サンプルコード、解答集のソースコードについての変更と訂正

変更について

main関数の型にはintを指定する

main関数の型にはintを指定しましょう。

以下のソースコードはmain関数の型にint型を指定し忘れたまま掲載しています。

List5_1.c	List5_2.c	List5_5.c	List5.c
List6_1.c	List7_1.c	List6.c	List7.c
List10_1.c	List10_2.c	List10_3.c	List12_2.c
List16.c	List17.c	List18.c	List20.c
List21.c	List22.c		

以上のソースコードについては、以下に解説する理由により変更をした方がベターです。

本書に掲載するソースコードはK&Rによる『プログラミング言語C–ANSI規格準拠– 第2版』Brian W.Kernighan ・D.M.リッチー 著・石田 晴久訳の冒頭に掲載している次のコードをお手本にしています。

『プログラミング言語C』(P.7)に掲載されたhello.c

```
#include <stdio.h>

main()
{
    printf("hello, world\n");
}
```

これをgccでコンパイルすると、次のような警告が得られます。

gccでhello.cをコンパイルすると表示される警告

```
$ gcc hello.c
```

```
hello_world.c:3:1: warning: type specifier missing, defaults to 'int'
      [-Wimplicit-int]
main()
^
1 warning generated.
```

この警告は「型指定子が抜けています。デフォルトはintです」と書かれています。
型指定子とは「何かの型を指定するもの」で、ここではmain関数の型を指定するものが抜けているよ、という意味になります。

警告は、「こうした方が良いよ」というgccというC言語コンパイラ製作者からのアドバイスです。なるべくこの警告が表示されなくなるようにソースコードを変更しましょう。

ここでmain関数の型を指定する方が良いとされる最大の理由は、作成されたプログラムが正しく実行されたかどうかをOS側が把握できた方が良いからです。これに対してOSの上で実行しないプログラムもあります。OSそのものがその代表例です。そのほか、家電製品等に組み込む機器に書き込まれるプログラムは、OSを介せず直接機器を制御するものがあります。本書、そしてこの解説で取り扱っているのは、なにがしかのOS上で動かすプログラムに関するお話です。

OSを介せず、直接機器を制御する場合を「フリースタンディング環境（freestanding environment）」と呼び、OSを介して機器を制御する場合を「ホスト環境（hosted environment）」と呼びます。

本書の学習は、ホスト環境におけるC言語プログラミングです。ですから、main関数にはint型を指定するべきなのです。

hello.cをコンパイルした後で、次のシェルスクリプトcheck_aout.shを実行してください。

check_aout.sh

```
#!/bin/csh

if ( { ./a.out } ) then
  echo succeeded
else
  echo failed
endif
```

hello.cのコンパイルと実行結果のチェック

```
$ gcc hello.c
hello.c:3:1: warning: type specifier missing, defaults to 'int' [-
Wimplicit-int]
main()
^
1 warning generated.
```

```
$ csh check_aout.sh
hello, world
succeeded
```

hello.cは正しく実行されたように見えますが、実はmain関数が目的の処理を正しく実行できたかどうかはわからないのです。

ソースコードをhello2.cのように変更すれば、仮にですがmain関数が目的の処理を実行できたことを明確にできます。目的の処理を実行できたしるしとして、return文に0を与えます。

hello2.c

```c
#include <stdio.h>

int main()
{
    printf("hello, world\n");
    return 0;
}
```

hello2.cをコンパイルし、チェックスクリプトを実行すると次のようになります。

hello2.cのコンパイルと実行結果のチェック

```
$ gcc hello2.c
$ csh check_aout.sh
hello, world
succeeded
```

警告が表示されなくなりました。そして、実行結果も成功であるとわかります。

試しに、main関数が目的の処理を実行できなかった場合を見てみます。目的の処理を実行できなかった場合には、return文に0以外の整数値を与えます。

hello3.c

```c
#include <stdio.h>

int main()
{
    printf("hello, world\n");
    return 1;
}
```

hello3.cをコンパイルし、チェックスクリプトを実行すると次のようになります。

hello3.cのコンパイルと実行結果のチェック

```
$ gcc hello3.c
$ csh check_aout.sh
```

```
hello, world
failed
```

以上の理由から本書では、全くの初学者に配慮して、K&Rのhello.cそのままで学習を始めるのではなく、何かしらのOS上でC言語のソースコードを書いて実行する場合を想定し、main関数にはint型を指定した方が良いというスタンスを持っています。

main関数の引数にはvoidを指定する

main関数の引数にはhello4.cのように、voidを指定しましょう。

hello4.c

```c
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
    return 0;
}
```

