第1回 GDB ゼミ

ゼミ担当者 ： 岩橋 崇史，米田 真純，長野 林太郎
指導院生 ： 實田 健
開催日 ： 2003 年 3 月 31 日

ゼミ内容: 本ゼミでは，バグの発見や修正を支援するソフトウェアの一つである GDB について述べる．
まず，GDB の概要を述べ，バグが存在するプログラムを用いて，GDB の使用方法について説明する．

1 GDB とは

GDB とは，デバッガと言われるプログラムの誤り (バグ) の発見や修正を支援するソフトウェアの１つである．Free Software Foundation(FSF) が開発しているフリーソフトであり，最新の GDB 5.3 では C,C++言語に対応している[1]．また，一部の機能制限はあるが，Fortran，Pascal にも対応している．GDB には大きく分けて 4 つの機能が備わっている．なお，GDB は逐次プログラムを対象としており，並列プログラムを対象としていない．

- プログラムの動作を詳細に指定して，プログラムを実行させる．
- 指定した条件でプログラムを停止させる．
- プログラムが止まった時に，何が起こったのかを調べる．
- バグによる副作用を修正し，別のバグを調べるためにプログラムの状態を変更する．

2 GDB の使用方法

本章では，Fig. 1 で示す既知のバグが存在するサンプルプログラム（hoge.c，calc.c）を用意し，これを使って，GDB の使用方法を説明する．Fig. 2 に，サンプルプログラムの実行結果を示す．

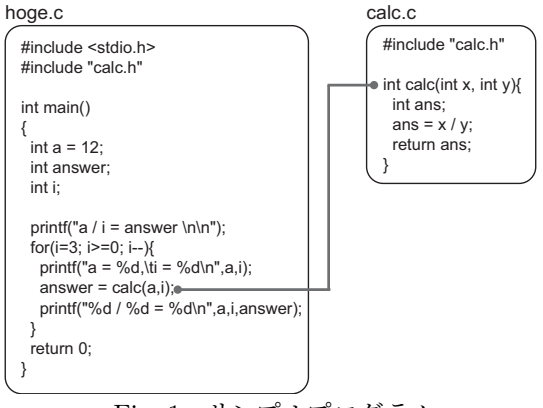

Fig. 1 サンプルプログラム

2.1 GDB のインストール

GDB を Debian にインストールするには，次のコマンドを実行すればよい．

```
# apt-get install gdb
```

[1]Java 言語の場合は jdb という Java 用のデバッガが存在する．jdb は Java の開発環境である JDK に標準で含まれているため，JDK をインストールすればすぐに利用可能である．コマンドは一部 GDB と異なるものが存在する

```
$ ./run_hoge
a / i = answer
a = 12, i = 3
12 / 3 = 4
a = 12, i = 2
12 / 2 = 6
a = 12, i = 1
12 / 1 = 12
a = 12, i = 0
Floating point exception (core dumped)
```

Fig. 2　実行結果

2.2　デバッグ情報の生成

プログラムを GDB を用いて実行するには，コンパイル時にデバッグ情報を生成する必要がある．このデバッグ情報は，オブジェクトファイルに格納され，その中に個々の変数や関数の型，ソースコード内の行番号と実行形式コードのアドレスの対応などが含まれる．デバッグ情報の生成を要求するには，コンパイラの実行時に-g オプションを指定する．

```
$ gcc hoge.c calc.c -g -o run_hoge
```

2.3　GDB の起動

GDB を起動するには，次のようにデバッグされる実行プログラム名を引数に指定する．これにより，実行ファイル run_hoge が GDB 制御下に置かれ，GDB のコマンド待ち状態になる

```
$ gdb run_hoge
[ messeage...]
(gdb)
```

2.4　実行

```
(gdb) run
```

とすることで，プログラムの実行を開始する．

2.5　続行

```
(gdb) cont
```

とすることで，ブレークポイントなどで一度停止しているプログラムを続行する．

2.6　終了

```
(gdb) quit
```

とすることで，GDB を終了する．

3　GDB の主要コマンド

プログラムのバグを見つけるために，有用な GDB のコマンドについて説明する．

3.1　ブレークポイント

GDB では，デバッグ作業を効率よく行うためにブレークポイントというものを設定することができる．ブレークポイントとは，プログラムの実行を強制的に一時停止される場所のことである．プログラム中のバグが潜んでいそうな場所にブレークポイントを設定し，プログラムの処理を中断し，それ以降のソースコードを 1 行ずつ実行し，バグの

問題個所を特定することができる．

ブレークポイントを設定するには break コマンドを使う．break コマンドの引数には，ソースファイル名：関数名もしくは行番号を指定する．なお，引数として関数名および行番号のみを指定した場合，対象となるソースファイルはメインソースファイルである．

```
(gdb) break ソースファイル名:関数名 or 行番号
```

補足として，C++のように引数の型および数が違うが同じ関数名の関数が複数存在するプログラムにおいて，ブレークポイントの設定方法を説明する．その場合は，引数として，関数名に加え，その引数の型を補足すればよい．

```
(gdb) break ソースファイル名:関数名 (引数の型，引数の型，…)
```

