[Why does scanf() need “%lf” for doubles, when printf() is okay with just “%f”?](http://stackoverflow.com/questions/210590/why-does-scanf-need-lf-for-doubles-when-printf-is-okay-with-just-f)

Q: Why is it that scanf() needs the l in "%lf" when reading a double, when printf() can use "%f" regardless of whether its argument is a double or a float?

Example code:

double d;

scanf("%lf", &d);

printf("%f", d);

A: Because C will promote floats to doubles for functions that take variable arguments. Pointers aren't promoted to anything, so you should be using %lf, %lg or %le (or %la in C99) to read in doubles.

# Q: [What is the difference between exit(0) and exit(1) in C?](http://stackoverflow.com/questions/9944785/what-is-the-difference-between-exit0-and-exit1-in-c)

**What is the difference between exit(0) and exit(1) in C language?**

A:

exit(0) indicates successful program termination & it is fully portable, While  
exit(1) (usually) indicates unsucessful termination. However, it's usage is non-portable.

Note that the C standard defines EXIT\_SUCCESS and EXIT\_FAILURE to return termination status from a C program.

0 and EXIT\_SUCCESS are the values specified by the standard to indicate successful termination, however, only EXIT\_FAILURE is the standard value for returning unsucessful termination. 1 is used for the same in many implementations though.

# Q: [What is the difference between %0.2lf and %.2lf as printf placeholders?](http://stackoverflow.com/questions/15765289/what-is-the-difference-between-0-2lf-and-2lf-as-printf-placeholders)

I am aware that putting any number of 0's before the width of the placeholder implements zero-padding. For example, printf("%02d", 6); prints 06.

But what does putting a single 0 before the precision of the placeholder do? For example, for both printf("%0.2lf", 0.123); and printf("%.2lf", 0.123);, the output is 0.12.

If it does nothing, is there a preferred format?

A:

They are "equivalent". If you were to use "%07.2", then it would make a difference, by adding extra zeros on the front.

Edit: Originally had "%04.2", which of course doesn't make any difference, because a float with two decimals is always 4 wide anyway.

These examples should show the difference:

"%0.2lf", 0.123 -> 0.12 (zero padded min. width of 0, 2 decimal places).

"%6.2lf", 0.123 -> \_\_0.12 (space padded min. width of 6, 2 decimal places).

"%06.2lf", 0.123 -> 000.12 (zero padded min. width of 6, 2 decimal places).

"%0.6lf", 0.123 -> 0.123000 (min width of 0, 6 decimal places).

The first zero specifies zero padding, followed by the minimum width, which has a default of 0. Thus it is effectively ignored by itself (since you cannot pad 0 width).

Incidentally, the correct form is %f, not %lf for printf

## **Example: Factorial of a Number**

#include <stdio.h>

int main()

{

int n, i;

unsigned long long factorial = 1;

printf("Enter an integer: ");

scanf("%d",&n);

// show error if the user enters a negative integer

if (n < 0)

printf("Error! Factorial of a negative number doesn't exist.");

else

{

for(i=1; i<=n; ++i)

{

factorial \*= i; // factorial = factorial\*i;

}

printf("Factorial of %d = %llu", n, factorial);

}

return 0;

}

THE MOST IMPORTANT OF THIS CODE IS THAT-

factorial \*= i; // factorial = factorial\*i;

## C program to sort 2d array by row or column

**#include<stdio.h>**

**#include<conio.h>**

**int** main( )

{

**int** a[][6]={

{25,64,96,32,78,27}, //Desired solution : {25,27,32,64,78,96},

{50,12,69,78,32,92} // {50,92,78,12,32,69}

};

**int** i, j, k, temp, temp1 ;

//Bubble sorting is applieed on one first row while the other row is swapped

**for**(j=1;j<6;j++)

{

**for**(i=0; i<5; i++)

{

**if**(a[0][i]>a[0][i+1])

{

temp=a[0][i];

a[0][i]=a[0][i+1];

a[0][i+1]=temp;

temp1 = a[1][i];

a[1][i] = a[1][i+1];

a[1][i+1]=temp1;

}

}

}

printf ( **"\n\nArray after sorting:\n"**) ;

**for** ( i = 0 ; i <2; i++ )

{

**for**(j=0; j<6; j++)

{

printf ( **"%d\t"**, a[i][j] ) ; //printing sorted array

}

printf(**"\n"**);

}

getch();

}

# Q: [What does `if(!\*str) …` mean for a `char\* str`?](http://stackoverflow.com/questions/33730314/what-does-ifstr-mean-for-a-char-str)

What does !\*s do in this function:

void f( char \*s) {

if( !\*s ) {

return;

}

f( s+1 );

putchar( \*s );

}

int main( void ) {

f("kernighan");

putchar('\n');

return 0;

}

The output of this program is nahginrek; which I think it swapped the left character with the right character and keeps doing it till it reaches the middle?

A:

It checks if the character pointed to by s is ascii null (NUL) '\0' which is the string delimiter (last char) in C strings.

