プログラミング演習 第 6 回

問 A [06A]: 組み合わせの個数 (再帰関数版)

異なる n 個のものから m 個を選ぶ方法の個数 ${}_n\mathrm{C}_m$ は、

$$ {}_n\mathrm{C}_m = \frac{n!}{m!(n-m)!} $$

である (ただし $0 \leqq m \leqq n$、$n! = n \times (n-1) \times \ldots \times 2 \times 1$、$0! = 1$)。まずは、以下の漸化式が成り立つことを証明し、レポートに記せ (つまりメールの本文に書く)。

$$ {}_n\mathrm{C}_m = {}_{n-1}\mathrm{C}_{m-1} + {}_{n-1}\mathrm{C}_m $$

次に、この漸化式を利用して ${}_n\mathrm{C}_m$ の値を求める再帰関数 *combination* を定義せよ。

この関数のプロトタイプ宣言は以下のようであり、${}_n\mathrm{C}_m$ を計算して返す。

```
int combination(int n, int m);
```

さらに、標準入力 (端末等) から n と m を読み取り、`combination(`n`,` m`)` の結果を表示するプログラムを作成せよ。なお、戻り値の型を `int` とすると、桁溢れを起こして正しくない値になる場合もあるが、ここでは結果が `int` 型に収まるような入力が与えられるものと仮定してよい。

入出力例

入力 1

```
8 2
```

入力 1 に対する出力

```
28
```

入力 2

```
1 1
```

入力 2 に対する出力

```
1
```

入力 3

```
41 7
```

入力 3 に対する出力

```
22481940
```

問 B [06B]: 組み合わせの個数 (非再帰関数版)

スタックを用いて、問 A で作成した再帰関数から再帰を除去せよ。

入出力例

問 A と同じ。

ヒント

計算を継続するためにどの変数の値を保存する必要があるか考える。

問 C [06C]: 実行時スタックの追跡

図 1 で示される関数 *fiv* に対し、`fiv(3)` を呼び出したときに、コード中のコメント /* この時点の様子 n */ における実行時スタックの様子を書け。ただし配列 `v` の `v[0]` には**自分の学籍番号の下二桁**を、`v[1]` には**自分の学籍番号の下二桁*63+15** をセットすること[1]。

実行時スタックの書き方であるが以下のようにする。

[1] たとえば西暦 2050 年入学の学籍番号 5011091 番の学生なら `int v[2] = {91, 91*63+15};` とする。

- 変数 (引数と局所変数) の値を “変数名=値” で表す。値が未定義であれば値のところに疑問符 (?) を書く。複数の変数がある場合、それらをカンマ (,) で区切る。
- 同じレベルの関数呼び出しにおける変数の値 (の並び) を中括弧 ({}) で囲む。複数の呼び出しがある場合、それらをカンマ (,) で区切る。このとき、左から右に向かって呼び出しが時間的に新しくなるように並べる。
- スタック全体を大カッコ ([]) で囲む。
- 各行に指定された位置におけるスタックの様子を実行順に書く。
- 最右端にどの時点の様子かを書く。

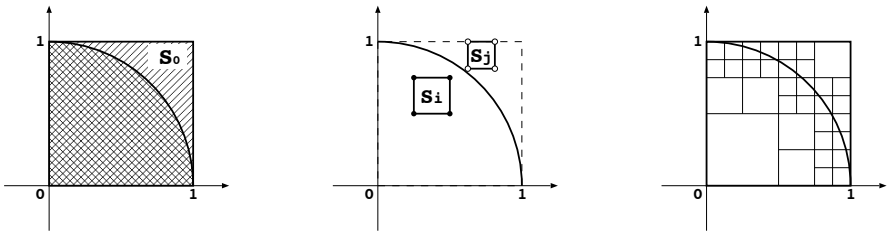
たとえば、図 2 で示す関数 *ss* に対して `ss(4)` と呼び出した場合には以下のようになる (引数 `n` と局所変数 `m` の変化に注目せよ)。

```
[{n=4, m=?}]                                          // ss(4) の呼び出し時の様子 1
[{n=4, m=?}, {n=3, m=?}]                              // ss(3) の呼び出し時の様子 1
[{n=4, m=?}, {n=3, m=?}, {n=2, m=?}]                  // ss(2) の呼び出し時の様子 1
[{n=4, m=?}, {n=3, m=?}, {n=2, m=?}, {n=1, m=?}]      // ss(1) の呼び出し時の様子 1
[{n=4, m=?}, {n=3, m=?}, {n=2, m=1}]                  // ss(2) の呼び出し時の様子 2
[{n=4, m=?}, {n=3, m=5}]                              // ss(3) の呼び出し時の様子 2
[{n=4, m=14}]                                         // ss(4) の呼び出し時の様子 2
```

