2024年　6 月　13 日

デバッガーについて

Cのソースコードをコンパイルする時にエラーメッセージージを吐く場合には、メッセージを読めば間違えた行番号を知ることができますが、 Segmentation fault など実行時のエラーで終了する場合や、想定と異なる誤った結果が得られる場合には、問題のある場所を見つけるのは困難です。こうした場合、①printf文などを挿入してどこまで正常に動いているか、その時の変数の値を確かめる素朴な方法と、②デバッガを使用する方法があります。printf文を使う場合には計算機が効率を重視して、printf文の位置でただちに出力が行われないためにエラーの場所を見誤ることがあるので、printf文の直後に`fflush(stdout);` を挿入してprintfしたらすぐに標準出力されるようにしておきましょう。

デバッガを使う場合、まずソースコードを、

```
% gcc testprogram.c -g –o testprogram
```

のように **-g オプション**をつけてコンパイルしなおします。

続いて、

```
% gdb testprogram
```

として、デバッガーの上でプログラムを実行します。

gdbが起動したら、

```
 run
```

としてプログラムを実行し、異常終了したところで（まれにgdbでは正常終了してしまう場合もある）

```
 bt
```

と入力すると、プログラムが停止した行が示されます。btはbacktrace

この時の変数 i の値が知りたければ、

```
 print i
```

とすれば表示されます。（うまくいかないこともある）

注意が必要なのは、デバッグするプログラムが **testprogram　Leuco.fasta** といった**アーギュメントを取って実行する形式の場合**にはgdbの起動は同様に

```
gdb  testprogram
```

とgdbの起動後に

```
 run  Leuco.fasta
```

とrunに続けてアーギュメントを記すことです。

複数のアーギュメントを取る場合にも run の後に複数のアーギュメントを続けて記述します。

ubuntuにgdbがインストールされていない場合、 gdbと入力するとコマンドが見つからないと表示される場合

```
sudo apt update
sudo apt install gdb
```

として、gdbをインストールして下さい（ubuntuのパスワードが必要です）

tar（テープアーカイブ）とgzip（ファイル圧縮）について

今回の基本課題ではcity.cとcity.hという２つのファイルを提出していただきます。これらのファイルをバラバラに送るのではなく、いつものように名前をつけた 123Yamada10city というディレクトリにまず収めていただきます。(123とYamadaを自分の番号と名前で置き換え)

```
mkdir 123Yamada10city
mv city.? 123Yamada10city
```

このディレクトリをまず、tarコマンドによって一つのファイルに作り替えます

```
tar -cf 123Yamada10city.tar 123Yamada10city
```

最後のアーギュメントがアーカイブするディレクトリ名、その一つ前がファイルに変換された.tarファイルの名前です

できたtarファイルをgzipにより圧縮します

```
gzip 123Yamada10city.tar
```

gzipが終了すると、.gzというサフィックスのついた圧縮ファイル 123Yamada10city.tar.gzができているはずです。これをメールに添付します。受け取った側がこのファイルを元の２つのファイルが入ったディレクトリに戻すには

```
gunzip 123Yamada10city.tar.gz
tar -xf 123Yamada10city.tar.gz
```

とします。

今日の実習について

まず、ホームディレクトリの下にC_PROG/10というディレクトリを作ってその中で作業します。
C_PROGというディレクトリが既にあれば、以下のようにディレクトリを作成し、city_pop.datとcity_pop2.datの２つのファイルをダウンロードしコピーしておきます。

```
cd C_PROG
mkdir  10
cd 10
git  clone  https;//github.com/NGS-maps/Practice10
cp Practice10/* .
```

としておきます

これまでに、行なっている「実行時にアーギュメントを指定してプログラムの挙動を制御する」タイプのプログラミングについて復習します。UNIX系のシステムで作成するプログラムの多くは、何らかの形で入力データを読み込み、出力を書き出す形式を取ります。標準入出力で、キーボードから入力、ディスプレイへ出力、というインタラクティブな形式を取ることでパイプ、リダイレクトの機能を活用するスタイル（フィルタ型）もUNIXでは多く取られますが、一般的なアプリケーションを作る場合のプログラミングスタイルとしては、ファイルからの入力と、ファイルへの結果出力を行うことが多くなります。この場合、入出力のファイル名を、実行時にコマンド名の後に続けて入力できるようにすると、使いやすくなります。たとえば、more, less, cat などもコマンド名の後に開きたいファイル名が指定できることで汎用性のあるコマンドとなっています

