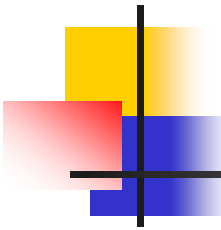
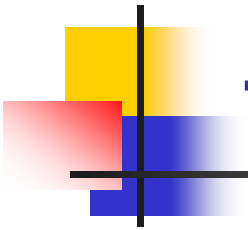
プログラミング及び演習
第13回 大規模プログラミング
(2017/07/21)

講義担当
大学院情報学研究科知能システム学専攻
教授　森　健策
大学院情報学研究科知能システム学専攻
助教　小田　昌宏



本日の講義・演習の内容

- 大きなプログラムを作る
  - 教科書 第12章
  - gdb
- 講義・演習ホームページ
  - http://www.newves.org/~mori/17Programming
- ところで,
  - プログラミング及び演習もいよいよ終盤です。
  - 来週はC++を簡単に講義します。



デバッガを使おう

- デバッガとは?
  - プログラムの動作の様子を解析するプログラム
  - 機能例
    - １行１行プログラムを実行
    - 変数の内容を確認
    - メモリの様子を確認
    - 関数の呼び出し履歴を確認

gdbとは1（centos: man gdbより)

- GDB をはじめとするデバッガは、プログラムが実行中もしくはクラッシュした時にそのプログラムの‘‘内部’’で何が行なわれているか/行われていたかを調べるのに使用されます。

gdbとは2（centos: man gdbより)

- GDB は、４ つの機能 (加えてこれらをサポートする機能) によって実行中にバグを見つけることを手助けします。
  - 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、何が起こったか調べる。
  - バグによる副作用を修正し、別のバグを調べるためプログラムの状態を 変更する。

gdbとは3 (centos: man gdbより)

- GDB では C, C++, Modula-2 などで書かれたプログラムのデバッグが行なえます。
- GNU Fortran コンパイラが完成すれば Fortran もサポートされます。
- GDB はシェルコマンドgdbで起動されます。いったん起動すると、GDB　コマンドquitを実行して終了するまで、端末からコマンドを読み続けます。gdbのオンラインヘルプは(gdbの中で) helpコマンドを実行すれば表示されます。

gdbとは4 (centos: man gdbより)

- gdb は引数やオプション無しで起動できますが、たいてい、1 つか 2 つの引数 を付けて起動します。実行プログラムを引数にする場合は以下のようになります:
  - gdb program
- また実行プログラムと core ファイルの両方を指定することもできます:
  - gdb program core

gdbとは5 (centos: man gdbより)

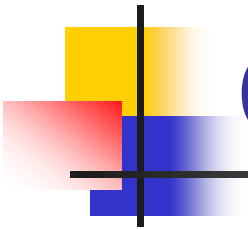
- もし 実行中のプロセスのデバッグを行ないたい場合には、第 2 引数としてcoreの代わりにプロセス ID を指定します:
  - gdb program 1234
- これは GDB をプロセス ID 1234 のプロセスに接続します(このとき ‘1234’とい う名前のファイルが存在してはいけません。GDB はまず core ファイルを最初にチェックしにいくからです)。

Segmentation faultが発生するプログラム lec13-seg1.c

```
#include <stdio.h>
static int func1();
static int *myAlloc();
main()
{
  int val;
  val = func1();
  printf("val = %d¥n",val);
}
static int func1()
{
  int retVal;
  int *retPtr;
  retPtr = myAlloc();
  *retPtr = 10;
  retVal = *retPtr;
  return retVal;
}
```

```
static int *myAlloc()
{
  return NULL;
}
```