問 D [06D]: 四分円の面積

次の考え方に基づいて、単位円の第一象限の面積 ($\pi/4$) を求めるものとする。

- $(0,0)$, $(1,0)$, $(0,1)$, $(1,1)$ を頂点とする正方形を S_0 とする。求める面積は、単位円と S_0 が重なる部分の面積である (下図 (a))。
- 第一象限に限定すれば、軸と平行に置かれた正方形 S について次のことが成り立つ。
  - **A.** S の 4 つ全ての頂点が単位円の内部にあれば、単位円と正方形の重なる面積は正方形の面積と一致する (下図 (b) の正方形 S_i)。
  - **B.** S の 4 つ全ての頂点が単位円の外部にあれば単位円と正方形の重なる面積は 0 である (下図 (b) の正方形 S_j)。
- そこで、S_0 を出発点とし、上の **A.** か **B.** のどちらかの条件が成り立つか正方形が十分小さくなるまで、正方形を 4 等分して再帰的に単位円と重なる部分の面積を求める (下図 (c))。十分に小さな正方形まで分解しても **A.** か **B.** の条件を満たさなかった時は、その正方形の面積を 0 とする。



(a) (b) (c)

上記の手順を用いて、与えられた正方形と単位円との重なる面積を計算する再帰関数 *overlap* を定義せよ。プロトタイプ宣言は以下のようであり、(x1, y1) と (x2, y2)(x1≦x2, y1≦y2) を対角の座標とする正方形と単位円とが重なる面積を返す。

```
double overlap(double x1, double y1, double x2, double y2);
```

また、図 3 の *main* 関数と組み合わせ、コンパイル・実行すると、正方形の分解の十分さに応じて $\pi/4$ の近似値を印字するプログラムを作成せよ。

ヒント

どこまで小さく正方形を分解するかを指定する定数をマクロ EPSILON として定義しておき、*overlap* の引数の値から得られる正方形の辺の長さと EPSILON の値を比較して、再帰呼び出しを続けるかどうか判定すればよい。この EPSILON の具体的な値は各自で決定すればよいが、0.78539 までは $\pi/4$ に一致し、かつできる限り早く (少なくとも 3 秒未満で) 結果を得られるようにせよ。また正方形が単位円に含まれるかどうかは原点から $(x1, y1)$ および $(x2, y2)$ までの距離と単位円の半径を比べれば判定できる。

補足

プログラムの実行時間を計測するには time コマンドを使えばよい。たとえば、実行ファイル a.out に対して、以下のようなコマンド行を実行すると、

```
% time ./a.out
0.785396
0.131u 0.007s 0:00.13 100.0%    5+192k 0+0io 0pf+0w
```

のように数値がいくつか表示される。その左端の値として $nn.nnn\mathbf{u}$ などと出力されるが、これはプログラムの実行に $nn.nnn$ 秒かかったことを意味する。よって、本問では 3 以上にならないようにすればよい[2]。

[2]詳しくは time コマンドのマニュアルページに参照のこと。

```
int v[2] = {学籍番号の下二桁, 学籍番号の下二桁 * 63 + 15};

int fiv(int n) {
  int a;
  int b;
  /* この時点の様子 1 */
  if (n <= 1) {
    return v[n];
  } else {
    a = fiv(n - 1);
    /* この時点の様子 2 */
    b = fiv(n - 2);
    /* この時点の様子 3 */
    return a + b;
  }
}
```

図 1: 実行時スタック追跡対象の関数

```
int ss(int n) {
  int m;
  /* この時点の様子 1 */
  if (n < 2) {
    return n;
  } else {
    m = ss(n-1);
    /* この時点の様子 2 */
    return n*n + m;
  }
}
```

図 2: 実行時スタックの書き方を示すための関数例

```
int main(void) {
  printf("%f\n", overlap(0.0, 0.0, 1.0, 1.0));
  return 0;
}
```

図 3: 四分円の面積を印字する main 関数