Fig. 3 に break コマンドを使ってバグを見つける例を示す． で示すとおり，main 関数にブレークポイントを指定する．そして，run コマンドにより，プログラムを実行する．すると で示すとおり，ブレークポイントの手前でプログラムの実行が一時停止する． と同様に のように，calc.c の calc 関数の前にもう一つブレークポイントを設定する．cont コマンドでプログラムの処理を続行していく後に， のようにバグを見つけることができた．

```
(gdb) break main …
Breakpoint 1 at 0x80483f6: file hoge.c, line 6.
(gdb) run
Breakpoint 1, main () at hoge.c:6 …
6         int a = 12;
(gdb) break calc …
Breakpoint 2 at 0x8048496: file calc.c, line 5.
(gdb) cont
Continuing.
a / i = answer
a = 12, i = 3
Breakpoint 2, calc (x=12, y=3) at calc.c:5
5         ans = x / y;

（中略）

(gdb) cont
Continuing.
12 / 1 = 12
a = 12, i = 0
Breakpoint 2, calc (x=12, y=0) at calc.c:5
5         ans = x / y;
(gdb) cont
Continuing.

Program received signal SIGFPE, Arithmetic exception. …
0x0804849d in calc (x=12, y=0) at calc.c:5
5         ans = x / y;
```

Fig. 3　GDB を用いたプログラムのバグ発見例

なお，ブレークポイントに関連したコマンドがあるので，それについて説明する．

- ブレークポイントの表示
  現在設定されているブレークポイントの状態を表示するには，次のようなコマンドを事項する．

```
(gdb) info break
Num Type           Disp Enb Address    What
1   breakpoint     keep y   0x080483f6 in main at hoge.c:6
        breakpoint already hit 1 time
2   breakpoint     keep y   0x08048496 in calc at calc.c:5
        breakpoint already hit 4 times
```

- ブレークポイントの削除
  特定のブレークポイントを削除する場合は，info break コマンドでブレークポイント番号を調べた上で，次のコマンドを実行する．

```
(gdb) delete 1
(gdb) info break
Num Type           Disp Enb Address    What
2   breakpoint     keep y   0x08048496 in calc at calc.c:5
        breakpoint already hit 4 times
```

なお，設定されている全てのブレークポイントを削除するには，次のコマンドを実行する．

```
(gdb) delete
```

3.2 list コマンド

list コマンドはソースファイルを表示するコマンドである．関数名を指定することにより，関数の中のソースを見ることも可能である．list コマンドでは 10 行ずつソースファイルが表示される．

```
(gdb) list
1       #include <stdio.h>
2       #include "calc.h"
3
4       int main()
5       {
6         int a = 12;
7         int answer;
8         int i;
9
10        printf("a / i = answer \n\n");
(gdb)
11        for(i=3; i>=0; i--){
12          printf("a = %d,\ti = %d\n",a,i);
13          answer = calc(a,i);
14          printf("%d / %d = %d\n",a,i,answer);
15          printf("\n");
16        }
17        return 0;
18      }
(gdb)
Line number 19 out of range; hoge.c has 18 lines.
(gdb)
```

Fig. 4　list コマンドの使用

なお，gdb では 1 つ前に入力したコマンドを覚えているため，同じコマンドを複数回実行する場合にはコマンドの省略が可能である．

```
(gdb) list calc
1       #include "calc.h"
2
3       int calc(int x, int y){
4         int ans;
5         ans = x / y;
6         return ans;
7       }
(gdb)
Line number 8 out of range; calc.c has 7 lines.
(gdb)
```

Fig. 5　list コマンドの使用（関数名 calc の指定）

3.3　print コマンド

print コマンドは変数の中身を表示するコマンドである．中身を表示したい変数名を print コマンドの引数に与える．なお，出力書式を指定することも可能である（2 進数，アドレスなど）．

```
(gdb) break 12
Breakpoint 1 at 0x8048480: file hoge.c, line 12.
(gdb) run
Starting program: /home/yoneda/gdb/gdb_program/run_hoge
a / i = answer


Breakpoint 1, main () at hoge.c:12
12          printf("a = %d,\ti = %d\n",a,i);
(gdb) print a
$1 = 12
(gdb) print i
$2 = 3
(gdb)
```

Fig. 6　print コマンドの使用

配列に対して print コマンドを使用した場合の実行例を以下に示す．Fig. 7 はサンプルプログラム，Fig. 8 は実行結果である．

```
1       #include<stdio.h>
2
3       int main(){
4         int i;
5         int a[10];
6
7         for(i=0;i<10;i++){
8           a[i]=i;
9           fprintf(stderr,"a[%d] = %d\n",i,a[i]);
10        }
11        return 0;
12      }
```