コマンド名　入力データファイル　出力データファイル

このように、入力データと出力データのそれぞれのファイル名を実行時に読み込む場合、 main関数の引数としての argc の値はコマンド名も含めたアーギュメントの数として３となり、argv[0]にコマンド名、argv[1]に入力データファイル名、argv[2] に出力データファイル名が入ります。

```
int main(int argc,char *argv[])
{
char infile_name[128];                                  // 入力ファイル名 132文字を超えないと想定
char outfile_name[128];                                 // 出力ファイル名 132文字を超えないと想定

if(argc != 3)                                       // スペースで区切られたアーギュメントの数
 {                                                  //   プログラム名　入力ファイル名　出力ファイル名　として実行
 printf("Wrong number of argument %d\n”,argc);              //    上記の３つが揃っていなければエラーで終了
 printf(“Usage: cities input_file_name outfile_name\n”);
 return 1;
 }

strcpy(infile_name,argv[1]);                            // argv[1] を入力ファイル名にコピー
printf("Input File Name is %s\n",infile_name);
strcpy(outfile_name,argv[2]);                           // argv[2] を出力ファイル名にコピー
printf("Output File Name is %s\n",outfile_name);

FILE *infile;                                           // 入力ファイルハンドルを定義し
infile = fopen(infile_name,"r");                        // アーギュメントで取ったファイル名を読み込み”r”で開く
if(infile == NULL)                                      // ファイルが開けない場合
 {
 printf("File not found: %s\n",infile_name);            // 指示されたファイル名が存在しないと述べて、
 return 1;                                              // エラーを返す
 }
////////////////////////////////////////////////////////////////////////////
                    /////ここにファイル読み込みコードが入る
////////////////////////////////////////////////////////////////////////////
fclose(infile);      //読み終わったら閉じる

////////////////////////////////////////////////////////////////////////////
////////////////////////// 入力データを使った計算を行う//////////////////////////////
////////////////////////////////////////////////////////////////////////////

FILE *outfile;                                      // 出力ファイルを開くには、
outfile = fopen(outfile_name,"w");                      // アーギュメントで取ったファイル名を読み込み”w”で開く
if(outfile == NULL)                                     // ファイルが開けない場合
 {
 printf("File not found: %s\n",outfile_name);                   // 指示されたファイル名が存在しないと述べて、
 return 1;                                              // エラーを返す
 }
////////////////////////////////////////////////////////////////////////////
                    /////ここにファイル書き出しコードが入る
////////////////////////////////////////////////////////////////////////////
fclose(outfile);     //書き終わったら閉じる
return 0;
}
```

＃ファイルの入出力について

１行ずつファイルの内容を読み込むには

```
char buff[128];
while(fgets(buff,128,infile)
 {
   // 各行の入った文字列 buff を処理
 }
```

1文字ずつファイルの内容を読み込むには

```
char c;
while((c=fgetc(inpfile)) != EOF)
 {
   // 各文字の入った文字型変数 c を処理
 }
```

ファイルポインタを巻き戻すには

```
rewind(infile);
```

ファイルへ出力するには

```
fprintf(outfile,"%15s %8d %8.2f\n",cities[i].name,cities[i].population,cities[i].area);
```

printfをfprintfとして、カッコ内のアーギュメントの１つ目を、出力用のファイルハンドルにする。

構造体（Structure）

構造体は幾つかの異なるデータタイプの集合です。たとえば、以下のような市の名前と、人口、面積などの表となっているデータがあったとき、それぞれの市に対応するデータを格納する為の構造体を定義することでデータの構造化をはかることが出来ます。

cities.dat

```
-------------------------------------------------------------
City            Population      Area(km^2)
Maebashi        342678          311.64
Takasaki        371127          459.51
Saitama         1232577         217.49
Yokohama        3688624         437.38
Nara            364738          276.84
Ikoma           119258          53.18
-------------------------------------------------------------
```

上記のcities.dat をC_PROG/10に移動しておく

市の名前は文字列、人口は整数、面積は実数です。例えば、人口密度を計算して格納する場所も構造体の中に確保するとすると、実数の変数をもう一つとります。構造体の定義としては typedef文を用いると、変数宣言のときにその都度 structと書かなくてすむので便利です。

```
typedef struct city_info
 {
 char  name[20];                 // 市の名前　20文字を超えないと想定
 int   population;               // 人口
 float area;                     // 面積（平方キロメートル）
 float population_density;       // 人口密度
 } city_info;
```

