Tutorial Link https://codequotient.com/tutorials/Recursion - Direct &amp; Indirect/5a0145eecbb2fe34b7775046

TUTORIAL

Recursion - Direct & Indirect

Chapter

Most basic examples of recursion, demonstrate direct recursion, in which a function calls itself. Indirect recursion occurs when a function is called not by itself but by another function that it called (either directly or indirectly). In other words, if you ask someone for his age, and he does not answer you directly. Instead he tells I am 5 years younger than you, so he answer the problem for which you have to find the answer again. Which is a indirect recursion. In Programming, if some function func() calls func(), that is direct recursion, but if func() calls func2() which calls func(), then that is indirect recursion of func(). Following examples show the direct and indirect recursions: -

Direct Recursion

```
// An example of direct recursion
void directRecFun()
{
        // Some code....
        directRecFun();         // Function calling itself.
        // Some code...
}
```

The following program is an example of direct recursion: -

Javascript

```javascript
function fact_recursive(num){
    if (num == 0){
        return 1;
    }
    else{
        let fact = num * fact_recursive(num-1);      // Call recursively with lesser number.
        return fact;
    }
}

```

C

```c
int fact_recursive(int num)
{
  int result;
  if (num == 0)         // Base case for recursion.
    return 1;     // fact() return 1 if argument is 0
  else{
    result = num * fact_recursive(num-1);      // Call recursively with lesser number.
    return result;
  }
}


```

Java

```java
static int fact_recursive(int num)
{
  int result;
  if (num == 0)         // Base case for recursion.
    return 1;     // fact() return 1 if argument is 0
  else
  {
    result = num * fact_recursive(num-1);      // Call recursively with lesser number.
    return result;
  }
}

```

```python
def fact_recursive(num):
  if (num == 0):                   # Base case for recursion.
    return 1                       # fact() return 1 if argument is 0
  else:
    result = num * fact_recursive(num-1);      # Call recursively with lesser number.
    return result

```

Python 3

```cpp
int fact_recursive(int num)
{
  int result;
  if (num == 0)          // Base case for recursion.
    return 1;     // fact() return 1 if argument is 0
  else
  {
    result = num * fact_recursive(num-1);      // Call recursively with lesser number.
    return result;
  }
}
```

C++

Indirect recursion

Indirect recursion looks like the following calls: -

```
void indirectRecFun1()
{
        // Some code...
        indirectRecFun2();              // Fun1 calling fun2()
        // Some code...
}
void indirectRecFun2()
{
        // Some code...
        indirectRecFun1();              // Fun2 is calling fun1()
again for recursion
        // Some code...
}
```

The following program is an example of indirect recursion: -

Javascript

```
function func1(number)  {
    if(number>0){
        console.log("\nIn func1(), Number=" + number);
        func2(number-1);      // Call func2() with lesser number.
    }
    else
        console.log("\nIn func1(), Number=" + number);
}

function func2(number)  {
    if(number>0)
    {
        console.log("\nIn func2(), Number=" + number);
        func1(number-1);      // Call func1() with lesser number.
    }
    else
        console.log("\nIn func2(), Number=" + number);
}

function main(){
    let number = 5;
    func1(number);
}

main()
```

C

```
void func1(int number)
{
  if(number>0)
  {
    printf("\nIn func1(), Number=%d\n", number);
    func2(number-1);      // Call func2() with lesser number.
  }
  else
    printf("\nIn func1(), Number=%d\n", number);
}

void func2(int number)
```

```
{
  if(number>0)
  {
    printf("\nIn func2(), Number=%d\n", number);
    func1(number-1);      // Call func1() with lesser number.
  }
  else
    printf("\nIn func2(), Number=%d\n", number);
}

int main()
{
  int number;
  number = 5;
  func1(number);        // Call the Recursive version
  return 0;
}

