an integer (which is a power of 2).

33. Write a recursive function to reverse a string. (Level 4)

Answer: Here is a solution that uses the standard `strcpy` function for copying characters from the string:

```
char *my_strrev(char *s){
  return (!*s) ?
  s : strcpy(s, strncat(my_strrev(s+1),s,1));
}
```

34. How do you reverse the words in a string? For example, "This sentence needs to be reverted" should be converted to "reverted be to needs sentence This." An efficient and in space solution is preferable. (Level 5)

Answer: Reverse the string in-place by swapping nth letter with (len – n – 1)th letter from n = 0 to n/2. Then reverse each word in the string in same way.

```
// this function reverses the string starting from
     pointer p1
// to the end position pointed by p2.
void str_rev(char* p1, char *p2) {
  char *p = p1;
  while (p1 < p2) {
    char temp = *p1;
    *p1 = *p2;
    *p2 = temp;
    p1++;
    p2--;
  }
}

// this function reverses the words in the given string
// (whose start and end are indicated by p1 and p2)
void str_word_rev(char *p1, char *p2) {
  //step1: reverse all the characters in the string
  //first ...
  str_rev(p1, p2);

  // step2: now reverse every word in the string ...
  char *p = p1;
  p2 = p1;
  while ( *p != '\0' ) {
    while ( (*p2 != '\0') && (*p2 != ' ') ) {
      p2++;
  }
  str_rev(p1, p2 - 1);
  p2++;
  p1 = p2;
  p = p2;
  }
}
```

35. A doubly linked list has two pointers—`next` and `prev`—for traversing in both directions. Is it possible to convert that doubly linked list to use only one pointer—say `next`—and still provide the ability to traverse in other directions? (Level 5)

Answer: Yes, it is possible, and here is the description of one way to achieve.

The idea is to represent the pointers in an integer of equivalent size. Then use `ex-or` operation (^) on both `next` and `prev` pointers and store the result in the `next` pointer. Using the `head` pointer, the first node can be accessed directly; using (`head ^ next`) gives the `prev` pointer. Applying this resulting `prev` on `next` gives the actual next pointer, i.e. (`prev ^ next`). In this way, we can implement the doubly linked list that will occupy the same space as the equivalent singly linked list.

3

Data Structures and Algorithms Questions

In this chapter, we'll discuss some important questions in data structures and algorithms. However, problem-solving questions are not covered in this chapter. They are covered in a separate chapter dedicated to "Problem Solving". (see Chapter 2)

1. **A linear list of elements in which deletion can be done only from front-end and insertion from rear-end is known as: (Level 1)**

   (a) queue

   (b) stack

   (c) B+ tree

   (d) deque

   Answer: (a) queue.

2. **A linear list of elements in which insertions and deletions can take place at both ends (front and rear end) but not in the middle is known as: (Level 1)**

   (a) queue

   (b) stack

   (c) B+ tree

   (d) deque

   Answer: (d) deque.

3. **Which of the following data structures is suitable for modeling telephone connections in a network? (Level 2)**

   (a) circular linked lists

   (b) graphs

(c) m-ary trees

(d) sets

Answer: (b) graphs.

4. Which of the following data structures is suitable for implementing a file directory structure in an operating system? (Level 2)

(a) multiple levels of stacks

(b) multiple levels of linked lists

(c) binary tree

(d) m-ary tree

Answer: (d) m-ary tree.

5. Which of the following data structure is used for implementing recursion? (Level 2)

(a) queue

(b) stack

(c) B+ tree

(d) deque

Answer: (b) stack.

6. Which of the following data structure is useful for converting an expression in infix notation to postfix notation? (Level 2)

(a) queue

(b) stack

(c) B+ tree

(d) deque

Answer: (b) stack.

7. Which of the following sorting procedure is slowest? (Level 2)

(a) shell sort

(b) heap sort

(c) bubble sort

(d) merge sort

Answer: (c) bubble sort.

8. In the average case, which of the following sorting algorithms is fastest? (Level 2)

(a) quick sort

(b) selection sort

(c) bubble sort

(d) insertion sort

Answer: (a) quick sort. Quick sort performs fastest in the average case.

9. If sorting is done by comparison, what is the *minimum* worst case time complexity of a sorting algorithm? (Level 2)

(a) O(N)

(b) O(log N)

(c) O(N log N)

(d) O(N^2)

Answer: (c) O(N log N). If the sorting is done by comparing the values, then it has been proved that any sorting algorithm will require minimum O(N log N) time to do the sorting.

10. What is the maximum number of comparisons done in selection sort algorithm? (Level 1)

(a) O(N)

(b) O(log N)

(c) O(N log N)

(d) O(N^2)

Answer: (a) O(N).

Note that it is not O(N^2)—that is the worst case time complexity of selection sort. O(N) is maximum the number of comparisons required in this algorithm for completing the sort.

