Cプログラミング演習1(再)　6

講義では、Cプログラミングの基本を学び
演習では、やや実践的なプログラミングを通して学ぶ

関数の呼び出し(選択ソート)

選択ソートのプログラム
(findMinValue, findAndReplace ができて
いるとする)

```
#include <stdio.h>
#define InFile "data.txt"
#define OutFile "sorted.txt"
#define NUM 1000

// ここにfindMinValueとfindAndReplace の関数
//  定義を書く

// 選択ソート : 配列arr[0]からarr[n-1]までをソートvoid selection
Sort(int arr[ ], int n) {
    int i;
    for ( i=0  ; i<n; i++)
        findAndReplace(arr,i,n);
}
```

```
int main(void)
{
   int  i, n=0, arr[NUM];
   FILE *fp;
   // ファイルを読み込み配列arrに記憶
   fp = fopen(InFile,"r");
   while (fscanf(fp,"%d", &arr[n]) != EOF) {
         n++;
   }
   fclose(fp);
   // n個の要素を持つ配列をソート
   selectionSort(arr,  n);
   // arr の中身を ファイルに書出す
   fp = fopen(OutFile,"w");
   for (i=0; i<n; i++) {
      fprintf(fp, "%d¥n", array[i]);
   }
   fclose(fp);
   return 0;
}
```

今までの復習

プログラムで最低限必要なもの

- 入力（キーボードから、ファイルから）
- 出力（画面へ、ファイルへ）
- 条件分岐：条件の成立・不成立により、異なる動作をする
- 繰り返し: 一定の回数の繰返し、条件成立の間の繰返し
- 関数の定義、関数の呼び出し

Cではそれ以外に、ポインタ、データ構造（配列や自作の構造）、ライブラリの利用、（C++ではクラス）が必要

アルゴリズム

アルゴリズムとは

料理で言えばレシピ

要するに、処理の段取り

どのような処理をどのような順で行うか

アルゴリズムによって処理効率に大きな違いがあることも

クイックソート(Quick sort)

クイックソート:関数名 quickSort

入力: 数を要素とする配列 array、先頭の要素のインデックス(int) *f*

　　最後の要素のインデックス(int) *l*

出力: 昇順にソートされた配列 (arrayの中身を書き換えるので不要)

手順:

1) *f* >= *l* ならば終了

2) m =array[(*f*+*l*)/2], i=*f*, j=*l* とする

以下を繰り返す(無限ループ)

3) j < i の間以下を繰り返す

(1) array[i] < m かつ i < *l* の間 i=i+1を繰り返す

(2) m < array[j] かつ j > *f* の間 j=j-1を繰り返す

(3) j <= i ならば 繰り返し終了

さもなくば array[i]とarray[j]をswap (値を取り替え)し、i=i+1 と j=j-1を行う

i=i+1　および j=j-1を行う

4) quickSort(array, *f*,j) と quickSort(array, j+1,*l*) を行って終了

疑問:
これで本当にソートを実現できるのか？

まずは手で動作を確かめよう

配列 を {3} や、{2,0,-5} や {2, 3, -5, 0}
として、どのようにこのアルゴリズムが動くか
紙に書いて確かめてみよう

それで納得できたら、プログラムにしてみよう

アルゴリズムからプログラムへ

ソートを例題として、ある書式で書かれたアルゴリズムをどのようにプログラムにするか、学ぼう

クイックソート:関数名 quickSort

入力: 数を要素とする配列 array、先頭の要素のインデックス(int) *f* と、最後の要素のインデックス(int) *l*

出力: 昇順にソートされた配列 (array) --- void

手順:

1) *f* >= *l* ならば終了

2) m =array[(*f*+*l*)/2]、i=*f*, j=*l* とする

3) 以下を繰り返す(無限ループ)

(1) array[i] < m かつ i < *l* の間 i=i+1を繰り返す

(2) array[j] > m かつ j > *f* の間 j=j-1 を繰り返す

(3) j <= i ならば 繰り返し終了
さもなくば array[i]とarray[j]をswap (値を取り替え)し、 i=i+1 と j=j-1を行う

4) quickSort(*array*,*f*,,j) と quickSort(array, j+1,*l*) を行って終了

アルゴリズムの名称≒関数名

関数の引数

関数の返り値

関数本体(ブロックの中身)

アルゴリズムからプログラムへ

クイックソート:関数名 quickSort

入力: 数を要素とする配列 array、先頭の要素のインデックス(int) *f* と、最後の要素のインデックス(int) *l*

出力: 昇順にソートされた配列 (array) --- void

手順:

