|  |
| --- |
| ISLAMIC UNIVERSITY OF TECHNOLOGY  Organization of Islamic Cooperation  Board Bazar, Gazipur |

RECURSION

CSE 4107

Talha Ibn Aziz

1. What is recursion? What is the mechanism of recursion?

Recursion is the process by which a function calls itself. Such a function is known as a recursive function. The recursion continues until some condition is met to prevent it. That condition is called the base case. A major problem with a recursive function is the possibility of infinite recursions, where no condition is given to stop the recursions. The system will eventually run out of memory, causing what is known as a “stack overflow”.

A recursion simply starts another instance of the same function, using different input parameters. It should be noted however, that the previous instance of the function has not finished until a value is returned to it from the next function. For example, the following code uses recursion to find the factorial of a positive number.

#include <stdio.h>  
int factorial (int i)  
{  
 if(i == 1) return 1;  
 else return i \* factorial(i - 1);  
}  
int main()  
{  
 int i = 12;  
 printf("The factorial of %d is %d.", i, factorial(i));  
}

C

OUTPUT: The factorial of 12 is 479001600.

The function factorial works by taking an integer i and multiplying it by the factorial of (i - 1). To get the factorial of (i - 1) the function must call itself with the new input parameter, i.e. (i - 1), thus multiplying (i - 1) with the factorial of (i - 2). This process continues until the function is asked for the factorial of i = 1, at which point the function does not call itself, and instead returns the integer 1.

All the results of the previous recursions are now calculated one by one, and the final result is returned to the main function.

The entire process looks somethings like this (if i = 3):

3! (first function call)

2! (second function call)

1! (third function call)

1 (result of third function call)

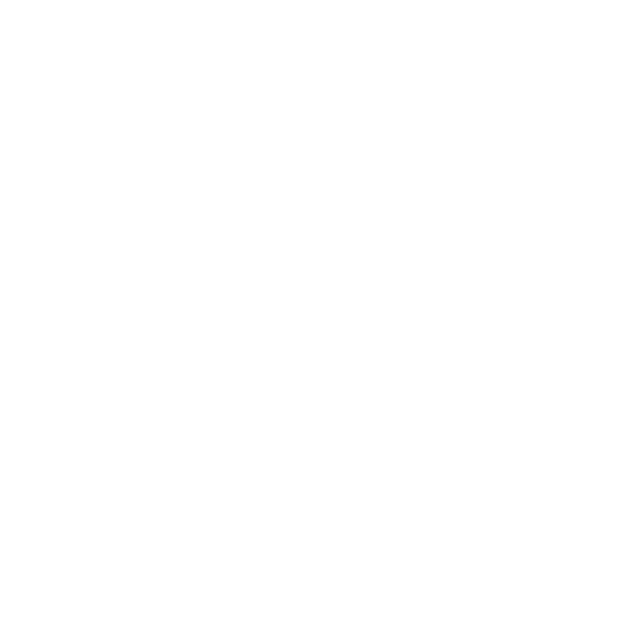
2 x 1 (result of second function call)

3 x 2 x 1 (result of first function call)

6 (result returned to main function)

2. Explain the working principle of the “flood fill” algorithm with code.

The Flood Fill algorithm helps in visiting each and every point in a given area. It determines the area connected to a given cell in a multi-dimensional array, and changes the value of those cells. For example, selecting a single cell and changing its value using this algorithm, will in turn change the values of every cell connected to that cell that had the same value, and every cell connected to those cells. Thus, the Flood Fill algorithm is dependent on recursion. This is the method used in tools like ‘Bucket Fill’ in MS Paint and the game Minesweeper.



For example, in the diagram above, the cell (1, 1) is clearly connected to (0, 0), (0, 1),

(0, 2), (1, 0), (1, 2), (2, 0), (2, 1) and (2, 2). The Flood Fill algorithm would visit each of these cells if the cell (1, 1) was selected, given that none of them lie outside the boundary conditions stated.

The following code is an example of how the Flood Fill algorithm is implemented. For simplicity, cells connected diagonally are being ignored and no special boundary conditions are given.

#include<stdio.h>  
int arr[3][3] = {1, 1, 1, 1, 1, 2, 1, 2, 1};  
printing()  
{  
 int i, j;  
 for (i=0; i<3; i++)  
 {  
 for (j=0; j<3; j++) printf("%d\t", arr[i][j]);  
 printf("\n");  
 }  
}  
  
  
flood\_fill (int a, int b, int old\_value, int new\_value)  
{  
 arr[a][b] = new\_value;  
 if(arr[a+1][b] == old\_value)  
 flood\_fill (a+1, b, old\_value, new\_value);  
 if(arr[a-1][b] == old\_value)  
 flood\_fill (a-1, b, old\_value, new\_value);  
 if(arr[a][b+1] == old\_value)  
 flood\_fill (a, b+1, old\_value, new\_value);  
 if(arr[a][b-1] == old\_value)  
 flood\_fill (a, b-1, old\_value, new\_value);  
}  
  
  
int main ()  
{  
 printf("Before Flood Fill:\n");  
 printing();  
 flood\_fill (0, 0, arr[0][0], 5);  
 printf("After Flood Fill:\n");  
 printing();  
}

C

OUTPUT:

Before Flood Fill:

1 1 1

1 1 2

1 2 1

After Flood Fill:

5 5 5

5 5 2

5 2 1

The code works by giving the initial flood\_fill function two integers (0 and 0) to tell it which cell to start in, the value of that cell and the value to change the cell into. The function changes the required cell to the new value, and checks each of the cells above, below, to the left and to the right to see if they have the old value. If any of them do, the function calls itself recursively and uses that cell’s position as the new input parameters. This continues until every cell that had the old value and was connected uninterruptedly to the first cell has been changed into the new value. It should be noted that the cell at the bottom right remains unchanged, since it is not uninterruptedly connected to the first cell (again, we have ignored diagonal connections).