

プログラミング入門 II

関数と再帰呼び出し

佐藤

プログラミング入門 II

再帰呼び出しに必要な知識

- ◆ いろいろな変数
 - 局所変数
 - 大域変数
 - 静的変数
- ◆ 有効範囲 (scope)と有効期限 (extent)
- ◆ 引数の値渡し

プログラミング入門 II

いろいろな変数

- ◆ 局所変数 (local variable)
 - 関数の中に宣言されているもの
 - 関数の定義に書かれたパラメータ変数もその1つ
- ◆ 大域変数 (global variable)
 - 関数の外に宣言されているもの
- ◆ 静的変数 (static variable)

プログラミング入門 II

局所変数

- ◆ 局所変数
 - 関数の中に宣言されているもの
 - 関数の定義に書かれたパラメータ変数もその1つ
- ◆ 関数には、いろいろなプログラムが書ける
 - 局所変数は、作業領域としてつかう。
- ◆ 局所変数の値は、関数の実行が終わると無効になる。
- ◆ 違う関数の中の局所変数は、使えない

プログラミング入門 II

大域変数

- ◆ 大域変数
 - 関数の外に宣言されているもの
 - 関数の外に書くことができる
- ◆ 関数の間で共有するデータをいれておくもの。

プログラミング入門 II

大域変数の悪い例

```
int t;                                int goo(int x, int y)
                                     {
int foo(int x,int y)                 t = x-y;
{                                     z = foo(x,y);
  t = x + y;                          ... ;
  return t;                             return t;
}
```

副作用 (side effect)

局所的しか使わない一時的なデータにはできるだけ局所変数を使い、必要とときだけ大域変数を使うようにしましょう。

変数の有効範囲と有効期限

- ◆ 有効範囲 - どこで使えるか
 - 局所変数
 - 大域変数
- ◆ 参照できる範囲 スコープ (Scope) という
- ◆ 有効期限 - 値がいつまで有効か
- ◆ 保持される期間 有効期限 (extent) という

変数の有効期限 (extent)

- ◆ 局所変数は、その宣言された関数が終了すると値が無効になる
 - 変数のメモリが解放される
 - 一時的に用いるのにつかう
- ◆ 大域変数は関数が終わっても値は保持される
 - 関数が終わってもなにか値を保持する必要がある場合には大域変数にいらておく
- ◆ 関数から返すことのできる値は1つなので、たくさんの値を返したい場合には、大域変数を用意してそこにいれて、関数からも戻ったときにその変数を参照する

静的変数 static variable

- ◆ 局所変数にstaticをつける
- ◆ 有効範囲は、局所変数と同じ
- ◆ 有効期限は、大域変数と同じ

```
foo(){
    static int i;
    ...
    i = 10;
    ...
    x = i;
}
```

引数の値渡し

- ◆ 関数呼び出しでは、引数に書かれた式を実行した結果の値が使われる
- ◆ これを値渡し call by value という

引数の値渡し

```
main() {
    int i;
    i = 10;
    j = foo(i);
    printf("i=%d\n", i);
}

int foo(int i)
{
    i = 100;
    return i;
}
```

関数の再帰呼び出し

- ◆ ある関数が、自分自身を呼び出すことを再帰呼び出し(recursive call)

```
int factorial(int n)
{
    if(n == 0) return 1;
    else return n*factorial(n-1);
}
```

プログラミング入門 II

関数の再帰呼び出し

- ◆ ある関数が、自分自身を呼び出すことを **再帰呼び出し(recursive call)**
- ◆ 再帰呼び出しの考え方は情報科学のなかでもっとも重要なことのひとつ
- ◆ この考え方を身につけることによって、いろいろな問題が簡単に解ける強力な考え方です。

プログラミング入門 II

再帰の考え方

- ◆ あるnについて問題を解くときに、n-1についての解答でとくことができるときに使うことができます。
- ◆ プログラムを考えるときに、特定のケース、特定の数についてプログラムを考えるだけでなく、それを一般的な入力、一般的なnについて考えることはプログラミングについての非常に重要な考え方です。

プログラミング入門 II

ハノイの塔

- ◆ ハノイの塔とは、3本の柱があり、一番左の柱に下から大きな盤、その上に順に小さな盤が乗っているというもの
- ◆ 問題は以下の条件で、左の柱から、右の柱に移すというパズル
 - 一度に1枚の盤しか移すことができない(片手しか使えない)
 - 小さな盤の上に大きな盤を乗せてはいけない(壊れてしまう)
- ◆ 中間的な置き場として真ん中の柱を使うことができますが、上の条件は満たしていません。

プログラミング入門 II

ハノイの塔

- ◆ まず、大小2枚の場合を考えてみる。
 - まず小さい盤を真ん中に移して、大きな盤を左に移し、最後に小さい盤を左に移せばよいことになります。
 - では3枚ではどうでしょうか。この場合、まず、2枚の手順を使って、上2枚を真ん中に移します。それで、一番下にある盤を右の柱に移し、また2枚の手順を使って、真ん中から右に移せばいいということになります。
- ◆ これを一般化して、n枚の場合には、
 - n-1枚をあいているところに移す。
 - 1番下の番を目的のところ(右の柱)に移す。
 - n-1枚を目的のところに移す。
- ◆ n枚の盤を移すという関数を定義して、その関数に再帰呼び出しを使うことで、エレガントにプログラミングすることができます。

プログラミング入門 II

数字のプリント

- ◆ 任意の整数を一文字の出力関数 (putchar)を使って、プリントアウトするという問題です。例えば、123という数字の場合は"1","2","3"と出力するという問題
 - 一桁の場合であれば、単にその数字を'0'に加えることによって、出力します。
 - では2桁はどうでしょう。この場合はまず、10で割って、その商が2桁目なので、その数字を印刷します。あまりを1桁にしてプリントすればいいわけです。
- ◆ これを一般化してn桁の数とすると、数dに対し、
 - まず、一桁、つまり10よりも小さければ、putchar(d+'0')
 - 最下位以外のn-1桁について、プリントする。つまり、d/10を印刷。
 - 最下位の桁、d%10を印刷。

プログラミング入門 II

数字のプリント

```
void print_number(int d)
{
  if(d >=10) print_number(d/10);
                /* 上の桁を出力 */
  putchar('0'+d%10);
                /* 最下位の桁を出力 */
}
```

ナップサック問題

- ◆ いくつかの荷物を袋に最大の値段になるように袋に詰める組み合わせを求める問題
- ◆ N 個の荷物があ、り、個々の荷物の重さを w_i 、値段を p_i とする。袋(knapsack)には最大 W の重さまで入れることができる。このとき、袋に入れることができる荷物の組み合わせを求め、そのときの値段を求めなさい。
- ◆ 簡単に考えれば、 2^N の組み合わせがある。

```
int knap_search(int i,int cp, int M)
{
    int Opt;
    int ii,m,l,r;
    if (i < N && M > 0){
        if(M >= W[i]){
            l = knap_seach(i+1,cp+P[i],M-W[i]);
            r = knap_serach(i+1,cp,M);
            if(l > r) Opt = l;
            else Opt = r;
        } else
            Opt = bb_seq(i+1,cp,M);
        } else Opt = cp;
    return(Opt);
}
```

```
int N;
int Cap;
int W[MAX_N];
int P[MAX_N];
```

```
int main()
{
    データの読み込み...
    opt = knap_search(n,0,Cap);
}
```