4.4 デバッグどうしよう？

まずは、次のソースを見てもらおうか。

```
#include <stdio.h>

int main(void){
    int *x;
    *x = 1;
    printf(" %d",*x);
    return 0;
}
```

まぁ、うまく動かないよね[1]。実際、コンパイルして動かそうとしても次のとおりになる。

```
bash-3.2$ gcc test.c -o test
bash-3.2$ ./test
セグメンテーション違反です
bash-3.2$
```

この例はシンプルだが、実際にはもっと複雑なプログラムを作る。その際にプログラムが予想通りに動かなければ、当然直さねばなるまい。ひとつの方法は**printf**関数を埋め込んで、どこまでうまく動くのかをチェックする方法だ。が、実はうまく働かないことも多い[2]。そのときに使って欲しいのが**gdb**(GNU debugger)コマンドだ。

まずは、プログラムのコンパイル時にデバッグ情報を実行ファイルに組み込もう。

```
gcc test.c -o test -O0 -g3
```

-O0　は「最適化をしない」、-g3は「マクロを使えるようにする」という意味のオプションであり、必ず指定するようにしよう。これにより、出来上がった実行ファイルtestを使ってデバッグすることができる。

準備は万端だ。gdbを使おう。まずは次のコマンドを実行。

[1]どこがまずいかわかるかな？予想してみよう。
[2]特にセグメンテーション違反が起こるときなどがいい例。

```
gdb ./test
```

実行すると次のように表示される。

```
GNU gdb 6.7.1
Copyright (C) 2007 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later ...
This is free software: you are free to change and ...
There is NO WARRANTY, to the extent permitted by law.  ...
and "show warranty" for details.
This GDB was configured as "x86_64-vine-linux"...
Using host libthread_db library "/lib64/libthread_db.so.1".
(gdb)
```

この(gdb)がユーザーからのコマンドを受け付けているところである。コマンドr(=run)を入力してみよう。

```
(gdb) r
Starting program: /home/sit/masaomi/basic1b/example/test

Program received signal SIGSEGV, Segmentation fault.
0x0000000000400484 in main () at test.c:5
5          *x =1;
```

このなかで、次の部分はtest.cの5行目でセングメンテーション違反(Segmentation fault)が発生していることを示している[3]。

```
5          *x =1;
```

また、ポインタを使う前にはmallocなどを用いて領域を確保しておく必要があった[4]。これを含め、またこれに伴って必要となるstdlib.hをインクルードすると次のようになる。

[3]まぁ、そもそも“Program received signal SIGSEGV, Segmentation fault.... in main () at test.c:5”　の部分に書いてあることでもある。

[4]この知識はデバッグで必要かつ有用。理由はハマリポイントだから。

Table 4.1: (gdbの主なコマンド)

command	意味
break 関数名	プレークポイントを指定した関数に設定する。
run [引数リスト]	プログラムの実行を開始します(もしあれば引数リストを引数に渡す)。
bt	バックトレース: プログラムのスタックを表示します。
print 式	式の値を表示します。
next	次のプログラム行を実行します。その行内の全ての関数は 1 ステップで実行されます。
step	次のプログラム行を実行します。もしその行に関数が含まれていれば、その関数内をステップ実行していきます。

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

int main(void){
    int *x;
    x=(int *)malloc(sizeof(int));
    *x = 1;
    printf(" %d",*x);
    return 0;
}
```

これで正しいプログラムとなった。このようにプログラムに誤りがあり正しく動かないときに、どこに誤りがあるかを発見する際のヒントを得たいときに便利である。

ちなみに、gdbを抜けるときはq(=quit)を入力しよう。

```
(gdb) q
bash-3.2$
```

コマンドにはいろいろなものがある。表4.1は代表例である[5]。

[5]man gdbの出力からの抜粋（一部改変)。

使い方の詳細は以下の文献に譲ろう。
http://www.cabrillo.edu/s̃hodges/cs19/progs/guide_to_gdb_1.1.pdf

また、man gdbを実行してみよ。

4.5 ゲームの仕様(2)

先回はクイズゲームを実装した。今回は以下の機能を実装しよう。

- タイピングデータの読み込み
- タイピングの入力、時間管理、正誤判定

タイピングデータは以下のように定義される構造体DATAの配列で保持するものとする。この構造体は、ヘッダファイルに定義しておく(typing.hを見よ)。

