プログラミング演習I第10回レポート

学籍番号：2364902

名前：キム　ギュソク

(1) 課題番号と課題内容

A. 階乗の計算を再帰を利用する関数と、ループを利用して求める。しかし、途中計算がわかるように途中計算を出力しながら計算を行う。

(2) フローチャートまたは疑似言語によるアルゴリズムの記述

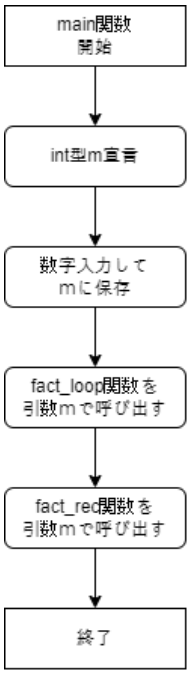

図 1。main関数

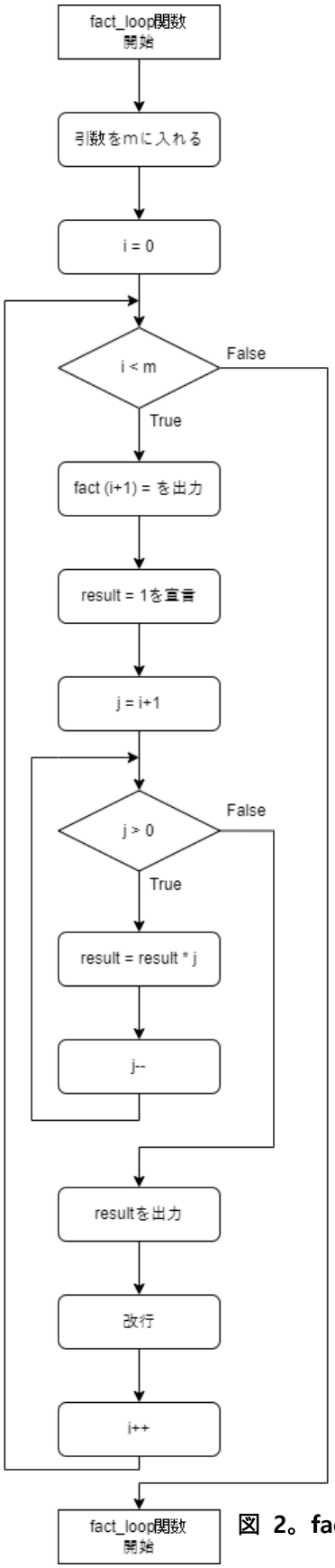

図 2。fact_loop関数

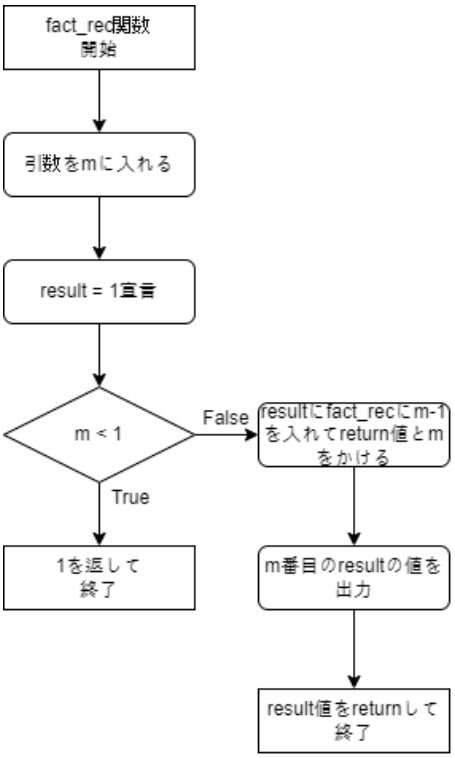

図 3。fact_rec関数

(3) アルゴリズムが正しいことの説明

A. 正当性

i. この問題で求めたい結果は階乗の計算つまりFactorialの計算を求めている。階乗の計算は理論的には次のようになる。

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$$

このような計算を階乗計算という。この計算はnを1ずつ減りながら掛け算をする反復処理を含めているのでループを利用した計算方法と再帰関数を利用したけいさん方法が存在する。

1. ループの計算

A. ループを利用して求めることはとても簡単である。iをインデックスとしておいてそれを増やしながら1からnまでかけるだけである。今回の課題では途中の計算結果がわかるようにするために1!, 2!, 3!, …(n-1)!, n!を全て出力する必要がある。このような計算は1からnまでの繰り返しとその数字の階乗を求めるための繰り返し文二つがあれば計算できる。