!\*s will be true if \*s is '\0'.

Note that it is not the same as checking if s is NULL which means that the pointer s points to address zero.

Q: Which is better/more generally accepted?

This:

if(condition)

{

statement;

}

Or:

if(condition)

statement;

I tend to prefer the first one, because I think it makes it easier to tell what actually belongs in the if block, it saves others from adding the braces later (or creating a bug by forgetting to), and it makes all your if statements uniform instead of some with braces and some without. The second one, however, is still syntactically correct and definitely more compact. I'm curious to see which is more generally preferred by others though.

A:

The first is better because the second is error-prone. For example, let's say you are temporarily commenting out code to debug something:

if(condition)

// statement;

otherStatement;

Or adding code in a hurry:

if(condition)

statement;

otherStatement;

This is obviously bad. On the other hand, the first one does feel too verbose at times. Therefore, I prefer to just put everything on one line if it's sufficiently short and simple:

if(condition) statement;

This cuts down on syntactic noise while making the construct look like it does what it actually does, making it less error-prone. Provided that this syntax is only used for very simple, short conditions and statements, I find it perfectly readable.

### Q: TIME COMPLEXITY??

### Definition of "big Omega"

We need the notation for the **lower bound**. A capital omega Ω notation is used in this case. We say that f(n) = Ω(g(n)) when there exist constant c that f(n) ≥ c\*g(n) for for all sufficiently large n. Examples

* n = Ω(1)
* n2 = Ω(n)
* n2 = Ω(n log(n))
* 2 n + 1 = O(n)

### Definition of "big Theta"

To measure the complexity of a particular algorithm, means to find the upper and lower bounds. A new notation is used in this case. We say that f(n) = Θ(g(n)) if and only f(n) = O(g(n)) and f(n) = Ω(g(n)). Examples

* 2 n = Θ(n)
* n2 + 2 n + 1 = Θ( n2)

### Analysis of Algorithms

The term analysis of algorithms is used to describe approaches to the study of the performance of algorithms. In this course we will perform the following types of analysis:

* the *worst-case runtime complexity* of the algorithm is the function defined by the maximum number of steps taken on any instance of size a.
* the *best-case runtime complexity* of the algorithm is the function defined by the minimum number of steps taken on any instance of size a.
* the *average case runtime complexity* of the algorithm is the function defined by an average number of steps taken on any instance of size a.
* the *amortized runtime complexity* of the algorithm is the function defined by a sequence of operations applied to the input of size a and averaged over time.

**Example.** Let us consider an algorithm of sequential searching in an array.of size n.

Its *worst-case runtime complexity* is O(n)   
Its *best-case runtime complexity* is O(1)   
Its *average case runtime complexity* is O(n/2)=O(n)

### Amortized Time Complexity

Consider a dynamic array stack. In this model push() will double up the array size if there is no enough space. Since copying arrays cannot be performed in constant time, we say that push is also cannot be done in constant time. In this section, we will show that push() takes amortized constant time.

Let us count the number of copying operations needed to do a sequence of pushes.

|  |  |  |  |
| --- | --- | --- | --- |
| **push()** | **copy** | **old array size** | **new array size** |
| 1 | 0 | 1 | - |
| 2 | 1 | 1 | 2 |
| 3 | 2 | 2 | 4 |
| 4 | 0 | 4 | - |
| 5 | 4 | 4 | 8 |
| 6 | 0 | 8 | - |
| 7 | 0 | 8 | - |
| 8 | 0 | 8 | - |
| 9 | 8 | 8 | 16 |

We see that 3 pushes requires 2 + 1 = 3 copies.

We see that 5 pushes requires 4 + 2 + 1 = 7 copies.

We see that 9 pushes requires 8 + 4 + 2 + 1 = 15 copies.

In general, 2n+1 pushes requires 2n + 2n-1+ ... + 2 + 1 = 2n+1 - 1 copies.

Asymptotically speaking, the number of copies is about the same as the number of pushes.

2n+1 - 1

limit --------- = 2 = O(1)

n→∞ 2n + 1

We say that the algorithm runs at **amortized constant time**.

# Q: [How to reverse a singly linked list using only two pointers?](http://stackoverflow.com/questions/1801549/how-to-reverse-a-singly-linked-list-using-only-two-pointers)

I would be wondered if there exists some logic to reverse the linked list using only two pointers.

The following is used to reverse the single linked list using three pointers namely p, q, r:

struct node

{

int data;

struct node \*link;

};

void reverse()

{

struct node \*p = first,

\*q = NULL,

\*r;

while (p != NULL)

{

r = q;

q = p;

p = p->link;

q->link = r;

}

q = first;

}

Is there any other alternate to reverse the linked list? what would be the best logic to reverse a singly linked list, in terms of time complexity?

Any alternative? No, this is as simple as it gets, and there's no fundamentally-different way of doing it. This algorithm is already O(n) time, and you can't get any faster than that, as you must modify every node.