11. What are "static" and "dynamic" data structures? (Level 1)

Answer: Static data structures are of fixed size. For example, once we create an array in C language by specifying its size, we cannot modify it (however, we can change the data that the data structure stores). Dynamic data structures grow and shrink in memory size as required (as data are inserted or removed from the data structure). For example, a linked list is a dynamic structure.

12. What do you mean by "worst cast time complexity"? (Level 2)

Answer: 'Worst case time complexity' is a time complexity measure in which the worst-case scenario for time taken to do the steps taken in the algorithm is considered. The measure considers the maximum number of computation steps required on any input size N. It considers the closest approximation of the maximum time taken. It is given by the 'big-O' notation. For example, O(1) indicates the algorithm is of 'constant time complexity.'

13. What are the different kinds of search algorithms available? (Level 1)

Answer: Search algorithms can be classified based on the approach used in searching values. For example, sequential, binary and hashing. If the algorithm does the search sequentially by traversing the elements in the data structure, it is a sequential. If it does it by traversing by recursively dividing the search space into two (as in binary search), it is binary. If hashing is done to find the element in the underlying data structure (say a hash table), then it is hashing.

14. List basic "worst case time complexity" functions in the increasing order of complexity. (Level 2)

Answer: Constant time—O(1)

Logarithmic time—O(log N)

Linear time—O(N)

Quasilinear—O(N log N)

Quadratic time—O(N^2)

Cubic time—O(N^3)

Exponential time—O(2^N).

15. Describe "bubble sort" algorithm. (Level 2)

Answer: "Bubble sort" algorithm starts from one end of the list that is to be sorted.

It keeps comparing the adjacent elements starting from the first two to last two elements. For one traversal through the list, the last element contains the largest (or smallest, depending on the comparison function) element in that list. The second step is repeated in the list leaving the last element. This process is repeated till there are no more elements to compare (or a pass completes in which there was no swapping done—which means that the list is already in sorted order).

Bubble sort has worst-case complexity of O(N^2). Following is the code that sorts the array in ascending order:

```
void bubble_sort(int arr[], int len) {
  for (int i = 0; i < (len - 1); i++) {
    for (int j = 1; j < len; j++) {
      if (arr[j] < arr[j-1]) {
        int temp = arr[j];
        arr[j] = arr[j-1];
        arr[j-1] = temp;
      }
    }
  }
}
```

16. What is an ADT? Give an example for an ADT. (Level 2)

Answer: Abstract Data Type (ADT) refers to a structure with a set of operations defined on it. The internal implementation details of the structure are typically irrelevant and the ADT is described by the functionality (operations) it provides.

For example, `stack` is an ADT. A stack has well-known operations such as `push` and `pop`. These operations also have restriction that they can be done at only one end of the data structure. We can use a stack by just knowing its "interface" (i.e., set of functions it provides) without knowing its "implementation" details (i.e., is it implemented using an array, linked list, etc).

17. What is a queue? Mention some of the areas in computer science where queues are useful? (Level 2)

Answer: A queue data structure supports insertions (enqueue operations) from one end, and deletion operations (dequeue operations) from the other end. Because of this nature, the first elements that are inserted in the queue are the first ones to get deleted also, which is why queues are FIFO (First-In-First-Out) structures.

Queues are used in a wide range of applications in computer science. Notably, they are useful in operating systems and networks for storing list of items waiting for a resource.

18. What are the three common notations in which expressions can be represented? What are the advantages of each of these notations? (Level 2)

Answer: Based on how the operators and its operands are arranged, there are three common notations in which expressions can be represented: prefix, infix, and postfix notations.

Here is a simple example. The expression (A + B) * C is in infix form. Its prefix form is * +ABC. Its postfix form is AB + C *.

The infix notation is the natural form, easy to understand, and the most commonly used form in mathematics. In prefix and postfix forms, no explicit parenthesis is needed to force the precedence of operators. Also, it is easy to automate the evaluation of expressions in these two forms. Due to this reason, infix expressions are converted either to prefix or postfix before evaluating them and these two forms are used widely in computers. Being specific, using stack data structure, we can use the postfix form directly to evaluate the expressions.

19. What are the three ways to traverse a binary tree? (Level 2)

Answer: In-order, pre-order, and post-order are three ways to traverse a binary tree. Consider that L is left node, M is middle node, and R is right node. The traversal order in in-order is L-M-R; pre-order is M-L-R; and post-order is L-R-M.

20. What are two main ways to traverse a graph? (Level 2)

Answer: Breadth-First-Search (BFS) and Depth-First-Search (DFS) are the two ways to traverse a graph.