1) *f* >= *l* ならば終了

2) m =array[(*f*+*l*)/2]、i=*f*, j=*l* とする

3) 以下を繰り返す(無限ループ)

(1) array[i] < m かつ i < *l* の間 i=i+1を繰り返す

(2) array[j] > m かつ j > *f* の間 j=j-1 を繰り返す

(3) j <= i ならば繰り返し終了
さもなくば array[i]とarray[j]をswap (値を取り替え)し、 i=i+1 と j=j-1を行う

4) quickSort(*array,f,*,j) と quickSort(array, j+1,*l*) を行って終了

条件分岐

繰り返し --- while? for?

繰り返し --- while? for?

swap(変数の値の交換)

返り値

アルゴリズムの名称≒関数名

関数の引数

関数の返り値

関数本体(ブロックの中身)

順番通りにコードにする

アルゴリズムからプログラムへ

クイックソート:関数名 quickSort

入力: 数を要素とする配列 array、先頭の要素のインデックス(int) *f* と、最後の要素のインデックス(int) *l*

出力: 昇順にソートされた配列 (array) --- void

手順:

1) *f* >= *l* ならば終了

2) m =array[(*f*+*l*)/2]、i=*f*, j=*l* とする

3) 以下を繰り返す(無限ループ)

(1) array[i] < m かつ i < *l* の間 i=i+1を繰り返す

(2) array[j] > m かつ j > *f* の間 j=j-1 を繰り返す

(3) j <= i ならば 繰り返し終了
さもなくば array[i]とarray[j]をswap (値を取り替え)し、 i=i+1 と j=j-1を行う

4) quickSort(*array*,*f*,,j) と quickSort(array, j+1,*l*) を行って終了

```
void quickSort(int array[], int f, int l) {
```

関数名

関数の引数

(文字列を含む)配列を戻り値とする場合、
(ポインタを学んでいないため)
void を関数の型とする

インデックスは必ず整数(int)でなければならない

```
}
```

アルゴリズムからプログラムへ

クイックソート:関数名 quickSort

入力: 数を要素とする配列 array、先頭の要素のインデックス(int) *f* と、最後の要素のインデックス(int) *l*

出力: 昇順にソートされた配列 (array) --- void

手順:

1) *f* >= *l* ならば終了

2) m =array[(*f*+*l*)/2]、i=*f*, j=*l* とする

3) 以下を繰り返す(無限ループ)

(1) array[i] < m かつ i < *l* の間 i=i+1を繰り返す

(2) array[j] > m かつ j > *f* の間 j=j-1 を繰り返す

(3) j <= i ならば繰り返し終了
さもなくば array[i]とarray[j]をswap (値を取り替え)し、 i=i+1 と j=j-1を行う

4) quickSort(*array*,*f*,,j) と quickSort(array, j+1,*l*) を行って終了

```
void quickSort(int array[], int f, int l) {
    if (f >= l)  return;
}
```

関数

関数の引数

ここには「変数宣言」が入る
この関数で用いる(引数以外の)
変数の型と変数名を書く

アルゴリズムからプログラムへ

クイックソート:関数名 quickSort

入力: 数を要素とする配列 array、先頭の要素のインデックス(int) *f* と、最後の要素のインデックス(int) *l*

出力: 昇順にソートされた配列 (array) --- void

手順:

1) *f* >= *l* ならば終了

2) m =array[(*f*+*l*)/2]、i=*f*, j=*l* とする

3) 以下を繰り返す(無限ループ)

(1) array[i] < m かつ i < *l* の間 i=i+1を繰り返す

(2) array[j] > m かつ j > *f* の間 j=j-1 を繰り返す

(3) j <= i ならば繰り返し終了
さもなくば array[i]とarray[j]をswap (値を取り替え)し、 i=i+1 と j=j-1を行う

4) quickSort(*array*,*f*,,j) と quickSort(array, j+1,*l*) を行って終了

```
void quickSort(int array[], int f, int l) {
  int m;
  int i ,j, temp;

  m = array[(f+l)/2];  i=f; j=l;

}
```

関数

関数の引数

アルゴリズムからプログラムへ

クイックソート:関数名 quickSort

入力: 数を要素とする配列 array、先頭の要素のインデックス(int) *f* と、最後の要素のインデックス(int) *l*

出力: 昇順にソートされた配列 (array) --- void

手順:

1) *f* >= *l* ならば終了

2) m =array[(*f*+*l*)/2]、i=*f*, j=*l* とする

3) 以下を繰り返す(無限ループ)