大きなプログラムを作成する場合には、ソースコードを複数のファイルに分割するのが一般的です。プログラムが大きくなった場合、複数のソースコードに分割することを考えていくうえで、構造体の定義は、その構造体を使用する全てのソースで記述する必要があります。このためヘッダーファイルを別に作成して、その中で構造体定義を行い、構造体を使うすべてのソースファイルの中でそのヘッダーファイルをインクルードしておけば、構造体を変更する場合にもヘッダーファイルの１箇所だけを変更すればよいので、便利です。

構造体の定義を含む city.h というヘッダファイルを作る。

```
---- city.h -----------------
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct city_info
 {
 char  name[20];
 int   population;
 float area;
 float population_density;
 } city_info;
----------------------------
```

city.cという C のソースコードを記述するファイルを作る。

```
----- city.c --------------
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include “city.h” //上で定義したヘッダファイルをインクルード（同じディレクトリ内に置く）
int main(int argc, char *argv[])
  {
  int        num_city = 0;
  city_info   *cities;          // サイズで構造体をポインタ宣言をし malloc して記憶領域を確保
   int i;
   int  sum_population;
   float sum_area;
  ////////////////////////////////////////////////////////////////////////////////////
  ///////ここに入出力ファイル名の読み込み、ファイルを開き読み込み、データを処理、出力するコードを書く///
  ////////////////////////////////////////////////////////////////////////////////////
  }
----------------------------
```

ヘッダファイルをincludeするとヘッダ（.h）ファイルの中身がソースコード上のincludeした位置にそのまま展開される。

インクルードするヘッダファイルは、システムで定義されているstdio.hなどは <> で囲まれているが、自分で定義して同じディレクトリに置いておくヘッダファイルcity.h はダブルクォート""で囲まれていることに注意。

基本課題 city.c （必須：期限６月１６日正午、ただしできるだけ６月１４日中、
提出メールのタイトルは10山田_基本とする（名前は適宜変更）
提出先メールアドレスは （knakamura.maebit@gmail.com）
(OneDriveやGoogleDriveのファイルドロップ等にしない)

入力ファイル、city_pop.datまたはcity_pop2.datの全データを構造体配列を用いてメモリ上に読み込み、出力ファイルとして、**表の右端に各市の人口密度を記し**、また、**表の最後の行に、Totalの人口、面積、および全体の人口密度、を加えた表を作る**プログラムcity.h, city.cを作成し、###Yamada10cityと名前をつけたディレクトリに二つのファイルを格納し。tarおよびgzipしたアーカイブ圧縮ファイルをメール添付で提出する。ファイル名には日本語を入れないこと。エラーなど自分で解決できな問題点がある場合には、メールのタイトルに**「未解決」**と記し、本文に問題点を記述すること。

gcc -o city city.c

として、コンパイルし、出来た実行形式ファイル city に対して、

city city_pop.dat city_pop.out

のように入力と出力のファイル名をアーギュメントで指定して実行する形式とすること。

ヒント1:　　読み込みバッファ char buff[128]; を一つ定義し、

```
while(fgets(buff,128,infile))
   {
   // 1行ずつ処理
   }
```

として一行ずつ読み込む（**一行目は読み飛ばす**）

ヒント2: 各エントリー（市）のカウンタ int n_cities;を定義、１回目のスキャンでn_citiesをカウントし、
city_info 型の配列領域を mallocする。

```
city_info *cities;
cities=(#########)malloc(##################);
```

一旦 buff という長い文字列に１行を読み込んで、その行から、町の名前、人工・面積を抽出するために名前、人口、面積などの変数を sscanf関数で読み込む。

```
sscanf(buff,"%s %d %f",
     cities[n_cities].city_name,
     &cities[n_cities].population,&cities[n_cities].area);
```

ヒント3：最後に、人口密度を計算しながら、for文でn_cities回、まわしながらoutfileに結果を出力する

```
fprintf(outfile,"%18s %9d %9.3f %9.3f %9.3f",cities[i].city_name,.....);
```