$i = k\ (0 \leq k < m), \quad j = k+1$ *の時*
$result_{k+1} = j \times result_k = (k+1) \times result_k$
$result_{k+1} = (k+1) \times k \times result_{k-1}$
$result_{k+1} = (k+1) \times k \times (k-1) \times result_{k-2}$
…
$result_{k+1} = (k+1) \times k \times (k-1) \times \ldots \times 3 \times 2 \times 1$

このような方法を利用してiは0からm-1まで増やしてjにk+1を入れて減らしながら計算を行うとjの値が変わるたびに途中計算結果がわかると同時にmの階乗の理論的な計算と同じ計算になることがわかる。iが0からm-1までなのでk+1の階乗を求めることでmの階乗を求めることができるためこの計算は正当性を持っていると考えられる。

2. 再帰関数の計算

A. この計算はn! = n × (n-1)!であることを利用して求める方法である。計算過程は次のようになる

$n! = n \times (n-1)!$
$n! = n \times (n-1) \times (n-2)!$
$n! = n \times (n-1) \times (n-2) \times (n-3)!$
…
$n! = n \times (n-1) \times (n-2) \times (n-3) \times \ldots \times 3 \times 2 \times 1$

この計算も理論的な階乗の計算と同じ結果が出ることがわかる。よって正当な結果が得られると言える。

B. 停止性

i. このプログラムの中では無限ループを起こす可能性があるループや再帰関数三つ存在する。その各自の正しく停止するかを確認したいと思う。

1. fact_loop関数の内側ループ

A. このループはi＋１番目の階乗の計算を行うためのループである。この計算は1からかけていくのではなく逆順にかけていくのでjの値を1ずつ減らしながら０以下になると止まるように条件を付けているので正しく1まで計算を行って停止するようになっている。

2. fact_loop関数の外側ループ

A. このループは計算過程を示すために1からｎまでの階乗の計算結果を出力するためのループである。単純にUserからｍを入力してもらったら０からm-1つまりm回ループするだけなのでiを1ずつ足しながらiがm未満になるまでの条件を付けることでm-1までの繰り返して停止する。よって、正しく結果が得られると停止する。

(4) ソース・プログラムの説明

A. loopを利用するfact_loop関数と再帰を利用するfact_rec関数のプロトタイプを宣言する。そして、入力された数字を入れるためのｍ変数を宣言する。そして、scanfを利用してｍに入れる。その後、fact_loop関数とfact_rec関数にｍを引数として入れて呼び出す。fact_loop関数ではiを０に初期化してi<m条件を付けてi++でfor文を作る。i+1番目の計算結果を出力するためにfact %dとしてi+1を出力する。その後、i+1番目の階乗の計算を行うためにfor文でj=i+1として初期化してj—しながらj>0という条件を付ける。その繰り返し文の中でresult = result * jとして階乗の計算を行う。階乗の繰り返し計算が終わったらresultを出力する。そして、fact_rec関数では再帰呼び出すための変数resultを1で初期化して宣言する。その後、再帰の終了条件m ＜ 1を設定する。１をreturnする。elseを利用して普段はresultに再帰的なコードを入れてresultを出力してresultをreturnする。

(5) 考察

A. この課題ではループと再帰の違いがわかる。特に、計算過程を出力する中で出力される順番の違いが存在する。loopの場合 0 からm-1までに足しながら繰り返すと順番的に上がることがわかる。それに比べて、再帰の関数ではfact_rec(1)を入れるまでには計算結果がわからないのでfact_rec(1)まで再帰呼び出しを行ってその後から逆順にだんだん上がっていて計算を行うために呼び出す順番と計算の順番が逆になることがloopとの違いである。よって、再帰呼び出しの後に現在の計算の位置を出力する必要がある。また、loopの場合途中の計算結果を出力するために二重ループを使わなければならない。二重ループは実行時間を遅くする処理なので二重ループを使うfact_loopより一つのループと同じ処理を行う再帰を使う関数の方がより実行時間が早く効率的なコードであると考えられる。これを時間複雑度で見るとfact_recはO(n)であると考えられる。しかし、fact_loopの場合二重ループのためO(n^2)である。よって、fact_recの方がより効率的なアルゴリズムであると考えられる。このようなことから、単純に階乗の計算を行う際にはループと再帰は同じ時間複雑度であるため違いはほぼないと思うが途中計算を出力すると時間複雑度はこのように大きな差が出る。

(6) 感想

A. 今まではループと再帰の違いはないと思っていたがこのケースを解いてループと再帰の違いがはっきりわかるようになった。また、再帰関数の計算がどのような順番で行われているかわかるようになった。