(1) array[i] < m かつ i < *l* の間 i=i+1を繰り返す

(2) array[j] > m かつ j > *f* の間 j=j-1 を繰り返す

(3) j <= i ならば 繰り返し終了
さもなくば array[i]とarray[j]をswap (値を取り替え)し、 i=i+1 と j=j-1を行う

4) quickSort(*array*,*f*,,j) と quickSort(array, j+1,*l*) を行って終了

```
void quickSort(int array[], int f, int l) {
  int m;
  int i ,j, temp;

  while (1) {

  }
}
```

関数

関数の引数

```
while(1) {
    ….
}
```

は無限繰返しのパタン

アルゴリズムからプログラムへ

クイックソート:関数名 quickSort

入力: 数を要素とする配列 array、先頭の要素のインデックス(int) *f* と、最後の要素のインデックス(int) *l*

出力: 昇順にソートされた配列 (array) --- void

手順:

1) *f* >= *l* ならば終了

2) m =array[(*f*+*l*)/2]、i=*f*, j=*l* とする

3) 以下を繰り返す(無限ループ)

(1) array[i] < m かつ i < *l* の間 i=i+1を繰り返す

(2) array[j] > m かつ j > *f* の間 j=j-1 を繰り返す

(3) j <= i ならば繰り返し終了
さもなくば array[i]とarray[j]をswap (値を取り替え)し、 i=i+1 と j=j-1を行う

4) quickSort(*array*,*f*,,j) と quickSort(array, j+1,*l*) を行って終了

関数

関数の引数

```
void quickSort(int array[], int f, int l) {
  int m;
  int i ,j, temp;

  while (1) {
    while ((array[i] < m) && (i < l)) i++;
    while ((array[j] > m) && (j > f)) j--;

  }
}
```

アルゴリズムからプログラムへ

クイックソート:関数名 quickSort

入力: 数を要素とする配列 array、先頭の要素のインデックス(int) *f* と、最後の要素のインデックス(int) *l*

出力: 昇順にソートされた配列 (array) --- void

手順:

1) *f* >= *l* ならば終了

2) m =array[(*f*+*l*)/2]、i=*f*, j=*l* とする

3) 以下を繰り返す(無限ループ)

(1) array[i] < m かつ i < *l* の間 i=i+1を繰り返す

(2) array[j] > m かつ j > *f* の間 j=j-1 を繰り返す

(3) j <= i ならば繰り返し終了
さもなくば array[i]とarray[j]をswap (値を取り替え)し、 i=i+1 と j=j-1を行う

4) quickSort(*array,f,*,j) と quickSort(array, j+1,*l*) を行って終了

```
void quickSort(int array[], int f, int l) {
  int m;
  int i ,j, temp;

  while (1) {

    if (j <= i) break;
    temp = array[i]; array[i] = array[j];
    array[j]=temp; i++; j--;
  }

}
```

関数

関数の引数

break によって
繰り返しから抜ける

アルゴリズムからプログラムへ

クイックソート:関数名 quickSort

入力: 数を要素とする配列 array、先頭の要素のインデックス(int) *f* と、最後の要素のインデックス(int) *l*

出力: 昇順にソートされた配列 (array) --- void

手順:

1) *f* >= *l* ならば終了

2) m =array[(*f*+*l*)/2]、i=*f*, j=*l* とする

3) 以下を繰り返す(無限ループ)

(1) array[i] < m かつ i < *l* の間 i=i+1を繰り返す

(2) array[j] > m かつ j > *f* の間 j=j-1 を繰り返す

(3) j <= i ならば 繰り返し終了
さもなくば array[i]とarray[j]をswap (値を取り替え)し、 i=i+1 と j=j-1を行う

4) quickSort(*array*,*f*,,j) と quickSort(array, j+1,*l*) を行って終了

```
void quickSort(int array[], int f, int l) {
  int m;
  int i ,j, temp;
  while (1) {
  }
  quickSort(array, f, j);
  quickSort(array, j+1, l);
  return ;
}
```

関数

関数の引数

最後のreturnはなくてもよい

Quick sort関数の完成

関数の呼び出し(クイックソート)

