CME 3202 - Concepts of Programming Languages

Laboratory Worksheet#2

Recursion in C/C++

Recursion is a programming technique which allows the programmer to express operations in terms of themselves. In C/C++, this takes the form of a function which calls itself. A useful way to think of ***"Recursive Functions"*** is to imagine them as a process which is performed where one of the instructions is to ***"Repeat the Process".*** This makes it sound very similar to a loop (looping) because it repeats the same code. On the other hand, recursion makes it easier to express ideas in which the result of the recursive call is necessary to complete the task. Of course, it must be possible for the process to sometimes be completed without the recursive call. One simple example is that the idea of building a wall that is ten feet high:

If it is wanted to build a ten-foot-high wall, then it will be first built a nine-foot-high wall, and then add an extra foot of bricks. Conceptually, this is like saying the ***"build wall"*** function which takes a height, and if that height is greater than one, first calls itself to build a lower wall, and then adds one a foot of bricks.

Example 1

Example about recursive functions in ***C/C++ programming language***. Apply the following steps in order to accomplish the tasks.

Step 1

Open https://repl.it. Create a new repl by clicking +new repl button.

Step 2

Select ***C*** as the programming language. Click on the text editor, and add the following lines, and click the ***"Run"*** button.

```
void recurse()
{
  recurse(); // Function calls itself.
}

int main()
{
  recurse(); // Sets off the recursion.
}
```

Step 3

Paste the output of the code.

```
...Program finished with exit code 139
Press ENTER to exit console.
```

Step 4

Is it possible to see how many times the ***"recurse()"*** function is called before the program terminates? If it is possible, please write required code statements in the following.

If you want to exit program like the last step make sure that you don't print all the steps just print the last step and after some iterations program exit like in the step 3.

```
#include <stdio.h>

int counter = 0;
int maxCalls = 10000000; // Set a limit to avoid infinite recursion

void recurse() {
   counter++;
   // Only print the counter when the maximum call is reached
   if (counter == maxCalls) {
      printf("Total call count: %d\n", counter);
      return; // Terminate recursion at the base-case.
   }
   recurse();
}

int main() {
   recurse();
   return 0;
}
```

Step 5

Explain what the program does, and in which condition it terminates.

```
#include <stdio.h>
int fact(int number)
{
  if (number==1) return 1;
  return number*fact(number-1); //Function calls itself.
}

int main()
{
  printf("sayi %d\n", fact(3));
  return 0;
}
```

The function fact() computes the factorial of a number recursively. For fact(3), the function computes 3 × fact(2), where fact(2) computes 2 × fact(1). The final result is 3 × 2 × 1 = 6.

Step 6

Write the above program in non-recursive way in ***C*** programming language in the following.

```
#include <stdio.h>

int main() {
   int number = 3;
   int result = 1;

   for (int i = 1; i <= number; i++) {
      result *= i;  // result = result * i
```

```
  }

  printf("Result: %d\n", result);
  return 0;
}
```

Loops in C/C++

Loops are used to repeat a block of code. Being able to have your program repeatedly executed a block of code is one of the most basic and useful tasks in programming - many programs or websites which produce extremely complex output (such as a message board) are really only executing a single task in many times.

They may be executing a small number of tasks, but in principle, in order to produce a list of messages, it only requires repeating the operation of reading in some data and displaying it. So, one should think about what this means: a loop lets you write ***a very simple statement*** to produce a ***significantly greater result*** simply by repetition.

Example 2

Example about loops in ***C/C++ programming language***. Apply the following steps in order to accomplish the tasks.

Step 1

Open https://repl.it. Create a new repl by clicking +new repl button.

Step 2

Select ***C++*** as the programming language. Click on the text editor, and add the following lines, and click the ***"Run"*** button.

```
#include <iostream>

using namespace std;

int main()
{
    for ( int x = 1; x <= 10; x++ ) {
    cout<< x <<endl;
  }
}
```

Step 3

Paste the output of the code in the following.

```
1
2
3
4
5
6
7
8
```

```
9
10
```

Step 4

Write same codes with the above part by using ***"while"*** and ***"do-while"*** loops in ***C++ programming language***.

```
 #include <iostream>
using namespace std;

int main() {
   int x = 1;
   while (x <= 10) {
      cout << x << endl;
      x++;
   }
   return 0;
}
```

Step 5

Write one occasion (case) in which ***for-loops*** are more advantageous, and one that ***while-loops*** are more advantageous.

```
#include <iostream>
using namespace std;

int main() {
   int x = 1;
   do {
      cout << x << endl;
      x++;
   } while (x <= 10);
   return 0;
}
```

Recursion & Loops in Python

Example 3

Example about recursion and loops in ***Python programming language***. Apply the following steps in order to accomplish the tasks.

Step 1

Open https://repl.it. Create a new repl by clicking +new repl button.

Step 2

Select ***Python*** as the programming language. Click on the text editor, add the following lines, and click the ***"Run"*** button.

```
def absolute_value(num):
  """This function returns the absolute
  value of the entered number"""

  if num >= 0:
    return num
  else:
    return -num

print(absolute_value(2))

print(absolute_value(-4))
```

Step 3

Paste the output of the code in the following.

```
2
4
```

Step 4

Write a recursive factorial function and call the function to calculate 4!. Paste your code in the following.

```
def factorial(n):
  # Base case: if n is 0 or 1, return 1.
  if n == 0 or n == 1:
    return 1
  else:
    return n * factorial(n - 1)

print("4! =", factorial(4))
```

Step 5

Write a non-recursive factorial function by using ***"while"*** loops in ***Python programming language***.

```
def factorial_iterative(n):
  result = 1
  while n > 1:
    result *= n
    n -= 1
  return result

print("4! =", factorial_iterative(4))
```