ヒント4：全国の市の中の平均の人口密度は、各市の人口密度の平均とは異なります

出力イメージ　cities.out　　(###に適宜、数値がはいる)

```
------------------------------------------------------------
City            Population      Area(km^2)        Population density(km^-2)
Maebashi        342678          311.64            ####.##
Takasaki        371127          459.51            ####.##
Saitama         1232577         217.49            ####.##
Yokohama        3688624         437.38            ####.##
Nara            364738          276.84            ####.##
Ikoma           119258          53.18             ####.##

Total           ########        ###.###           ####.##
------------------------------------------------------------
```

応用課題　genes.c　（任意：期限６月１６日中、

提出メールのタイトルは10山田_応用とする（名前は適宜変更）

提出先メールアドレスは　(knakamura.maebit@gmail.com)

(OneDriveやGoogleDriveのファイルドロップ等にしない)

Ecoli_gene.txtというデータファイルはgff3フォーマットによる遺伝子領域記述ファイルの一部をawkにより抽出したものである。この内、空白文字区切りの３カラム目がCDSと記述された行のみについて、４カラム目と５カラム目に記された、CDS領域の開始点と終了点の整数値、及び、６カラム目の+または-で示されたCDSの方向に関する情報の３つの要素を記録する構造体を定義し、３カラム目がCDSの行数分この情報を記憶できる構造体の配列を確保し、情報を格納しなさい。ただし、３カラム目がregion, gene, repeat_region などCDS以外の行は読み飛ばして良い。

```
================================Ecoli_gene.txtのはじめの部分====================
NC_000913.3 RefSeq region 1 4641652 +
NC_000913.3 RefSeq gene 190 255 +
NC_000913.3 RefSeq CDS 190 255 +
NC_000913.3 RefSeq gene 337 2799 +
NC_000913.3 RefSeq CDS 337 2799 +
NC_000913.3 RefSeq gene 2801 3733 +
NC_000913.3 RefSeq CDS 2801 3733 +
NC_000913.3 RefSeq gene 3734 5020 +
NC_000913.3 RefSeq CDS 3734 5020 +
NC_000913.3 RefSeq gene 5234 5530 +
NC_000913.3 RefSeq CDS 5234 5530 +
NC_000913.3 RefSeq repeat_region 5566 5601 +
NC_000913.3 RefSeq repeat_region 5637 5670 -
NC_000913.3 RefSeq gene 5683 6459 -
NC_000913.3 RefSeq CDS 5683 6459 -
NC_000913.3 RefSeq gene 6529 7959 -
NC_000913.3 RefSeq CDS 6529 7959 -
NC_000913.3 RefSeq gene 8238 9191 +
NC_000913.3 RefSeq CDS 8238 9191 +
NC_000913.3 RefSeq gene 9306 9893 +
NC_000913.3 RefSeq CDS 9306 9893 +
NC_000913.3 RefSeq gene 9928 10494 -
NC_000913.3 RefSeq CDS 9928 10494 -
NC_000913.3 RefSeq gene 10643 11356 -
================================================================================
```

ここまでできたら、別にEcoli.fastaという塩基配列データを読み込み、それぞれの遺伝子の塩基配列をマルチファスタ形式で出力しなさい。このとき方向が - の行については相補逆鎖を出力する。例えば3行目にある1番目のCDS1につ

いては190塩基目から255塩基目までの６６塩基（かならず３の倍数になる）を出力し、15行目にあるCDS6については、5683塩基目から6459塩基目の777塩基の相補逆鎖を出力する。

genes.h, genes.c を記述し、

gcc -o genes genes.c

とコンパイルして

genes Ecoli.fasta Ecoli_gene.txt Ecoli_gene.fasta

のように実行するものとする
注： 入力ファイルは、１本の塩基配列が入ったシングルファスタを想定して良い。

==========出力例 Ecoli_gene.fasta==========

>CDS1_190+255
塩基配列（60字で改行）

>CDS2_337+2799
塩基配列

・・・・・・・・・・・・中略・・・・・・・・・・

>CDS6_5683-6459
塩基配列

・・・・・・・・・・・・後略・・・・・・・・・・

==========

提出するのは基本課題と同様tar.gzファイルで

```
mkdir 123Yamada10genes
cp genes.? 123Yamada10genes
tar -cf 123Yamada10genes.tar 123Yamada10genes
gzip 123Yamada10genes.tar
```

としてできた123Yamada10genes.tar.gzファイルをメール添付