先の quickSort 関数が書けたならば、次のようにして全体のプログラムを完成させる:

```
#include <stdio.h>
#define InFile "data.txt"
#define OutFile "sorted.txt"
#define NUM 1000

// ここにquickSort関数定義を書く

int main(void)
{
   int  i, n=0, arr[NUM];
   FILE *fp;
   // ファイルを読み込み配列arrに記憶
   fp = fopen(InFile,"r");
   while (fscanf(fp,"%d", &arr[n]) != EOF) {
        n++;
   }
   fclose(fp);
   // arr[0]からarr[n-1]までをソート
   quickSort(arr, 0, n-1);
   // arr の中身を ファイルに書出す
   fp = fopen(OutFile,"w");
   for (i=0; i<n; i++) {
      fprintf(fp, "%d¥n", arr[i]);
   }
   fclose(fp);
   return 0;
}
```

実行時間計測

```
#include <stdio.h>
#include <time.h>

int main(void)
{
 clock_t start, end;


 start = clock();
 /*
      ここに処理すべきものを書く
 */
 end = clock();
 printf( "処理時間:%d[ms]\n", end - start );
```

時間のためのヘッダファイル

時間の記憶用の型と変数

処理前の時点の時間(プロセス時間)を記憶

処理後の時点の時間(プロセス時間)を記憶

この差が処理時間(ms単位)

時間計測:2つの方式の比較

timeSelectionSort.c --- 選択ソートの処理時間の計測(80000個のデータのソート)

白井のPCでは 2012ms

timequickSort.c --- クイックソートの処理時間の計測(80000個のデータのソート)

白井のPCでは 250ms (ただし100回分)

したがって80000個のデータの場合、2000 : 2.5 ≒ 800:1

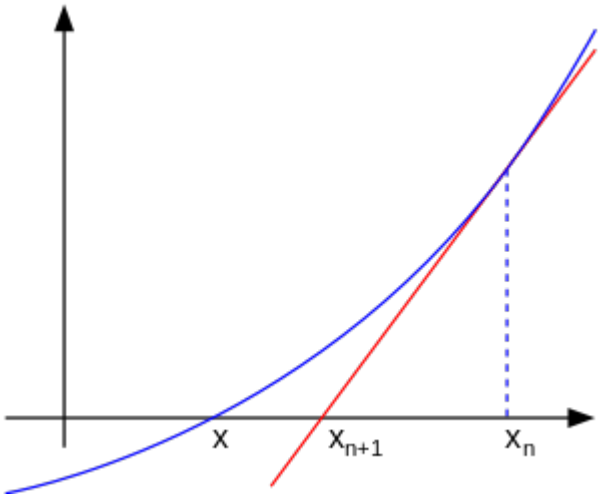
アルゴリズムによる実行時間の違い

方程式の根を求める(1)

変数xの関数$f(x)$に対し、$f(x)=0$となるxの値を求めることを考える。ここで、関数$f(x)$は微分可能とする。$f(x)$が微分可能であるとは、$f'(x)$が存在することとする。

このとき、xの近傍に適当な値 x_0 をとり、次の漸化式によりxに収束する数列を得ることができる場合が多い。

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



WikiPediaから引用

問題: これはなんという手法か？

方程式の根を求める(1)

方程式の根を求めるアルゴリズム
入力: 関数$f(x)$、その一階微分$df(x)$、
　　解にできるだけ近い値 x_0、閾値 th、
　繰り返し回数制限 $imax$
出力: $f(x)$=0となる根の近似値
手順:
1) x=x_0, i=0とする
2) i <= $imax$ ならば以下を繰り返す
　(1) $i = i$+1 とする
　(2) $y = x - f(x)/df(x)$
　(3) $(y - x)/x$ の絶対値が閾値th以下ならyを返り値として終了
　(4) $x = y$ とする
3)「収束しない」として異常終了

注意: Cでは関数を引数にする関数は作れないので、関数の引数リストから外しておく

課題1. 関数名を solveとして以下を考えよ(適切なものにせよ)
(1) 引数リストは?
(2) 返り値は?

課題2.「手順」をコード化せよ。
ただし収束しない場合は、その旨表示してから、0.0を返り値とするものとする

変数と関数の型に注意せよ

方程式の根を求める(1)

課題3: solve関数のプログラムができたら、次のコードで試してみよ

```
#include <stdio.h>
#include <math.h>
double f(double x) {
      // x^2 – 3
      return (x*x – 3.0);
}

double  df(double x) {
     // 2*x
     return(2.0*x);
}
```

```
// ここに作成した solve関数を
// 書き込む

int main(void) {
    double  ans, th=1.0e-8, x0 = 1.0;
    int imax = 100;
   ans = solve(x0,th,imax);
   printf("方程式の根=%10.3f\n",ans);
   return(0);
}
```

方程式の根を求める(1)