It looks like your code is on the right track, but it's not quite working in the form above. Here's a working version:

#include <stdio.h>

typedef struct Node {

char data;

struct Node\* next;

} Node;

void print\_list(Node\* root) {

while (root) {

printf("%c ", root->data);

root = root->next;

}

printf("\n");

}

Node\* reverse(Node\* root) {

Node\* new\_root = 0;

while (root) {

Node\* next = root->next;

root->next = new\_root;

new\_root = root;

root = next;

}

return new\_root;

}

int main() {

Node d = { 'd', 0 };

Node c = { 'c', &d };

Node b = { 'b', &c };

Node a = { 'a', &b };

Node\* root = &a;

print\_list(root);

root = reverse(root);

print\_list(root);

return 0;

# Implement Stack using Queues

A stack can be implemented using two queues. Let stack to be implemented be ‘s’ and queues used to implement be ‘q1′ and ‘q2′. Stack ‘s’ can be implemented in two ways:

**Method 1 (By making push operation costly)**  
This method makes sure that newly entered element is always at the front of ‘q1′, so that pop operation just dequeues from ‘q1′. ‘q2′ is used to put every new element at front of ‘q1′.

push(s, x) // x is the element to be pushed and s is stack

1) Enqueue x to q2

2) One by one dequeue everything from q1 and enqueue to q2.

3) Swap the names of q1 and q2

// Swapping of names is done to avoid one more movement of all elements

// from q2 to q1.

pop(s)

1) Dequeue an item from q1 and return it.

**Method 2 (By making pop operation costly)**  
In push operation, the new element is always enqueued to q1. In pop() operation, if q2 is empty then all the elements except the last, are moved to q2. Finally the last element is dequeued from q1 and returned.

push(s, x)

1) Enqueue x to q1 (assuming size of q1 is unlimited).

pop(s)

1) One by one dequeue everything except the last element from q1 and enqueue to q2.

2) Dequeue the last item of q1, the dequeued item is result, store it.

3) Swap the names of q1 and q2

4) Return the item stored in step 2.

// Swapping of names is done to avoid one more movement of all elements

// from q2 to q1.

**References:**  
[Implement Stack using Two Queues](http://stackoverflow.com/questions/688276/implement-stack-using-two-queues)

### Asked in: [Accolite](http://www.practice.geeksforgeeks.org/tag-page.php?isCmp=1&tag=Accolite), [Adobe](http://www.practice.geeksforgeeks.org/tag-page.php?isCmp=1&tag=Adobe), [Amazon](http://www.practice.geeksforgeeks.org/tag-page.php?isCmp=1&tag=Amazon), [CouponDunia](http://www.practice.geeksforgeeks.org/tag-page.php?isCmp=1&tag=CouponDunia), [Grofers](http://www.practice.geeksforgeeks.org/tag-page.php?isCmp=1&tag=Grofers), [Kritikal Solutions](http://www.practice.geeksforgeeks.org/tag-page.php?isCmp=1&tag=Kritikal%20Solutions), [Oracle](http://www.practice.geeksforgeeks.org/tag-page.php?isCmp=1&tag=Oracle),[Snapdeal](http://www.practice.geeksforgeeks.org/tag-page.php?isCmp=1&tag=Snapdeal)

This article is compiled by **Sumit Jain**and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

# Backtracking | Set 2 (Rat in a Maze)

We have discussed Backtracking and Knight’s tour problem in [Set 1](http://geeksforgeeks.org/?p=12916). Let us discuss Rat in a [Maze](http://en.wikipedia.org/wiki/Maze)as another example problem that can be solved using Backtracking.

A Maze is given as N\*N binary matrix of blocks where source block is the upper left most block i.e., maze[0][0] and destination block is lower rightmost block i.e., maze[N-1][N-1]. A rat starts from source and has to reach destination. The rat can move only in two directions: forward and down.  
In the maze matrix, 0 means the block is dead end and 1 means the block can be used in the path from source to destination. Note that this is a simple version of the typical Maze problem. For example, a more complex version can be that the rat can move in 4 directions and a more complex version can be with limited number of moves.

Following is an example maze.

Gray blocks are dead ends (value = 0).

[](http://geeksforgeeks.org/wp-content/uploads/ratinmaze_filled11.png)

Following is binary matrix representation of the above maze.

{1, 0, 0, 0}

{1, 1, 0, 1}

{0, 1, 0, 0}

{1, 1, 1, 1}

Following is maze with highlighted solution path.

[](http://geeksforgeeks.org/wp-content/uploads/ratinmaze_filled_path1.png)

Following is the solution matrix (output of program) for the above input matrx.

{1, 0, 0, 0}

{1, 1, 0, 0}

{0, 1, 0, 0}

{0, 1, 1, 1}

All enteries in solution path are marked as 1.