21. What is a "hash" table (or a "hash" map)? (Level 2)

Answer: "Hash" table (or "hash" map) is an associative container (that maps key to value) data structure. It supports efficient lookup for a given key and returns the corresponding value. For example, a hash table can be used for storing details of names and intercom phone extension numbers in a company. Given a person's name—which is a key—the extension number—the value for that key—for that person can be efficiently retrieved. With a well-chosen hashing function, both insertion and searching can be of O(1) time complexity.

22. What is a binary tree? (Level 2)

Answer: A binary tree is a data structure in which every node has at most two child nodes (i.e., a node can have zero, one or two child nodes). Child nodes are referred to as left and right nodes. Each node can have data in it.

23. What is a binary search tree? (Level 2)

Answer: A binary search tree is a binary tree in which elements are stored in a sorted order. The left subtree of a node will have values less than the value in the node. Similarly, the right subtree of a node will have values greater than the value in the node.

24. Which of the following best describes "quick sort" algorithm? (Level 3)

(a) greedy algorithm

(b) backtracking algorithm

(c) dynamic programming algorithm

(d) divide-and-conquer algorithm

Answer: (d) divide-and-conquer algorithm.

25. What are 'divide-and-conquer' algorithms? Give an example of a sorting algorithm, which is based on this 'divide-and-conquer' approach. (Level 3)

Answer: 'Divide-and-conquer' is an approach in problem solving with three steps: divide, solve (conquer), and combine the results. The input is divided into a set of pieces and the problem is solved on those small pieces (using the divide-and-conquer approach repeatedly), and then the results are combined together to get the final output.

'Divide-and-conquer' algorithms are typically *recursive* since the divided small pieces are solved by repeatedly applying the 'divide-and-conquer' approach.

Merge sort is an example of 'divide-and-conquer' algorithms. Say, we're given an array of 10 integers, and the objective is to sort the array. In merge sort, we divide the array into two parts—(1 to 5) and (5 to 10)—and merge sort is repeatedly applied on these two subparts. The resulting sub-arrays are combined together and finally the two sub-arrays are combined together to get the sorted array. The function `merge_sort` that shows how it works is discussed further:

```
// merges the array from arr[p to q] and arr[q to r]
void merge(int arr[], int p, int q, int r) {
  int temp[r];  // create a temporary array to merge the
  elements
  int i = p, k = p;
  int j = q;
  // loop and copy the elements from two arrays to
  // the temp array in sorted order
  while( (i <= q) && (j <= r) ) {
    if(arr[i] <= arr[j])
      temp[k++] = arr[i++];
    else
      temp[k++] = arr[j++];
  }
```

```
    while(i <= q) // copy left-over elements from p to q
      temp[k++] = arr[i++];
    while(j <= r) // copy left-over elements from q to r
      temp[k++] = arr[j++];
    for(int m = 0; m < r; m++)  // copy temp to original
    array
      arr[m] = temp[m];
}

// given array arr, it sorts the elements from p to r
void merge_sort(int arr[], int p, int r) {
  if(p < r) {
    int q = (p + r)/2;
    merge_sort(arr, p, q);
    merge_sort(arr, q, r);
    merge(arr, p, q, r);
  }
}
```

26. Consider these three data structures: `stack`, `queue` and `deque`. Which is a general data structure with which other two data structures can be implemented? (Level 3)

Answer: Deque is a general data structure because it supports insertions and deletions from either end. Given a deque implementation, it is possible to create a stack or queue data structure from that, which are more specialized versions.

27. What is a complete binary tree? (Level 3)

Answer: A complete binary tree is a binary tree in which every level of the tree is completely filled except for the last (bottom) level. In the last level, the leaves are filled from the left to right.

28. What is an AVL tree? (Level 3)

Answer: An AVL tree (named after its inventors Adelson-Velskii and Landis) is a height-balanced tree. For every node in the tree, the heights of left and right subtrees differ by at most 1 level, so it is a balanced tree.

29. What are "ordered" and "unordered" data structures? (Level 3)

Ordered data-structures keep the data in a specific order (such as sorted order) so that the data can be retrieved fast. An operation (for example, comparison operator) is used to find the ordering relationship between the objects stored in the data structure. Example for ordered data structure is a binary search tree where the elements are stored in an ordered manner (left node is smaller than the middle node and right node is bigger than the middle node).

Unordered data structures do not care for any ordering relationship between the elements that are stored in the data structure. For example, a hash table is an unordered data structure—it can store the data anywhere in the hash table and the exact location depends on the hash value (and the hash function used).

30. There are 10 nodes in a binary tree. How many nodes point to null in this tree? (in other words, how many null branches are there in the tree)? (Level 3)

Answer: The binary tree has 11 null branches.

In general, if there are N nodes in a binary tree, there are (N + 1) null nodes in the tree.