Fig. 7　サンプルプログラム

Fig. 8 のように配列の場合には，配列内の数字すべてが出力される．

3.4　next コマンドと step コマンド

next コマンドと step コマンドは，どちらもソースコードの一行分づつ実行させる命令である．コマンド待ち受け状態で next あるいは step コマンドを入力すると，停止するときに表示された一行分が実行される．

next コマンドと step コマンドは，関数の取り扱いが異なる．next コマンドでは，次に実行する行内に関数があっ

```
(gdb) break 9
Breakpoint 1 at 0x804849f: file test.c, line 9.
(gdb) run
Starting program: /home/yoneda/gdb/Hoge/test1_run

Breakpoint 1, main () at test.c:9
9           fprintf(stderr,"a[%d] = %d\n",i,a[i]);
(gdb) print a
$1 = {0, 134518128, 1, 134513249, 1073926332, 1075062976, -1073743172, 1074965022, -1073743272,
  1073796736}
       <中略>
(gdb) print a
$2 = {0, 1, 2, 3, 4, 5, -1073743172, 1074965022, -1073743272, 1073796736}
       <中略>
(gdb) print a
$3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Fig. 8　配列に対する print コマンドの使用

た場合，関数の実行も含めて一行分の実行を行う．それに対して step コマンドでは，関数サブルーチン内の一行分だけ実行する．

```
hoge.c
1       #include <stdio.h>
2       #include "calc.h"
3
4       int main()
5       {
6         int a = 12;
7         int answer;
8         int i;
9
10        printf("a / i = answer \n\n");
11        for(i=3; i>=0; i--){
12          printf("a = %d,\ti = %d\n",a,i);
13          answer = calc(a,i);
14          printf("%d / %d = %d\n",a,i,answer);
15          printf("\n");
16        }
17        return 0;
18      }
```

```
calc.c
1       #include "calc.h"
2
3       int calc(int x, int y){
4         int ans;
5         ans = x / y;
6         return ans;
7       }
```

Fig. 9　ソースファイル

Fig. 10 と Fig. 11 を見ると，step コマンドでは関数 calc のサブルーチン内を一行実行しているが，next コマンドでは関数 calc を一行のプログラムとして実行している．この点が next コマンドと step コマンドの違いである．

```
(gdb) break 13
Breakpoint 1 at 0x8048496: file hoge.c, line 13.
(gdb) run
Starting program: /home/yoneda/gdb/gdb_program/run_hoge
a / i = answer

a = 12, i = 3

Breakpoint 1, main () at hoge.c:13
13          answer = calc(a,i);
(gdb) next
14          printf("%d / %d = %d\n",a,i,answer);
(gdb)
12 / 3 = 4
15          printf("\n");
(gdb) cont
Continuing.

a = 12, i = 2

Breakpoint 1, main () at hoge.c:13
13          answer = calc(a,i);
(gdb)
```

Fig. 10　next コマンドの使用

```
gdb) break 13
Breakpoint 1 at 0x8048496: file hoge.c, line 13.
(gdb) run
Starting program: /home/yoneda/gdb/gdb_program/run_hoge
a / i = answer

a = 12, i = 3

Breakpoint 1, main () at hoge.c:13
13          answer = calc(a,i);
(gdb) step
calc (x=12, y=3) at calc.c:5
5         ans = x / y;
(gdb)
6         return ans;
(gdb)
7       }
(gdb)
main () at hoge.c:14
14          printf("%d / %d = %d\n",a,i,answer);
(gdb) cont
Continuing.
12 / 3 = 4

a = 12, i = 2

Breakpoint 1, main () at hoge.c:13
13          answer = calc(a,i);
(gdb)
```

Fig. 11　step コマンドの使用

3.5　コマンドのオンラインヘルプと省略名

コマンドの説明を help コマンドで調べることができる．Fig. 12 中の　で示したとおり，引数なしで help コマンドを入力すると，コマンドのカテゴリーを表示する．Fig. 12 中の　のように，help コマンドの引数にカテゴリー名を与えると，カテゴリーに属するコマンドの一覧が表示される．Fig. 12 中の　のように，コマンド名を help コマンドの引数に与えると，そのコマンドの説明が表示される．

各コマンドは省略形で入力することができる．例えば break コマンドは“ bre ”や“ b ”のように後ろの文字を省略

できる．step と stepi のように途中まで同じ場合は，該当するコマンド内に優先的に実行されるものである．この二つの場合では，“ s ”，は step コマンドに割り当てられている．

コマンド入力も省略できる．コマンド入力なしにエンターキーを押すと，直前のコマンドが実行される．next コマンドや step コマンドのように同じコマンドを何回も入力する必要があるときに使う．

```
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
<中略>
Type "help" followed by a class name for a list of commands in that class.
Type "help" followed by command name for full documentation.
Command name abbreviations are allowed if unambiguous.
(gdb) help running
Running the program.

List of commands:

attach -- Attach to a process or file outside of GDB
continue -- Continue program being debugged
detach -- Detach a process or file previously attached
finish -- Execute until selected stack frame returns
<中略>
Type "help" followed by command name for full documentation.
Command name abbreviations are allowed if unambiguous.
(gdb) help run
Start debugged program.  You may specify arguments to give it.
Args may include "*", or "[...]"; they are expanded using "sh".
Input and output redirection with ">", "<", or ">>" are also allowed.

With no arguments, uses arguments last specified (with "run" or "set args").
To cancel previous arguments and run with no arguments,
use "set args" without arguments.
```

Fig. 12　help コマンド