**Naive Algorithm**  
The Naive Algorithm is to generate all paths from source to destination and one by one check if the generated path satisfies the constraints.

while there are untried paths

{

generate the next path

if this path has all blocks as 1

{

print this path;

}

}

**Backtrackng Algorithm**

If destination is reached

print the solution matrix

Else

a) Mark current cell in solution matrix as 1.

b) Move forward in horizontal direction and recursively check if this

move leads to a solution.

c) If the move chosen in the above step doesn't lead to a solution

then move down and check if this move leads to a solution.

d) If none of the above solutions work then unmark this cell as 0

(BACKTRACK) and return false.

**Implementation of Backtracking solution**

* C/C++
* Java

|  |
| --- |
| /\* C/C++ program to solve Rat in a Maze problem using     backtracking \*/  #include<stdio.h>    // Maze size  #define N 4    bool solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N]);    /\* A utility function to print solution matrix sol[N][N] \*/  void printSolution(int sol[N][N])  {      for (int i = 0; i < N; i++)      {          for (int j = 0; j < N; j++)              printf(" %d ", sol[i][j]);          printf("\n");      }  }    /\* A utility function to check if x,y is valid index for N\*N maze \*/  bool isSafe(int maze[N][N], int x, int y)  {      // if (x,y outside maze) return false      if(x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1)          return true;        return false;  }    /\* This function solves the Maze problem using Backtracking.  It mainly     uses solveMazeUtil() to solve the problem. It returns false if no     path is possible, otherwise return true and prints the path in the     form of 1s. Please note that there may be more than one solutions,     this function prints one of the feasible solutions.\*/  bool solveMaze(int maze[N][N])  {      int sol[N][N] = { {0, 0, 0, 0},          {0, 0, 0, 0},          {0, 0, 0, 0},          {0, 0, 0, 0}      };        if(solveMazeUtil(maze, 0, 0, sol) == false)      {          printf("Solution doesn't exist");          return false;      }        printSolution(sol);      return true;  }    /\* A recursive utility function to solve Maze problem \*/  bool solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N])  {      // if (x,y is goal) return true      if(x == N-1 && y == N-1)      {          sol[x][y] = 1;          return true;      }        // Check if maze[x][y] is valid      if(isSafe(maze, x, y) == true)      {          // mark x,y as part of solution path          sol[x][y] = 1;            /\* Move forward in x direction \*/          if (solveMazeUtil(maze, x+1, y, sol) == true)              return true;            /\* If moving in x direction doesn't give solution then             Move down in y direction  \*/          if (solveMazeUtil(maze, x, y+1, sol) == true)              return true;            /\* If none of the above movements work then BACKTRACK:              unmark x,y as part of solution path \*/          sol[x][y] = 0;          return false;      }        return false;  }    // driver program to test above function  int main()  {      int maze[N][N]  =  { {1, 0, 0, 0},          {1, 1, 0, 1},          {0, 1, 0, 0},          {1, 1, 1, 1}      };        solveMaze(maze);      return 0;  } |

Run on IDE

Output: The 1 values show the path for rat

1 0 0 0

1 1 0 0

0 1 0 0

0 1 1 1

# [What does O(log n) mean exactly?](http://stackoverflow.com/questions/2307283/what-does-olog-n-mean-exactly)

I am currently learning about Big O Notation running times and amortized times. I understand the notion of *O(n)* linear time, meaning that the size of the input affects the growth of the algorithm proportionally...and the same goes for, for example, quadratic time *O(n2)* etc..even algorithms, such as permutation generators, with *O(n!)* times, that grow by factorials.

For example, the following function is *O(n)* because the algorithm grows in proportion to its input *n*:

f(int n) {

int i;

for (i = 0; i < n; ++i)

printf("%d", i);

}

Similarly, if there was a nested loop, the time would be O(n2).

But what exactly is *O(log n)*? For example, what does it mean to say that the height of a complete binary tree is *O(log n)*?

I do know (maybe not in great detail) what Logarithm is, in the sense that: log10 100 = 2, but I cannot understand how to identify a function with a logarithmic time.

A:

I cannot understand how to identify a function with a log time.

The most common attributes of logarithmic running-time function are that:

* the choice of the next element on which to perform some action is one of several possibilities, and
* only one will need to be chosen.

or

* the elements on which the action is performed are digits of n

This is why, for example, looking up people in a phone book is O(log n). You don't need to checkevery person in the phone book to find the right one; instead, you can simply divide-and-conquer, and you only need to explore a tiny fraction of the entire space before you eventually find someone's phone number.

Of course, a bigger phone book will still take you a longer time, but it won't grow as quickly as the proportional increase in the additional size.

We can expand the phone book example to compare other kinds of operations and their running time. We will assume our phone book has businesses (the "Yellow Pages") which have unique names andpeople (the "White Pages") which may not have unique names. A phone number is assigned to at most one person or business. We will also assume that it takes constant time to flip to a specific page.