31. How many nonleaf nodes are there in a binary tree of maximum of N leaf nodes? (Level 3)

Answer: In a binary tree, when there are a maximum number of leaf nodes (N), it has (N – 1) nonleaf nodes.

32. What is the maximum possible height of a binary tree with N nodes? (Level 3)

Answer: A binary tree that is either totally skewed to left or right can have height N.

33. What is the height of a complete binary tree with N nodes? (Level 3)

Answer: A complete binary tree with N nodes has the height (log N).

34. What is the difference between simple graph and multigraph? (Level 3)

Answer: A simple graph can just have a one edge between any two vertices. A multigraph can have more than one edge connecting two vertices.

35. What is the difference between "undirected" graphs (graphs) and "directed" graphs (digraphs)? (Level 3)

Answer: A graph is a collection of nodes (or vertices) and lines (or edges) that connect these nodes. A graph G is mathematically given by the notation G = (V, E), where V is a set of vertices and E is a set of edges).

An undirected graph (graph) does not have any direction—in other words, the edges (u, v) and (v, u) are same. Also, self-loops are not allowed. A directed graph allows direction in the edges, in other words, the edges (u, v) and (v, u) are different. Also, self-loops are allowed. Example of a graph and a digraph is given in Fig. 3.1.

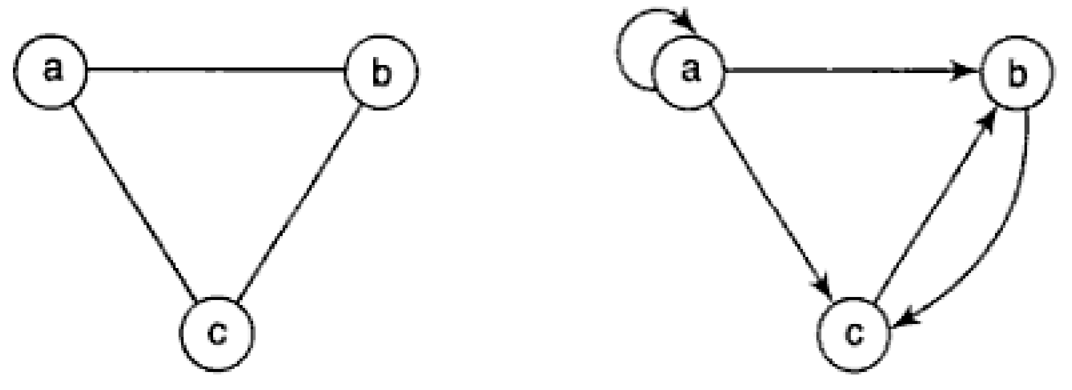

Figure 3.1 Example of an undirected graph and a directed graph

36. What are the two simple ways of representing graphs? (Level 3)

Answer: Adjacency matrix and associative lists are two simple ways to represent graphs.

37. What is a "priority" queue? Give some examples for algorithms where this data structure used. (Level 3)

Answer: A priority queue is a queue that has a 'priority' or 'rank' associated with the items that are inserted. The insertion and removal of the items from this data structure is done based on the priority of the values it stores (instead of the arrival order/time).

Priority queues are used in various algorithms such as heap sort, greedy algorithm, etc.

38. What is the relationship between trees and graphs? Are all trees graphs (or is it that all graphs are trees)? (Level 3)

Answer: A tree is a connected graph with no circuits (a circuit is the one where the starting point equals the end point). If we remove an edge from a tree, one or more nodes will become disconnected, so a tree is a 'minimally connected' graph. All trees are graphs (but not vice-versa).

4

Object-oriented Programming Questions

In this chapter, we'll cover theory questions in object-oriented programming. Note that we will not discuss object-oriented analysis and design or related topics here.

1. What is object-oriented programming? (Level 1)

Answer: Object-oriented programming is a programming paradigm in which programming is done using ADTs (Abstract Data Types). This programming style makes use of abstraction, polymorphism, inheritance, and runtime binding (virtual functions).

2. What is the significance of 'class' keyword in C++/Java? (Level 1)

Answer: The keyword 'class' (in C++ and Java) specifies an Abstract Data Type (ADT). ADTs expose operations that provide a higher level functionality, and the lower-level implementation details are isolated and hidden from the users of the class.

For example, consider a `circle` class. It provides the methods such as `draw`, `move`, `expand`, `erase`, and so on. To use a `circle` object, we only need to use these method names and *what* functionality these methods provide. It is not necessary for us to know *how* these methods are written. In particular, it is not necessary to know how the `circle` is actually implemented to use it—its underlying data structure can be anything. Even if the internal representation of the class is changed, the programs that *use* the class won't change unless the interface for those methods is changed. This nature of abstraction is the essential feature of ADTs.

