電子 1(木曜クラス)

第 11 回課題 T11

(ポインタ:教科書第 14 章、2005/06/30(Thu.))

基本問題

T11-1:配列の引数

(本提出期限 2005/06/30(Thu.)17:40、再提出期限 2005/07/14(Thu.)14:30)

提出物:Makefile、ソースファイル (arg_array.c)、入力ファイル (arg_array.in)、出力ファイル (arg_array.out)

他の関数内の 1 次元配列に標準入力から値を読み込む関数および、他の関数内の 1 次元配列の値を標準出力に出力する関数を作成せよ。これらの関数を利用して、標準入力からデータを読み込み、標準出力へデータを出力するプログラムを作成せよ。ただし、作成するプログラムは以下の仕様を満足させること。

(1) データを読み込む配列は `main` 関数のローカル変数とすること。また、読み込むデータ数は `main` 関数内で標準入力から読み込むこと。

(2) (1) の配列には最高 100 個までのデータを保存できるようにすること。この値はプログラムの先頭でマクロ定義し、マクロ定義の部分だけを書き換えるだけで変更できるようにすること。

(3) 関数は別に示すプロトタイプ宣言を持つようにすること。

(4) グローバル変数を用いないこと。

```
/* マクロ定義 */
#define N_MAX (100)  /* 一次元配列の最大の要素数*/

/* プロトタイプ宣言*/

/*
 * 配列に標準入力から指定個数分値を読み込む関数
 *
 * 引数 :
 * n        :読み込む要素数
 * array  : 値を読み込む 1 次元配列 (の先頭要素のアドレス)
 *          配列の要素は double 型の値
 *
 * 戻り値  : なし
 */
void scan_array(int n, double array[N_MAX]);

/*
 * 配列の指定個数分の内容を標準出力へ出力する関数
 *
 * 引数 :
```

```
 * n        : 出力する要素数
 * array    : 出力する内容を保持している配列 (の先頭要素のアドレス)
 *            配列要素は double 型の値
 *
 * 戻り値   : なし
 */
void print_array(int n, double array[N_MAX]);
```

実行例 1

```
~/T11/1$./arg_array
読み込む要素数は?
n=?
5

配列に 5 個の double 値を読み込みます。
10.4
2.3
-4.3
7.1
6.0

配列の 5 個の double 値を表示します。
10.4
2.3
-4.3
7.1
6.0

~/T11/1$
```

実行例 2

```
~/T11/1$./arg_array < arg_array.in
読み込む要素数は?
n=?

配列に 5 個の double 値を読み込みます。

配列の 5 個の double 値を表示します。
10.4
2.3
-4.3
7.1
6.0

~/T11/1$
```

arg_array.in 例

```
5

10.4
2.3
-4.3
7.1
6.0
```

応用問題

T11-2:ベクトルの足し算
(本提出期限 2005/07/07(Thu.)14:30、再提出期限 2005/07/14(Thu.)14:30)

提出物：Makefile、ソースファイル (add_vector.c)、入力ファイル (add_vector.in)、出力ファイル (add_vector.out)
　ベクトルの和を計算する関数を作成し、その関数を用いてベクトルの和を計算せよ。ただし、作成するプログラムは以下の仕様を満足させること。

(1) 全てのベクトルは、main 関数のローカル変数 (配列) とすること。また、ベクトルの次元 (読み込むデータ数) は main 関数内で標準入力から読み込むこと。

(2) ベクトルの各要素は、main 以外の関数を用いて、標準入力から読み込むようにせよ。(基本問題と同様に行なうこと。) また、出力も、main 以外の関数を用いて、標準出力へ出力するようにせよ。(基本問題と同様に行なうこと。)

(3) ベクトルは 100 次のものまで扱えるようにすること。この最高次数はプログラムの先頭でマクロ定義し、マクロ定義の部分だけを書き換えるだけで変更できるようにすること。

(4) 関数は別に示すプロトタイプ宣言を持つようにすること。

(5) グローバル変数を用いないこと。

```
/*
 * ベクトルの和演算を行う関数
 *
 * 引数 :
 * n         : ベクトルの次数
 * operand1 : 被演算ベクトル (和演算の左辺)(n 次ベクトル)
 * operand2 : 被演算ベクトル (和演算の右辺)(n 次ベクトル)
 * result    : 和演算結果を格納する場所 (n 次ベクトル)
 *
 * 戻り値   : なし
 */
int add(int n,
  double operand1[N_MAX], double operand2[N_MAX],
  double result[N_MAX]);
```

実行例 1

```
~/T11/2$./add_vector
ベクトルの次数は？
n=?
3

3 次ベクトルに double 値を設定します。
10.4
2.3
-4.3

3 次ベクトルに double 値を設定します。
2.2
-1.5
-3.0

ベクトルの和を計算中。

3 次ベクトルを表示します。
  12.6
   0.8
  -7.3
~/T11/2$
```

実行例 2

```
~/T11/2$./add_vector < add_vector.in
ベクトルの次数は？
n=?

3 次ベクトルに double 値を設定します。

3 次ベクトルに double 値を設定します。

ベクトルの和を計算中。

3 次ベクトルを表示します。
  12.6
   0.8
  -7.3
~/T11/2$
```

arg_array.in 例

```
3

10.4
2.3
-4.3

2.2
-1.5
-3.0
```