引数voidはあってもなくてもコンパイルが問題なく成功します。

gccはmain関数に引数のvoidがあってもなくてもコンパイルを成功させます。これは標準C言語の規約で、引数を指定しなかった場合は引数はなんでも良いことになっているからです。

main関数は何か引数を受け取って、その引数に応じた処理を実行することができます。本書は初学者を読者対象としていますので、main関数に引数を与えて実行する、という学習を省いています。そこで、main関数の引数はありませんよ、と明示するためにvoidと指定しています。

引数のvoidを省略した方がキーボードを叩く回数が減って、学習者の負担を減らせます。しかし、工業高校の学生の多くが受験する全国工業高等学校校長協会主催の情報技術検定３級の問題では、hello4.cの形式を標準としていますので、これに準じました。

以上、本書に掲載するソースコードのmain関数の引数にvoidを指定する理由でした。

訂正について

List80.c

アンサーブックに掲載した解答例List80.cにおいて、絶対値を取得するabs関数を利用するにあたって組み込むライブラリstdlib.hが欠けていました。

List80.c

```c
#include <stdio.h>
#include <math.h>
int main(void)
{
    int data[5] = {45,80,75,28,95};
    int temp = 0;
    //get average
    float average = 0.0;
    for(int i = 0; i < 5; ++i){
        average += data[i];
    }
    average = average / 5;
    printf("average = %f\n",average);
    //sort
    for(int i = 0; i < 5 - 1; ++i){
       for(int j = i + 1; j < 5; ++j){
           if ( data[j] < data[i] ){
               temp = data[i];
               data[i] = data[j];
               data[j] = temp;
           }
       }
    }
    //sort result
    for(int i = 0; i < 5; ++i){
       printf("%4d",data[i]);
    }
    printf("\n");
    //find nearest
    int obj = -1;
    for(int i = 0; i < 5; ++i){
        if ( average < data[i] ) {
            obj = i;
            break;
        }
    }
    if ( abs(data[obj]-average) < abs(data[obj-1]-average) ) {
        // nearest is current object
    } else {
        // nearest is current -1
        obj = obj - 1;
    }
    printf("Nearest average mark is %d\n",data[obj]);

    return 0;
}
```

コンパイルすると、次のような警告が得られます。

List80.cをコンパイルした結果

```
$ gcc List80.c
List80.c:37:10: warning: implicitly declaring library function 'abs'
with type
      'int (int)' [-Wimplicit-function-declaration]
    if ( abs(data[obj]-average) < abs(data[obj-1]-average) ) {
         ^
List80.c:37:10: note: include the header <stdlib.h> or explicitly
provide a
      declaration for 'abs'
List80.c:37:10: warning: using integer absolute value function
'abs' when
      argument is of floating point type [-Wabsolute-value]
    if ( abs(data[obj]-average) < abs(data[obj-1]-average) ) {
         ^
List80.c:37:10: note: use function 'fabsf' instead
    if ( abs(data[obj]-average) < abs(data[obj-1]-average) ) {
         ^~~
         fabsf
List80.c:37:35: warning: using integer absolute value function
'abs' when
      argument is of floating point type [-Wabsolute-value]
    if ( abs(data[obj]-average) < abs(data[obj-1]-average) ) {
                                  ^
List80.c:37:35: note: use function 'fabsf' instead
    if ( abs(data[obj]-average) < abs(data[obj-1]-average) ) {
                                  ^~~
                                  fabsf
3 warnings generated.
```

警告の最初の行には次のように書かれています。「整数型で、引数も整数のabs関数は、暗黙に宣言されたライブラリ関数です。」次に、「abs関数が明確に宣言されているヘッダファイルを組み込んでください。」とあります。それ以下の警告に関する解説は省きます。

gccコンパイラは、abs関数だけでなく、いくつかの算術関数を内部に持っています。それらの関数をもつライブラリをソースコードに宣言していなくてもコンパイルできる場合があります。そのような関数をビルトイン関数と呼びます。ただし、利用できるビルトイン関数はコンパイラの種類やバージョンによって違いがありますので、今回のように警告で済んだから良しとするのではなく、適切なライブラリを宣言しておくべきです。ビルトイン関数は特殊な用途において実行速度を高速化するためのものですから、今回のような場合で何となく利用するべきではありません。

