計算機言語 I 第 8 回 関数

この資料: http://www.math.u-ryukyu.ac.jp/~suga/gengo/2019-1/08.pdf

レポートへのツッコミ

教科書 p.92, 6.6 節の補足

ヘッダ (header) ファイルに対する解説がありますが, 少し補足しておきます.

#include は, cpp(C プリプロセッサ) への指示で, そのあとに続くファイルの内容をプログラムソースに取り込みます. ほとんどの場合, 教科書にあるようにヘッダファイルと呼ばれるものを取り込みますが, それ以外のファイルを取り込まないわけではありません. C のソースファイルを取り込んでも構わないし, プログラムで利用するデータを取り込んでも構わないのです.

この講義で利用している stdio.h, math.h はヘッダファイルと呼ばれます. 多くの場合, ヘッダファイルには, 次の 2 つを記述します.

- cpp のマクロ置換を利用した定数の定義
- プログラムソースに記述する関数のプロトタイプ宣言

大きなプログラムの作成では, ヘッダファイルに上のような関数のプロトタイプと定数定義を記述し, プログラムソース (hogehoge.c) は, 「関数の実装」を書くようにすることが多いようです. このようにすることで, ヘッダファイルを読むだけで, どのようなプログラムを作成しているかの概要がわかるからです.

教科書 p.93, 6.7 節, 再帰 (recursion) の補足

再帰とは, 関数の処理の途中で自分自身を関数呼び出しすることです. 再び自分自身に帰ってくるので再帰と呼ばれます.

計算機概論 I の maple の講義の回で, 再帰の話を 1 度しました. その際にも述べたことですが, 再帰の利点, 欠点は次になります.

- プログラムの記述が簡潔で分かりやすく (読みやすく) なる.
- 関数呼び出しの度にスタック領域と呼ばれるメモリを消費するので, 処理によってはメモリが足りなくなる.
- 関数呼び出しという動作が数多く起こるため, プログラムの動作が遅くなる (今回のレポート問題).

今回のレポート問題にある様に, 再帰を繰り返し処理で書けるのであれば, 実行時間を考えると, 繰り返し処

理で記述方が望ましいのがほとんどです.

ただし, 教科書の Hanoi の塔の様に再帰でしか書けないような処理もありますし, 繰り返し処理に変換するのが大変な場合もあります. 後者の例としては, プログラム言語処理系の作成が良く取り上げられます.

プログラム言語処理系では, 次の動作をして実際にコンピュータで動作するプログラムを作り上げます.

字句解析　プログラムのソースコードを「単語 (word)」に分解して, それがキーワードであるか, 数値であるか, 識別子であるか, 演算子であるかなどを調べる.

構文解析　上の単語の並びから, 実際にコンピュータでしなければならない動作を確定させる.

例えば, C を例にとると,「数から始まる文字列」があれば, それは数値に変換されますし, アルファベットから始まる文字列なら, それは, キーワードか識別子のどちらかをチェックします (字句解析). また, `(x+y)*z` の様な式があれば, $x + y$ を計算してその結果に z を掛けるという動作をさせるように機械語を生成します (構文解析).

このような動作ができるようにプログラミング言語の文法が決まっているわけですが, それを記述するのによく用いられるのが, 拡張バッカスナウア記法 (Extended Backus–Naur form, EBNF) です. EBNF で記述されたプログラム言語の処理系を作成しようとすると, 次のような複数の関数にまたがる再帰処理を書くのが最も自然です (書く方も書きやすいし, 読む方も読みやすい).

```
func1()
{
   ...
   func2();
}

func2()
{
   ...
   func3();
}

...

funcN()
{
   ...
   func1();
}
```

レポート問題

締切: 6 月 10 日 (月) 10:00(JST)

$a_0 = a_1 = 1$, $a_{n+1} = a_n + a_{n-1}$, $(n \geq 1)$ で定義される Fibonacci(フィボナッチ) の数列について, 次の 2 つのプログラムを書き, そのプログラムを提出するとともに, 3 番目の結果をレポートせよ. ただし, どちらの関数も, `unsigned long long` 型 (符号無し 64bit 整数) の変数を利用して, a_{50} の正しい値が表示できるようにすること. `printf` での `unsigned long long` の出力の指定方法は, `%llu`. (`man 3 printf` を参照).

1. 繰り返し (`for` でも `while` でも良い) を用いて, 自然数 n の入力に対して, a_n を出力する関数 `fibonacci(int n)` を書き, `main()` では, a_{50} を出力するプログラム.
   件名: gengo2019-1 report 8-1.

2. 再帰を利用して, 自然数 n の入力に対して, a_n を出力する関数 `fibonacci(int n)` を書き, `main()` では, a_{50} を出力するプログラム.
   件名: gengo2019-1 report 8-2

3. 上の 2 のプログラムの実行は終了するか? 2 のプログラムが終了するなら, 上の 1, 2 のプログラムについて `time` コマンドでレポートされる内容を提出せよ.
   件名: gengo2019-1 report 8-3.

`time` コマンドは, `time ./a.out` で, `a.out` の実行に対して要した CPU の使用時間をレポートします.