```
struct _data{
  char string[50];
  int score;
};
typedef struct _data DATA;
```

加えて、配列の要素数MAXLENは100とする。

今回の開発対象となるタイピングゲームの機能は以下の2つの関数を使って実現する。なお、関数の引数にポインタ（配列[6]含む）を指定すると、参照渡しになることを思い出しながら関数を実装せよ。

- `int dataReader(DATA *dataArray, int *n)`

  ポインタ *dataArray* は読み取ったタイピングデータを格納する配列を表す。この引数にはmain関数で領域確保を行ったポインタが入るものとする。さらに、ポインタnは読み取ったタイピングデータの個数を返すために利用する。タイピングデータはtyping.txtというテキストファイルに記載されているものとする。フォーマットは

  ```
  MySQL,6
  Cloud,4
  PostgreSQL,7
  ```

[6]配列の要素にあらず

のようにタイプするべき文字列と点数がカンマ区切りで並んでいるものとする。これを *dataArray* に格納する。なお、fscanfを以下を参考にして利用すること[7]。

```
fscanf(fp," %[^,],%d", dataArray[i].string, &dataArray[i].score)
```

全データを読み取ったら、読み取ったタイピングデータの個数をポインタnを使って出力する。関数は、ファイルの読み込みに失敗したら-1を返し、成功したら0を返す。先回の quizReader関数を参考に作成するとよい。

- `int typing(DATA *dataArray, int n)`

  ポインタ *dataArray* はdataReader関数で読み取ったタイピングデータを格納する配列であり、変数nには[8]読み取ったタイピングデータの個数が入る。

  ユーザーは、最初、持ち時間として20秒持っているものとする。正解のスコアは最初、0点であるとする。

  dataArray のn個の要素からランダムに 1 つ選び[9]、stringメンバを画面に表示し、scanfを使ってユーザーの入力を取得する。表示された文字列と入力された文字列が等しければ[10]、持ち時間にscoreメンバの値が加えられ、scoreメンバの値がスコアにも加算される。等しくなければ持ち時間からscoreメンバの値が差し引かれる。さらに、等しい場合でも等しくない場合でも入力にかかった時間[11]が持ち時間が差し引かれる。もし持ち時間が0以上であれば「○○ seconds left. Your current score is ××」のように残り持ち時間、現在のスコアを表示する。持ち時間が0未満であれば、「TIME UP! Game over...」と画面に表示する。以上を、持ち時間が0未満になるまで繰り返す。

  最後に関数はスコアを戻り値として返す。

  C言語標準のtime.hに定義されているtime_t構造体およびtime関数、difftime関数を使って

[7] fscanfの書式フォーマット部分の先頭に半角スペースが入ることに注意。余計な改行文字を除くためである。

[8] 今度はポインタではないことに注意。

[9] rand関数を使え。

[10] strcmp関数を使うこと。

[11] stringメンバが表示されてからユーザーがエンターキーを押すまでの時間

```
double diff;
time_t starttime=time(NULL), endtime;
...
endtime=time(NULL);
diff=difftime(endtime,starttime);
```

とすると、最初にtime関数が呼び出されてから次にtime関数が呼び出されるまでの時間を計測することができることを使え。

ヘッダファイルとして、stdio.h、typing.h　だけでなく、string.h、time.hも忘れずにインクルードすること。

これらの関数を、typing.cというファイルに記述せよ。対応するヘッダファイル(typing.h)は以下のようになるはず。

```
#ifndef TYPING_H_
#define TYPING_H_

#define MAXLEN 100

struct _data{
  char string[50];
  int score;
};

typedef struct _data DATA;

int DataReader(DATA *dataArray, int *n);
int typing(DATA *dataArray, int n);

#endif  /* TYPING_H_ */
```

なお、以下のプログラム（ファイル名をtyping_driver.c、実行ファイル名をtypingとせよ）に上記のプログラムをリンクして実行せよ。

```
#include <stdlib.h>
#include <stdio.h>
#include "typing.h"

int main(void){
```

```
  srand((unsigned)time(NULL));

  DATA *dataArray=(DATA *)malloc(sizeof(DATA)*MAXLEN);
  int num=0,ret=0,score=0;

  ret=dataReader(dataArray,&num);
  printf("%d sentences were read.\n",num);

  if(ret==0){
    score=typing(dataArray,num);
    printf("You got score:%d\n",score);
  }

  free(dataArray);

}
```