以上より、次のList80_fixed.cのようにstdlib.hをインクルードします。

List80_fixed.c

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
```

```c
int main(void)
{
    int data[5] = {45,80,75,28,95};
    int temp = 0;
    //get average
    float average = 0.0;
    for(int i = 0; i < 5; ++i){
        average += data[i];
    }
    average = average / 5;
    printf("average = %f\n",average);
    //sort
    for(int i = 0; i < 5 - 1; ++i){
       for(int j = i + 1; j < 5; ++j){
           if ( data[j] < data[i] ){
               temp = data[i];
               data[i] = data[j];
               data[j] = temp;
           }
       }
    }
    //sort result
    for(int i = 0; i < 5; ++i){
       printf("%4d",data[i]);
    }
    printf("\n");
    //find nearest
    int obj = -1;
    for(int i = 0; i < 5; ++i){
        if ( average < data[i] ) {
            obj = i;
            break;
        }
    }
    if ( abs(data[obj]-average) < abs(data[obj-1]-average) ) {
        // nearest is current object
    } else {
        // nearest is current -1
        obj = obj - 1;
    }
    printf("Nearest average mark is %d\n",data[obj]);

    return 0;
}
```

コンパイルすると、下のように２つの警告を得ます。

List80_fixed.cのコンパイル結果

```
$ gcc List80_fixed.c
```

```
List80_fixed.c:39:10: warning: using integer absolute value function
'abs' when
      argument is of floating point type [-Wabsolute-value]
    if ( abs(data[obj]-average) < abs(data[obj-1]-average) ) {
         ^
List80_fixed.c:39:10: note: use function 'fabsf' instead
    if ( abs(data[obj]-average) < abs(data[obj-1]-average) ) {
         ^~~
         fabsf
List80_fixed.c:39:35: warning: using integer absolute value
function 'abs' when
      argument is of floating point type [-Wabsolute-value]
    if ( abs(data[obj]-average) < abs(data[obj-1]-average) ) {
                                  ^
List80_fixed.c:39:35: note: use function 'fabsf' instead
    if ( abs(data[obj]-average) < abs(data[obj-1]-average) ) {
                                  ^~~
                                  fabsf
2 warnings generated.
```

最初の警告には次のように書かれています。「整数値の絶対値を返すabs関数に対して、引数に浮動小数点数が与えられています。」とあります。変数averageはプログラム中で計算した平均値が代入されるfloat型の変数です。これと配列dataに収められた整数値で減算をすると、その結果はfloat型になります。整数型を引数とするabs関数にfloat型の値を与えて絶対値に変換しようとしているために、警告しているのです。

このプログラムをabs関数の引数に対して厳密に実行すると、引数の型が一致していませんからエラーになるでしょうから、gccはコンパイルを通さず、実行はできないはずです。

しかし、このソースプログラムはコンパイル成功し、実行もできます。推察するに、コンパイラが気を利かせて、abs関数に与えた引数を、float型からint型の数値に変換してくれたのでしょう。

このような気遣いを、今後のgccにおいても許すかどうか不明ですから、gccからのアドバイスとしての警告ではfabsf関数を使うことを推奨しています。fabs関数は引数がdouble型、戻り値がdouble型です。今回は戻り値がdouble型で問題ないのですが、戻り値がfloat型になるfabsf関数を推奨しています。
せっかくですから警告のお勧めを採用して、fabsf関数を使用します。

以上より、適切な解答となるソースコードは次のList80_fixed2.cになります。

List80_fixed2.c

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main(void)
```

```
{
    int data[5] = {45,80,75,28,95};
    int temp = 0;
    //get average
    float average = 0.0;
    for(int i = 0; i < 5; ++i){
        average += data[i];
    }
    average = average / 5;
    printf("average = %f\n",average);
    //sort
    for(int i = 0; i < 5 - 1; ++i){
       for(int j = i + 1; j < 5; ++j){
           if ( data[j] < data[i] ){
               temp = data[i];
               data[i] = data[j];
               data[j] = temp;
           }
       }
    }
    //sort result
    for(int i = 0; i < 5; ++i){
       printf("%4d",data[i]);
    }
    printf("\n");
    //find nearest
    int obj = -1;
    for(int i = 0; i < 5; ++i){
        if ( average < data[i] ) {
            obj = i;
            break;
        }
    }
    if ( fabsf(data[obj]-average) < fabsf(data[obj-1]-average) ) {
        // nearest is current object
    } else {
        // nearest is current -1
        obj = obj - 1;
    }
    printf("Nearest average mark is %d\n",data[obj]);

    return 0;
}
```