3. What are "objects"? (Level 1)

Answer: "Objects" are created from classes (they are 'instances' of classes). An object has three essential aspects associated with it: *state,* *identity,* and *behavior.* Every object occupies some memory *space* (also known as 'state'). Every object has an address (*identity*) in the memory where the object is allocated. Every object has a set of methods that can be called its *behavior.*

4. What are "methods" and "fields"? (Level 1)

Answer: A class can have members. "Methods" and "fields" are two important members of classes. Member functions are known as methods and data members are known as fields.

5. What are the four important foundation concepts of OOP? (Level 1)

Answer: Object orientation is built on the strong foundations of encapsulation, abstraction, inheritance, and polymorphism.

6. What is "encapsulation"? (Level 1)

Answer: Encapsulation means combining data and related functions that use that data together and providing it as a logical entity. For example, a `Stack` class that stores integers encapsulates an array (field) of integers and methods such as `push`, `pop`, etc., which operate on that array.

7. What is "abstraction"? (Level 1)

Answer: "Abstraction" means hiding internal implementation details. For example, a `Stack` class can use an array internally for storing the data. That array can be a fixed-size array, dynamically allocated array, or some other data structure other than an array. A `Stack` class abstracts how it implements the functionality.

8. What is "polymorphism"? (Level 1)

Answer: In the real world, every message we pass has a context. Depending on the context, the entire meaning of the message could change and so also the response to the message. Similarly, in programming, depending on the object, a message can be interpreted in many ways. This capability of providing different interpretations based on the context is known as "polymorphism" in OOP.

9. What are "access specifiers"? (Level 1)

Answer: "Access specifiers" determine the accessibility of a class member. In general, there are three important access specifiers: `public`, `private`, and `protected`.

Public members are designed and supposed to be used by the users of the class. It is through the public members with which the class is accessed and used by the users of the class. These members are collectively referred to as the ***public interface*** of the class. Private members are not meant to be exposed to users and are just implementation detail of class that users of the class are not concerned about. Only the other class members can access the private member. In addition to the class members, derived classes can also access protected members. If there are no derived classes, protected members are equivalent to private members.

10. What is 'function overloading'? (Level 1)

Answer: In "function overloading", more than one method with the same name but different type of parameters and/or number of parameters can be defined. Depending on the actual number and/or static type of the parameters used, the compiler will resolve the call to the correct method.

11. What is 'operator overloading'? (Level 1)

Answer: In 'operator overloading', the predefined operators defined by the language can be supported by the classes also. With operator overloading, classes can behave like primitive types (like `int`, `float`, etc.) defined by the language. Depending on the arguments used for the operator, correct method call will be resolved by the compiler at compile-time.

12. What is the difference between the terms 'overloading' and 'overriding'? (Level 2)

Answer: The term 'overloading'—as in 'function overloading' and 'operator overloading'—refers to compile-time polymorphism. The term 'overriding' refers to providing an alternative function definition of a virtual function in a derived class. Overriding is useful for runtime polymorphism. With overloading, more than one method definition with the same name (but with different types/number of arguments) are provided, whereas in overriding, the methods with the same name are provided with alternative definition in derived class(es).

13. How do you broadly classify 'polymorphism'? (Level 2)

Answer: 'Polymorphism' can be broadly classified as compile-time and runtime polymorphism. Function overloading, operator overloading, and parametric types (templates in C++ or generics in Java) are done at compile-time. Dynamic binding (virtual functions) is runtime polymorphism.

14. What is 'object composition'? (Level 2)

Answer: In composition, one class has an instance of another class as a data member. In OOP, this relationship is also known as 'has-a' relationship.

15. What are 'abstract' classes? (Level 2)

Answer: Object orientation provides the ability to model the real-world entities in programming. For example, it is possible to write a `Car` class that models car objects that we use. Object orientation also supports modeling of abstract concepts. For example, we can create a `Vehicle` class that models various vehicles like `Car`, `Bike`, etc. Such classes are known as 'abstract' classes.

An abstract class cannot be instantiated. Also, an abstract class has at least one method that does not have a definition. Such methods are known as abstract methods. A class that inherits from an abstract class is also an abstract class. Such a class can become a concrete class only if all the abstract methods in the base classes are defined.

16. What are 'concrete' classes? (Level 2)

Answer: 'Concrete' classes can be instantiated (in other words, objects can be created from concrete classes). These classes have no abstract methods. Concrete classes are a logical complement of abstract classes.

17. What are 'interfaces' (or 'pure' abstract classes)? (Level 2)

Answer: Object-oriented design involves identifying abstract relationships and common properties between the classes while modeling real-world entities. When there are abstract properties that are common to various objects (sometimes unrelated objects), we can use 'interfaces'. Interfaces just specify a set of methods that should be implemented by all the classes that implement them. They do not provide any implementation of methods. Because of this, interfaces are also known as pure abstract classes.