Here are the running times of some operations we might perform on the phone book, from best to worst:

* **O(1) (worst case):** Given the page that a business's name is on and the business name, find the phone number.
* **O(1) (average case):** Given the page that a person's name is on and their name, find the phone number.
* **O(log n):** Given a person's name, find the phone number by picking a random point about halfway through the part of the book you haven't searched yet, then checking to see whether the person's name is at that point. Then repeat the process about halfway through the part of the book where the person's name lies. (This is a binary search for a person's name.)
* **O(n):** Find all people whose phone numbers contain the digit "5".
* **O(n):** Given a phone number, find the person or business with that number.
* **O(n log n):** There was a mix-up at the printer's office, and our phone book had all its pages inserted in a random order. Fix the ordering so that it's correct by looking at the first name on each page and then putting that page in the appropriate spot in a new, empty phone book.

For the below examples, we're now at the printer's office. Phone books are waiting to be mailed to each resident or business, and there's a sticker on each phone book identifying where it should be mailed to. Every person or business gets one phone book.

* **O(n log n):** We want to personalize the phone book, so we're going to find each person or business's name in their designated copy, then circle their name in the book and write a short thank-you note for their patronage.
* **O(n2):** A mistake occurred at the office, and every entry in each of the phone books has an extra "0" at the end of the phone number. Take some white-out and remove each zero.
* **O(n · n!):** We're ready to load the phonebooks onto the shipping dock. Unfortunately, the robot that was supposed to load the books has gone haywire: it's putting the books onto the truck in a random order! Even worse, it loads all the books onto the truck, then checks to see if they're in the right order, and if not, it unloads them and starts over. (This is the dreaded **[bogo sort](http://en.wikipedia.org/wiki/Bogosort)**.)
* **O(nn):** You fix the robot so that it's loading things correctly. The next day, one of your co-workers plays a prank on you and wires the loading dock robot to the automated printing systems. Every time the robot goes to load an original book, the factory printer makes a duplicate run of all the phonebooks! Fortunately, the robot's bug-detection systems are sophisticated enough that the robot doesn't try printing even more copies when it encounters a duplicate book for loading, but it still has to load every original and duplicate book that's been printed.

# Q: C Program to Solve Josephus Problem using Linked List

This C Program Solves the Josephus Problem using Linked List. Josephus Problem talks about a problem where there are people standing in a circle waiting to be executed. The counting out begins at some point in the circle and proceeds around the circle in a fixed direction. In each step, a certain number of people are skipped and the next person is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last person remains, who is given freedom.

Here is source code of the C Program to Solve Josephus Problem using Linked List. The C program is successfully compiled and run on a Linux system. The program output is also shown below.

1. */\**
2. *\* C Program to Solve Josephus Problem using Linked List*
3. *\*/*
4. #include <stdio.h>
5. #include <stdlib.h>
7. struct node
8. {
9. int num;
10. struct node \*next;
11. };
13. void create(struct node \*\*);
14. void display(struct node \*);
15. int survivor(struct node \*\*, int);
17. int main()
18. {
19. struct node \*head = NULL;
20. int survive, skip;
22. create(&head);
23. printf("The persons in circular list are:**\n**");
24. display(head);
25. printf("Enter the number of persons to be skipped: ");
26. scanf("%d", &skip);
27. survive = survivor(&head, skip);
28. printf("The person to survive is : %d**\n**", survive);
29. free(head);
31. return 0;
32. }
34. int survivor(struct node \*\*head, int k)
35. {
36. struct node \*p, \*q;
37. int i;
39. q = p = \*head;
40. while (p->next != p)
41. {
42. for (i = 0; i < k - 1; i++)
43. {
44. q = p;
45. p = p->next;
46. }
47. q->next = p->next;
48. printf("%d has been killed.**\n**", p->num);
49. free(p);
50. p = q->next;
51. }
52. \*head = p;
54. return (p->num);
55. }
57. void create (struct node \*\*head)
58. {
59. struct node \*temp, \*rear;
60. int a, ch;
62. do
63. {
64. printf("Enter a number: ");
65. scanf("%d", &a);
66. temp = (struct node \*)malloc(sizeof(struct node));
67. temp->num = a;
68. temp->next = NULL;
69. if (\*head == NULL)
70. {
71. \*head = temp;
72. }
73. else
74. {
75. rear->next = temp;
76. }
77. rear = temp;
78. printf("Do you want to add a number [1/0]? ");
79. scanf("%d", &ch);
80. } while (ch != 0);
81. rear->next = \*head;
82. }
84. void display(struct node \*head)
85. {
86. struct node \*temp;
88. temp = head;
89. printf("%d ", temp->num);
90. temp = temp->next;
91. while (head != temp)
92. {
93. printf("%d ", temp->num);
94. temp = temp->next;
95. }
96. printf("**\n**");
97. }

advertisements

$ **gcc** josephus.c

$ .**/**a.out

Enter a number: 1

Do you want to add a number **[**1**/**0**]**? 1

Enter a number: 2

Do you want to add a number **[**1**/**0**]**? 1

Enter a number: 3

Do you want to add a number **[**1**/**0**]**? 1

Enter a number: 4

Do you want to add a number **[**1**/**0**]**? 1

Enter a number: 5

Do you want to add a number **[**1**/**0**]**? 1

Enter a number: 6

Do you want to add a number **[**1**/**0**]**? 1

Enter a number: 7

Do you want to add a number **[**1**/**0**]**? 0

The persons **in** circular list are:

1 2 3 4 5 6 7

Enter the number of persons to be skipped: 3

3 has been killed.