```

Java

```
class Main{
  static void func1(int number)
  {
    if(number>0)
    {
      System.out.println("\nIn func1(), Number=" + number);
      func2(number-1);      // Call func2() with lesser number.
    }
    else
      System.out.println("\nIn func1(), Number=" + number);
  }

  static void func2(int number)
  {
    if(number>0)
    {
      System.out.println("\nIn func2(), Number=" + number);
      func1(number-1);      // Call func1() with lesser number.
```

```java
    }
    else
      System.out.println("\nIn func2(), Number=" + number);
  }

  public static void main(String[] args)
  {
    int number;
    number = 5;
    func1(number);
  }
}
```

Python 3

```python
def func1(number):
  if(number>0):
    print("\nIn func1(), Number= " + str(number))
    func2(number-1)              # Call func2() with lesser number.
  else:
    print("\nIn func1(), Number=" + str(number))

def func2(number):
  if(number>0):
    print("\nIn func2(), Number= " + str(number))
    func1(number-1)      # Call func1() with lesser number.
  else:
    print("\nIn func2(), Number=" + str(number))

number = 5
func1(number)
```

C++

```cpp
void func1(int number)
{
  if(number>0)
  {
    cout<<"\nIn func1(), Number="<<number<<endl;
    func2(number-1);      // Call func2() with lesser number.
  }
  else
    cout<<"\nIn func1(), Number="<<number<<endl;
}

void func2(int number)
{
  if(number>0)
  {
    cout<<"\nIn func2(), Number="<<number<<endl;
    func1(number-1);      // Call func1() with lesser number.
  }
  else
    cout<<"\nIn func2(), Number="<<number<<endl;
}

int main()
{
  int number;
  number = 5;
  func1(number);        // Call the Recursive version
  return 0;
}

```

Recursion is better to implement, when the problem is inherently recursive, like traversing a tree from root node, Towers of Hanoi, searching an array using binary search etc. The binary search algorithm is a method of searching a sorted array for a single element by cutting the array in half with each recursive pass. The trick is to pick a midpoint near the center of the array, compare the data at that point with the data being searched and then responding to one of three possible conditions: the data is found at the midpoint, the data at the midpoint is greater than the data being searched for, or the data at the midpoint is less than the data being searched for. Recursion is used in this algorithm because with each pass a new array is created by cutting the old one in half. The binary search procedure is then called recursively, this time on the new (and smaller) array. Typically the array's size is adjusted by manipulating a beginning and ending index. The algorithm exhibits a logarithmic order of growth because it essentially divides the problem domain in half with each pass.

An example of indirect recursion is determine whether an integer is even or odd:

How do we know if a number is even?

1. we know 0 is even.

2. we know that if n is even, then n-1 must be odd.

So following two functions can serve this purpose

```
int is_even(int n)
{
        if (n==0)
                return 1;
        else
                return(is_odd(n-1));
}
```

And how do we know if a number is odd?

1. It's not even.

```
int is_odd(int n)
{
        return (!is_even(n));
}
```

Both above functions are calling each other showing indirect recursion.

Javascript

```
function is_even(n){
  if (n==0)
    return 1
  else
    return(is_odd(n-1))
}

function is_odd(n){
  return(!is_even(n))
}

function main(){
    console.log(is_even(4))
}

main()
```

C

```c
int is_even(int n)
{
  if (n==0) return 1;
  else return(is_odd(n-1));
}

int is_odd(int n)
{
  return(!is_even(n));
}

int main(){
  printf("%d",is_even(4));
  return 0;
}

```

Java

```java
class Main {
  static boolean is_even(int n){
    if (n==0) return true;
    else return(is_odd(n-1));
  }

  static boolean is_odd(int n)
  {
    return(!is_even(n));
  }

  public static void main(String[] args)
  {
    System.out.print(is_even(4));
  }
}
```

Python 3

```
def is_even(n):
  if n==0:
    return 1
  else:
    return(is_odd(n-1))

def is_odd(n):
  return(not is_even(n))

print(is_even(4));
```

C++

```
int is_even(int n)
{
  if (n==0) return 1;
  else return(is_odd(n-1));
}

int is_odd(int n)
{
  return(!is_even(n));
}

int main(){
  cout<<is_even(4);
  return 0;
}
```