Consider iterating over a collection of objects. There is a wide variety of collections available—sets, maps, trees, graphs, vectors, lists, hash tables, etc.—that can be used for different purposes. One common property for these collection classes is that the data in the container can be accessed one by one, i.e., the container can be iterated. So, we can define an `Iterator` interface that provides methods such as `HasNext()`, `Current()`, and `MoveNext()`.

18. What is 'inheritance'? What are the two main types of inheritance? (Level 2)

Answer: 'Inheritance' is a reusability mechanism in object-oriented programming in which the common properties of various objects are exploited to form relationships with each others. The abstract and common properties are provided in the super class and those properties are available to the more specialized subclasses. This type of relationship is common in the real world so it acts as a way to model the real-world objects in programming.

If a class has a unique base class, it is single inheritance. If the derived class has multiple base classes, it is referred to as multiple inheritance. For example, a whale inherits the properties of a mammal (single inheritance). A two-in-one set derives both form tape and a radio (multiple inheritance).

19. What is class 'cohesion'? (Level 3)

Answer: 'Cohesion' indicates how closely the members are related to each other or how strongly the members depend on each other in a class. Highly cohesive classes or modules indicate good design.

20. What is class 'coupling'? (Level 3)

Answer: 'Coupling' means how two (or more classes) are dependent or strongly related to each other. Wher two classes are tightly coupled, change in one class usually requires change in the other class. Therefore, tightly coupled classes are not recommended.

21. What are the problems with 'multiple inheritance'? (Level 3)

Answer: 'Multiple inheritance' is common in the real world, so it is also useful in modeling real-world entities in programming. Though convenient to use, there are many problems in inheriting from multiple classes. A few important ones are:

Name clashes: When the base classes have members with the same name, then the derived classes deriving from multiple base classes will inherit the multiple members with the same name. Such name clashes are usually resolved by the programmer.

Repeated common base classes: When inheriting from multiple base classes, they can in turn inherit from a common base class. So, the same base class can be inherited twice. In those classes, resolution of the class should be done programatically by using language features.

Order of initialization issues: When a class has multiple base classes, the order of initialization of base class members depends on the programming language used. Such initialization order is often difficult to understand and follow correctly. Wrong initialization order can lead to subtle bugs.

Complexity: Multiple base classes can lead to complex inheritance hierarchies. Understanding the relationship between classes, how the members inherit features from the base classes, etc., can be difficult for the programmers who use the classes in a complex hierarchy.

To summarize, understanding, supporting, and using multiple inheritance to solve design problems can be complicated. Therefore, experienced class designers recommend avoiding multiple inheritance.

22. What is 'dynamic binding'? What is the significance of dynamic binding in OOP? (Level 3)

Answer: Resolving the method call at runtime is known as dynamic binding. 'Dynamic binding' is an essential feature in any OOP language. Dynamic binding is also known as 'virtual method call,' 'late binding,' 'dynamic method resolution,' and 'dynamic method invocation.'

In procedural programming, all functions are statically resolved. Object-oriented programming (OOP) supports virtual functions that are dynamically resolved. Based on the actual type of the object (if it's a base class object or any of the derived class objects) at runtime, method calls are resolved at runtime. This is a powerful feature and provides the basis of runtime polymorphism in object-oriented programming.

23. What do you mean by 'static type' and 'dynamic type' of an object? (Level 3)

Answer: In inheritance, a base class pointer or reference can hold a derived class object. We can invoke methods from the base class pointer or reference. If such a method is a virtual method, then the actual method invoked depends on the 'dynamic type' of the object pointed by the base class pointer or reference. The type of the base class pointer or reference is known as the 'static type' of the object. The actual runtime type of the actual object pointed by the pointer or reference is known as the 'dynamic type' of the object.

24. What are class 'libraries'? (Level 3)

Answer: A class 'library' is a set of reusable classes meant for providing a specific functionality (such as utility, networking, or user-interface related classes) that can be readily used by the application.

In procedural languages, functions provide the basis for reuse of functionality. The applications call functions from the runtime library for reuse. Class libraries are object-oriented equivalent for runtime libraries in procedural languages. They are usually class hierarchies and provide a higher level of abstraction than the direct code reuse.

25. What are 'monomorphic' and 'polymorphic' classes? (Level 4)

Answer: There are some classes that are used only for abstraction/ encapsulation and provide a specific functionality. They also do not make use of inheritance/virtual functions. Classes that do not have any virtual functions (runtime polymorphism) are known as "monomorphic" classes. The classes that have virtual functions (or virtual base classes) and are designed for making use of runtime polymorphism are known as "polymorphic" classes.

26. What are 'frameworks'? (Level 4)

Answer: 'Frameworks' provide domain-specific inheritance hierarchies that are meant for rapid application development in that domain.