6 has been killed.

2 has been killed.

7 has been killed.

5 has been killed.

1 has been killed.

The person to survive is : 4

# C library function - isdigit()

## **Description**

The C library function **void isdigit(int c)** checks if the passed character is a decimal digit character.

Decimal digits are (numbers) − 0 1 2 3 4 5 6 7 8 9.

## **Declaration**

Following is the declaration for isdigit() function.

int isdigit(int c);

## **Parameters**

* **c** − This is the character to be checked.

## **Return Value**

This function returns non-zero value if c is a digit, else it returns 0.

Q:

|  |
| --- |
| **Question 12**  **WRONG** |

A function f defined on stacks of integers satisfies the following properties. f(∅) = 0 and f (push (S, i)) = max (f(S), 0) + i for all stacks S and integers i.

If a stack S contains the integers 2, -3, 2, -1, 2 in order from bottom to top, what is f(S)?

|  |  |
| --- | --- |
| A | 6 |
| B | 4 |
|  | 3 |
|  | 2 |

[**Stack**](http://quiz.geeksforgeeks.org/?page_id=164)[**Gate IT 2005**](http://quiz.geeksforgeeks.org/?page_id=22019)  
[**Discuss it**](http://quiz.geeksforgeeks.org/gate-gate-it-2005-question-13/)

**Question 12 Explanation:**

f(S) = 0, max(f(S), 0) = 0, i = 2 f(S)new = max(f(S), 0) + i = 0 + 2 = 2   
f(S) = 2, max(f(S), 0) = 2, i = -3 f(S)new = max(f(S), 0) + i = 2 - 3 = -1   
f(S) = -1, max(f(S), 0) = 0, i = 2 f(S)new = max(f(S), 0) + i = 0 + 2 = 2   
f(S) = 2, max(f(S), 0) = 2, i = -1 f(S)new = max(f(S), 0) + i = 2 - 1 = 1   
f(S) = 1, max(f(S), 0) = 1, i = 2 f(S)new = max(f(S), 0) + i = 1 + 2 = 3   
   
Thus, option (C) is correct.   
   
Please comment below if you find anything wrong in the above post.

NOTE f(S) never takes the whole stack but only the elements of the stack.

# Q: [error: unknown type name ‘bool’](http://stackoverflow.com/questions/8133074/error-unknown-type-name-bool)

I downloaded the source code and wanted to compile the file of scanner. It produces this error:

[meepo@localhost cs143-pp1]$ gcc -o lex.yy.o lex.yy.c -ll

In file included from scanner.l:15:0:

scanner.h:59:5: error: unknown type name ‘bool’

In file included from scanner.l:16:0:

utility.h:64:38: error: unknown type name ‘bool’

utility.h:74:1: error: unknown type name ‘bool’

In file included from scanner.l:17:0:

errors.h:16:18: fatal error: string: No such file or directory

compilation terminated.

And I tried to use different complier to compile it, but it appeared different errors.

[meepo@localhost cs143-pp1]$ g++ -o scan lex.yy.c -ll

/usr/bin/ld: cannot find -ll

collect2: ld returned 1 exit status

My os is 3.0-ARCH, I don't know why this happened. How do I fix the error?

myQuestion – I got error in my code in DS with bool IsPresent() function??

A:

C90 does not support the boolean data type.

C99 does include it with this include:

#include <stdbool.h>

# Q: [Shouldn't the average search time for a linked list be O(N/2)?](http://stackoverflow.com/questions/20793020/shouldnt-the-average-search-time-for-a-linked-list-be-on-2)

I keep seeing [the search time](http://bigocheatsheet.com/) for linked lists listed as O(N) but if you have 100 elements in a list aren't you on average only comparing against 50 of them before you've found a match?

So is O(N/2) being rounded to O(N) or am I just wrong in thinking it's N/2 on average for a linked list lookup?

Thanks!

A:

The thing is, the order is really only talking about how the time increases as n increases.

So O(N) means that you have linear growth. If you double N then the time taken also doubles. N/2and N both have the same growth behaviour so in terms of Order they are identical.

Functions like log(N), and N^2 on the other hand have non-linear growth, N^2 for example means that if you double N the time taken increases 4 times.

It is all about ratios. If something on average takes 1 minute for 1 item will it on average take 2 minutes or 4 minutes for 2 items? O(N) will be 2 minutes, O(N^2) will take 4 minutes. If the original took 1 second then O(N) will take 2 seconds, O(N^2) 4 seconds.