先のプログラムが予想通り動いたならば、いろいろな方程式の根を求めてみよ(つまり、適切な f と df 関数を与える)

例: (1) $f(x) = x^5+3x^2 - 9x - 10$

(2) $f(x) = 3\sin x+8\cos 3x+1$

(3) $f(x) = (2x+3)\log x - 3$

方程式の根を求める(2)

今の方法よりも効率は劣るが、連続な関数fに対して　$f(x)=0$　の近似解を求める方法がある

$f(a) * f(b) < 0$ となる適当なaとbという2つの値を初期値とする

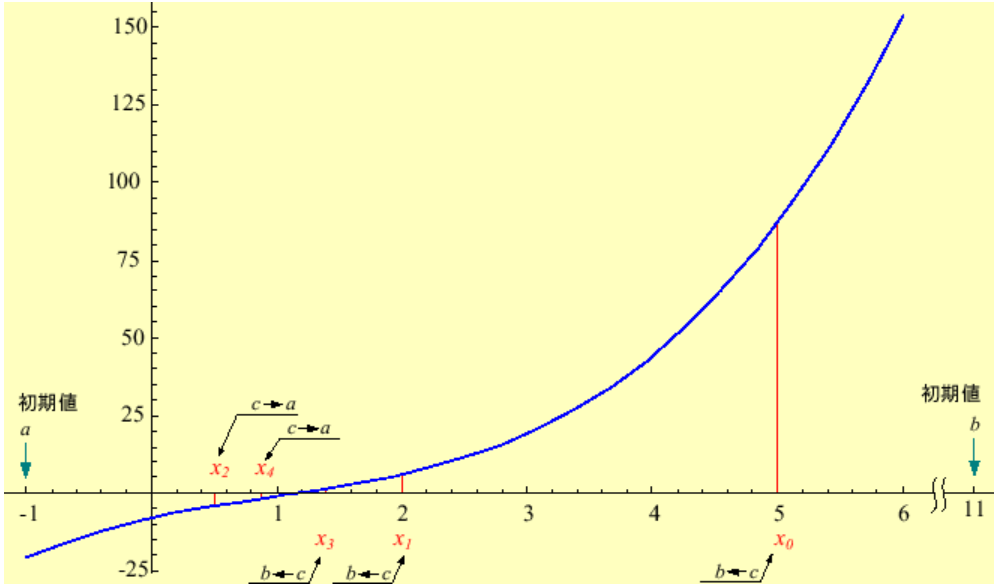
ここでは簡単のため、

$f(a) < 0, f(b) > 0$と仮定する

ある閾値thに対し $|a - b| > th$ である間以下を繰り返す:

$c=(a+b)/2$

f(c) < 0 ならばa=c, さもなくばb=cとする



問題: これはなんという手法か？

方程式の根を求める(2)

方程式の根を求めるアルゴリズム
入力: 関数$f(x)$ 、$f(a) < 0$、$f(b)>0$となる初期値$a, b,$ 閾値 th
出力: $f(x)=0$となる根の近似値
手順:
1) $|a-b|>th$ならば以下を繰り返す
  (1) $c=(a+b)/2$とする
  (2) $f(c) < 0$ならば $a=c$
    さもなくば $b=c$
2) c を返す

注意: Cでは関数を引数にする関数は作れないので、関数の引数リストから外しておく

課題4. 関数名を searchとして以下を考えよ(適切なものにせよ)
(1) 引数リストは?
(2) 返り値は?

課題5.「手順」をコード化せよ。
なぜ、繰り返しの回数制限をしていないのだろうか?

変数と関数の型に注意せよ

方程式の根を求める(2)

課題6: search関数のプログラムができたら、次のコードで試してみよ

```
#include <stdio.h>
#include <math.h>
double f(double x) {
      // x*x – 3
      return (x*x – 3.0);
}

double  df(double x) {
     // 2*x
     return(2.0*x);
}
```

```
// ここに作成した search関数を
// 書き込む

int main(void) {
   double  ans, th=1.0e-5, a=1.0. b=2.0;
   ans = solve(a, b,th);
   printf("方程式の根=%10.3f\n",ans);
   return(0);
}
```

方程式の根を求める(2)

先のプログラムが予想通り動いたならば、いろいろな方程式の根を求めてみよ。そして方程式の根を求める方法(1)による解と比較してみよ

例: (1) $f(x) = x^5+3x^2 - 9x - 10$

(2) $f(x) = 3\sin x+8\cos 3x+1$

(3) $f(x) = (2x+3)\log x - 3$