Frameworks can be thought of as template for creating readymade applications. A framework always has a set of classes related by inheritance. Few of the methods are left unimplemented. Depending on specific needs, those methods can be overridden to create a readily usable application.

27. How do you choose between interfaces and abstract classes? (Level 4)

Answer: Interfaces and abstract classes have similar constraints: they cannot be directly instantiated and they have to be extended or implemented by the derived classes. However, depending on the purpose or need, we can decide if we want to implement an abstract class or an interface.

Abstraction or Common Contract: If the class we provide abstracts common functionality from various concrete classes, provide an abstract class. If the class just provides signature of common functions to all derived classes (i.e., a contract), provide an interface.

Partial or No Implementation: If the class can be partially implemented, provide an abstract class. If it is not possible to provide any methods common to all the implementation, provide an interface.

Number of Base Classes: Languages like Java do not support multiple inheritance. When we need to derive from more than one class, provide an interface. If the derived classes can implement only that class, provide an abstract class.

28. What are 'design patterns'? (Level 4)

Answer: 'Design patterns' are reusable, extensible solutions to common design problems faced by designers of object-oriented systems.

It is observed that similar pattern of solutions to problems recur in various application domains. Design patterns capture the semantic characteristics of such solutions so that the user can possibly use it in similar circumstances for solving his or her own problems. Thus, patterns expose the developer to the various alternatives or possibilities available, giving him or her an outline of the solutions that were used successfully in past projects.

Design patterns are language-neutral and are of a higher level of abstraction than code.

29. What are the differences between class libraries and frameworks? (Level 5)

Answer: Class libraries are a set of classes that provide reusable classes for a specific problem domain. Frameworks are inheritance hierarchies that provide readymade solutions for a specific problem domain.

The major differences between class libraries and frameworks are:

- Class libraries need not have classes related by an inheritance hierarchy. However, frameworks are always classes provided in an inheritance hierarchy.
- An application uses a class library by instantiating necessary classes from that library. However, in frameworks, an application is created by overriding necessary methods in the framework classes.
- An application calls class library code whereas a framework calls application-specific code.
- A class library is a general set of classes. It can even contain one or more frameworks. A framework does not contain class libraries.

30. What are 'adaptor' classes? Give an example for adaptor class from STL library in C++. (Level 5)

Answer: 'Adaptor' classes modify the existing interface of an existing class and provide a new interface. In other words, an adapter class 'adapts' a class for a specific purpose.

The `std::stack` class is a classic example for an adaptor class from C++ STL library. The `stack` class modifies the interface of the `deque` class and hence it is an adapter class.

SECTION II

C Programming

5

C Multiple Choice Questions

In this chapter, we'll cover multiple choice questions in C that are useful for cracking written tests. Each of the following questions has a unique answer. For programming questions, assume that necessary header files (such as <stdio.h>) are included in the beginning of the programs.

1. **C is derived from which of the following languages? (Level 1)**

   (A) Fortran language

   (B) Pascal language

   (C) C++ language

   (D) B language

   Answer: (D) B language. The C language evolved from B programming language.

2. **Which of the following is *not* both unary and binary operator? (Level 1)**

   (A) & operator

   (B) / operator

   (C) + operator

   (D) * operator

   Answer: (B) / operator. The / (division) operator is not a unary operator. Others are also unary operators.

3. **Which of the following is a bit-wise operator? (Level 1)**

   (A) Arrow operator (->)

   (B) Not operator (!)

   (C) Dot operator (.)

(D) Tilde operator (~)

Answer: (D) Tilde operator (~). It is a bit-wise NOT operator. Note that not operator (!) is a logical operator and not a bit-wise operator.

4. Which of the following keywords does *not* indicate a storage class? (Level 1)

(A) const keyword

(B) extern keyword

(C) static keyword

(D) auto keyword

Answer: (A) const keyword. The keywords for storage classes are auto, register, extern, and static. Const and volatile are type qualifiers.

5. What is the output of the following program? (Level 1)

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

(A) 0

(B) 24

(C) 120

(D) 240

Answer: (A) 0. Since the initial value of fact is 0, whatever the value multiplied by that value is also 0. So the program prints 0.

6. Which of the following expressions is true? (Level 1)

```
struct s {
    int i;
    float f;
};

union u {
    int i;
    float f;
};
```

(A) sizeof(struct s) > sizeof(union u)

(B) sizeof(struct s) < sizeof(union u)

(C) sizeof(struct s) == sizeof(union u)

(D) sizeof(struct s) <= sizeof(union u)

Answer: (A) sizeof(struct s) > sizeof(union u). The size of struct is the size of all of its members (plus any padding). Size of union is equal to the size of the largest member of the union. So, in this code, size of the struct is greater than the size of the union.