The algorithm that takes 1 minute and the algorithm that takes 1 second are both O(N)!

# Q: [Hash table runtime complexity (insert, search and delete)](http://stackoverflow.com/questions/9214353/hash-table-runtime-complexity-insert-search-and-delete)

A:

[Hash tables](http://en.wikipedia.org/wiki/Hash_table) are O(1) **average and**[**amortized**](http://en.wikipedia.org/wiki/Amortized_analysis) case complexity, however it suffers from O(n)**worst case** time complexity. [And I think this is where your confusion is]

Hash tables suffer from O(n) worst time complexity due to two reasons:

1. If too many elements were hashed into the same key: looking inside this key may take O(n)time.
2. Once a hash table has passed its [load balance](http://en.wikipedia.org/wiki/Load_balancing_%28computing%29) - it has to rehash [create a new bigger table, and re-insert each element to the table].

However, it is said to be O(1) average and amortized case because:

1. It is very rare that many items will be hashed to the same key [if you chose a good hash function and you don't have too big load balance.
2. The rehash operation, which is O(n), can at most happen after n/2 ops, which are all assumedO(1): Thus when you sum the average time per op, you get : (n\*O(1) + O(n)) / n) = O(1)

Note because of the rehashing issue - a realtime applications and applications that need low [latency](http://en.wikipedia.org/wiki/Latency_%28engineering%29) - should not use a hash table as their data structure.

**EDIT:** Annother issue with hash tables: [cache](http://en.wikipedia.org/wiki/Cache)   
Another issue where you might see a performance loss in large hash tables is due to cache performance. **Hash Tables suffer from bad cache performance**, and thus for large collection - the access time might take longer, since you need to reload the relevant part of the table from the memory back into the cache.

# Q: [Hash table - why is it faster than arrays?](http://stackoverflow.com/questions/12020984/hash-table-why-is-it-faster-than-arrays)

A:

In cases where I have a key for each element and I don't know the index of the element into an array, hashtables perform better than arrays (O(1) vs O(n)).

The hash table search performs O(1) in the average case. In the worst case, the hash table search performs O(n): when you have collisions and the hash function always returns the same slot. One may think "this is a remote situation," but a good analysis should consider it. In this case you should iterate through all the elements like in an array or linked lists (O(n)).

Why is that? I mean: I have a key, I hash it.. I have the hash.. shouldn't the algorithm compare this hash against every element's hash? I think there's some trick behind the memory disposition, isn't it?

You have a key, You hash it.. you have the hash: the index of the hash table where the element is present (if it has been located before). At this point you can access the hash table record in O(1). If the load factor is small, it's unlikely to see more than one element there. So, the first element you see should be the element you are looking for. Otherwise, if you have more than one element you must compare the elements you will find in the position with the element you are looking for. In this case you have O(1) + O(number\_of\_elements).

In the average case, the hash table search complexity is O(1) + O(load\_factor) = O(1 + load\_factor).

Remember, load\_factor = n in the worst case. So, the search complexity is O(n) in the worst case.

I don't know what you mean with "trick behind the memory disposition". Under some points of view, the hash table (with its structure and collisions resolution by chaining) can be considered a "smart trick".

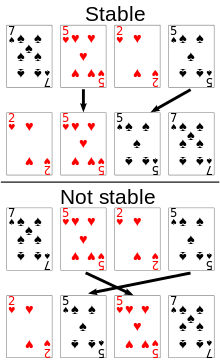
Of course, the hash table analysis results can be proven by math.

Q:

Heap sort is unstable sort why?

A:

### Stability**[**[**edit**](https://en.wikipedia.org/w/index.php?title=Sorting_algorithm&action=edit&section=2)**]**

[](https://en.wikipedia.org/wiki/File:Sorting_stability_playing_cards.svg)

An example of stable sort on playing cards. When the cards are sorted by rank with a stable sort, the two 5s must remain in the same order in the sorted output that they were originally in. When they are sorted with a non-stable sort, the 5s may end up in the opposite order in the sorted output.

When sorting some kinds of data, only part of the data is examined when determining the sort order. For example, in the card sorting example to the right, the cards are being sorted by their rank, and their suit is being ignored. This allows the possibility of multiple different correctly sorted versions of the original list. Stable sorting algorithms choose one of these, according to the following rule: if two items compare as equal, like the two 5 cards, then their relative order will be preserved, so that if one came before the other in the input, it will also come before the other in the output

# Inorder Successor in Binary Search Tree

In Binary Tree, Inorder successor of a node is the next node in Inorder traversal of the Binary Tree. Inorder Successor is NULL for the last node in Inoorder traversal.  
In Binary Search Tree, Inorder Successor of an input node can also be defined as the node with the smallest key greater than the key of input node. So, it is sometimes important to find next node in sorted order.

[](http://geeksforgeeks.org/wp-content/uploads/2009/09/BST_LCA.gif)

In the above diagram, inorder successor of **8**is **10**, inorder successor of **10**is **12**and inorder successor of **14**is**20**.