7. What is the output of the following program? (Level 1)

```
#define ADD(x) x + x
#define SUB(x) x - x

int main() {
  int y = ADD(3) / SUB(3);
  printf("%d", ADD(y));
}
```

(A) Divide by zero exception at runtime

(B) 1

(C) 2

(D) –2

Answer: (C) 2. The expression ADD(3) expands to 3 + 3. Expression SUB(3) expands to 3 – 3. So the initialization expression for y is 3 + 3 / 3 – 3. The / operator has higher precedence, so the expression is treated as (3 + (3 / 3) – 3), which is (3 + 1 – 3), which is 1. ADD(y) results in (y + y), which is 2.

8. Fill in the correct storage class in the given underlined blank space: "The compiler will give an error if we attempt to get the address of a variable with ——— storage class." (Level 1)

(A) register keyword

(B) extern keyword

(C) static keyword

(D) auto keyword

Answer: (A) register keyword.

9. Which of the following is *not* a stream that is opened by every C program? (Level 1)

(A) stdlog

(B) stdout

(C) stdin

(D) stderr

Answer: (A) stdlog. Every C program opens three streams—stdin, stdout, and stderr.

10. What is the return type of `malloc` function? (Level 1)

(A) void

(B) int

(C) int *

(D) void *

Answer: (D) void *. The declaration for `malloc` function is "void * malloc(size_t); "

11. Which of the following options describes the behavior of the following program? (Level 2)

```
auto int i;
int main() { }
```

(A) Compiler error

(B) No errors

(C) Linker error

(D) Runtime error

Answer: (A) Compiler error. Auto storage class can be used only for local variables.

12. Which of the following is *not* a numbering system supported for integer constants in C? (Level 2)

(A) Binary system

(B) Octal system

(C) Decimal system

(D) Hexadecimal system

Answer: (A) Binary system. Integer constants can be octal (prefixed by 0), decimal (usual constants), or hexadecimal (prefixed by 0x). C does not support representing integer constants as a binary number.

13. What is the output of the following program? (Level 2)

```
int main() {
  char str1[]="Hello";
  char str2[5]="Hello";
  printf("%d %d", sizeof(str1), sizeof(str2));
}
```

(A) 6 6

(B) 5 5

(C) Compiler error

(D) 6 5

Answer: (D) 6 5. The array syntax [] is for leaving the compiler to find the size of the array from the initialization expression. In `str1`, for the string "Hello" including space for the null termination character '\0', the size of the string is 6. For `str2`, we explicitly specify the size of the string as 5, which does not include the '\0' character. So, the program prints 6 5.

14. What is the output of the following program? (Level 2)

```
int foo(int x) {
  if(x <= 0)
    return 0;
  else
    return foo(x - 2) + x;
}

int main() {
  printf("%d", foo(6));
}
```

(A) 12

(B) 10

(C) 6

(D) 0

Answer: (A) 12. The function foo adds value of x decremented by 2 every time it is recursively called.

15. What is the output of the following program? (Level 2)

```
int main() {
  char str1[]="Hello";
  char str2[]="World";
  str1 = str2;
  printf("%s %s", str1, str2);
}
```

(A) Compiler error

(B) Hello World

(C) World Hello

(D) World World

Answer: (A) Compiler error. The variables `str1` and `str2` are arrays; it is not possible to assign to arrays and hence the compiler issues an error.

16. Which of the following operators has lowest precedence? (Level 2)

(A) Comma operator (,)

(B) Ternary operator (?:)

(C) Member access operator (.)

(D) Sizeof operator (sizeof)

Answer: (A) Comma operator. The comma operator has the lowest precedence in C.

17. Which of the following is a compile-time operator? (Level 2)

(A) Comma operator (,)

(B) Ternary operator (?:)

(C) Array access operator ([])

(D) Sizeof operator (sizeof)

Answer: (D) Sizeof operator. The sizeof operator is the only operator in C that is fully evaluated at compile-time itself. Therefore, it is referred to as compile-time operator.

18. Which of the following operators take three operands? (Level 2)

(A) Comma operator (,)

(B) Ternary operator (?:)

(C) Member access operator (.)

(D) Sizeof operator (sizeof)

Answer: (B) Ternary operator. As the name indicates, it is the only operator in C that takes three operands.

19. Which of the following operators can result in divide-by-zero error? (Level 2)

(A) * operator

(B) % operator

(C) . (dot) operator

CRACKING *the* C, C++, *and* Java INTERVIEW

S G Ganesh is currently working at Siemens, Bangalore. He has also authored the book *60 Tips on Object Oriented Programming* (Tata McGraw Hill). He can be reached at sgganesh@gmail.com.

Visit us at : www.tatamcgrawhill.com

The McGraw·Hill Companies

Mc Graw Hill **Professional**

ISBN-13: 978-0-07-007792-8
ISBN-10: 0-07-007792-4

9 780070 077928