Check out more of it here- <http://www.geeksforgeeks.org/inorder-successor-in-binary-search-tree/>

# Q: [What is satellite information in data structures?](http://stackoverflow.com/questions/14551845/what-is-satellite-information-in-data-structures)

A:

Satellite data refers to any "payload" data which you want to store in your data structure and which is*not* part of the *structure* of the data structure. It can be anything you want. It can be a single value, a large collection of values, or a pointer to some other location that holds the value.

For example, here's a list node for a singly linked list whose satellite data is a single integer:

struct node

{

node \* next;

int satellite;

};

In other words, the whole value of any given data structure lies in the data which it contains, which is the satellite data in your book's terminology. The data structure will additionally consume structural data (like the next pointer in the example) to perform the algorithms which define it, but those are essentially "overhead" from the user's perspective.

For associative containers, the "key" value performs a dual role: On the one hand it is user data, but on the other hand it is also part of the structure of the container. However, a tree can be equipped with additional satellite data, in which case it becomes a "map" from key data to satellite data.

At one extreme you have a fixed-size array which has *no* overhead and only payload data, and on the other extreme you have complicated structures like multiindexes, tries, Judy arrays, or lockfree containers which maintain a comparably large amount of structural data.

Q: How to delete the root of a tree data structure??

A:  
/\*  This function traverses tree in post order to

    to delete each and every node of the tree \*/

void deleteTree(struct node\* node)

{

    if (node == NULL) return;

    /\* first delete both subtrees \*/

    deleteTree(node->left);

    deleteTree(node->right);

    /\* then delete the node \*/

    printf("\n Deleting node: %d", node->data);

    free(node);

}

The above deleteTree() function deletes the tree, but doesn’t change root to NULL which may cause problems if the user of deleteTree() doesn’t change root to NULL and tires to access values using root pointer. We can modify the deleteTree() function to take reference to the root node so that this problem doesn’t occur. See the following code.

/\*  This function is same as deleteTree() in the previous program \*/

void \_deleteTree(struct node\* node)

{

    if (node == NULL) return;

    /\* first delete both subtrees \*/

    \_deleteTree(node->left);

    \_deleteTree(node->right);

    /\* then delete the node \*/

    printf("\n Deleting node: %d", node->data);

    free(node);

}

/\* Deletes a tree and sets the root as NULL \*/

void deleteTree(struct node\*\* node\_ref)

{

  \_deleteTree(\*node\_ref);

  \*node\_ref = NULL;

}

Obviously the double pointer method is going to help!! This is its power.

Call the function in main as deleteTree(&root);

And rest will happen all by itself.

# Q: How do we calculate space complexity?

A:

The space complexity of an algorithm or data structure is the maximum amount of space used at any one time, ignoring the space used by the input to the algorithm. The notation for space complexity is exactly the same as the notation for time complexity.  
  
Examples:  
  
Binary search uses Θ(1)Θ(1) space\*. The only additional space used is constant number of indices.  
  
Quick sort uses Θ(1)Θ(1) space\* if done in-place (where the input array is modified), but has Θ(n)Θ(n) space complexity otherwise (since it must build a new list of size nn).  
  
Storing a graph in adjacency list form takes Θ(V+E)Θ(V+E) space\*. For each vertex, we store a list of its edges. There are EE edges, so clearly we use Ω(E)Ω(E) space. However, we also need to store Ω(V)Ω(V) references to lists.  
  
Storing a graph in matrix form takes Θ(V2)Θ(V2) space. For each pair of vertices we use Θ(1)Θ(1)space to indicate whether or not the edge between them is present.  
  
  
\* This actually depends on the model you specify. In a more formal model, any index or pointer into a structure of size nn requires Ω(logn)Ω(log⁡n) bits, so the space complexity will be Ω(logn)Ω(log⁡n). Usually a simpler model is used where indices and pointers are considered to take Θ(1)Θ(1) space.  
  
The same kinds of simplifications are also done when considering time complexity. Usually adding two numbers is assumed to be a Θ(1)Θ(1) operation, but adding two nn bit numbers necessarily takes Ω(n)Ω(n) time.

To know more go to-

<https://www.quora.com/How-do-we-calculate-space-time-complexity-of-an-algorithm>

# Q: [How to use redirection in C for file input](http://stackoverflow.com/questions/19467865/how-to-use-redirection-in-c-for-file-input)

I need to get the file from the terminal, I know the command will look like:

./a.out < fileName.txt

I'm not sure how to use fgets() in my program to use the file requested from the terminal.

A:

Using redirection sends the contents of the input file to stdin, so you need to read from stdin inside your code, so something like (error checking omitted for clarity)

#include <stdio.h>

#define BUFFERSIZE 100

int main (int argc, char \*argv[])

{

char buffer[BUFFERSIZE];

fgets(buffer, BUFFERSIZE , stdin);

printf("Read: %s", buffer);

return 0;

}