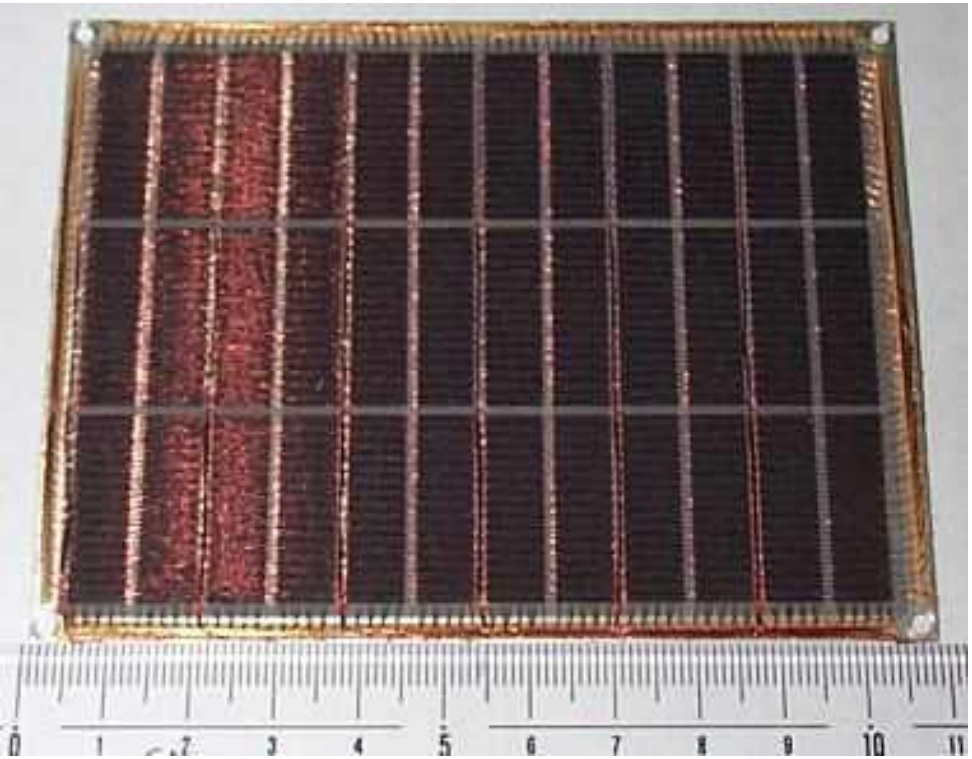
myAllocでint型変数を格納するメモリ領域を確保するはずなのに，確保せずに0ポインタを返している

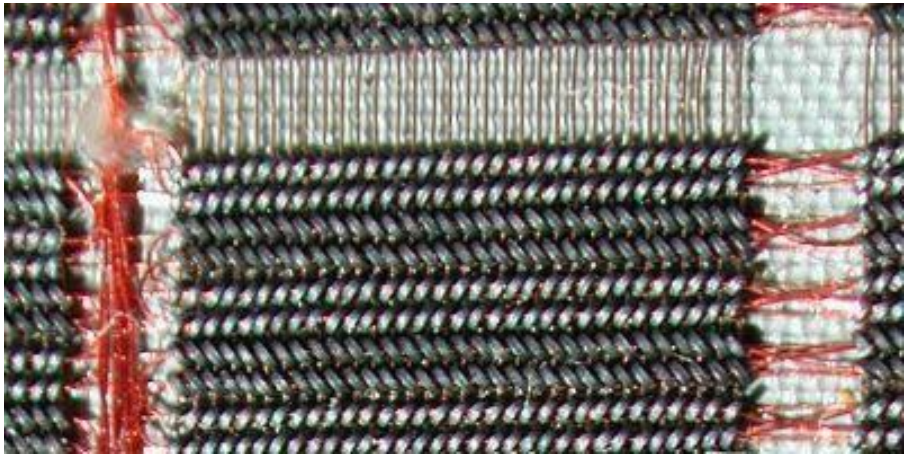


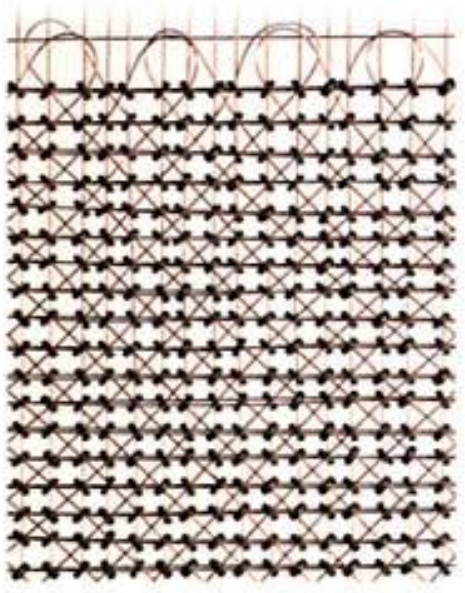
Core dump

- coreファイルとは?
  - Segmentation faultが発生したときにメモリの内容を記録したcoreファイルを吐き出す
  - core.プロセス番号
- coreファイルを得るには
  - limit coredumpsize 100m を実行
  - 100MByteまでのcoreファイルが生成
- 注意点
  - coreファイルは大きなファイルであるため，通常はlimit coredumsize 0 としcore dumpしないように
  - デバッグが終了したら必ず消すこと

coreメモリ

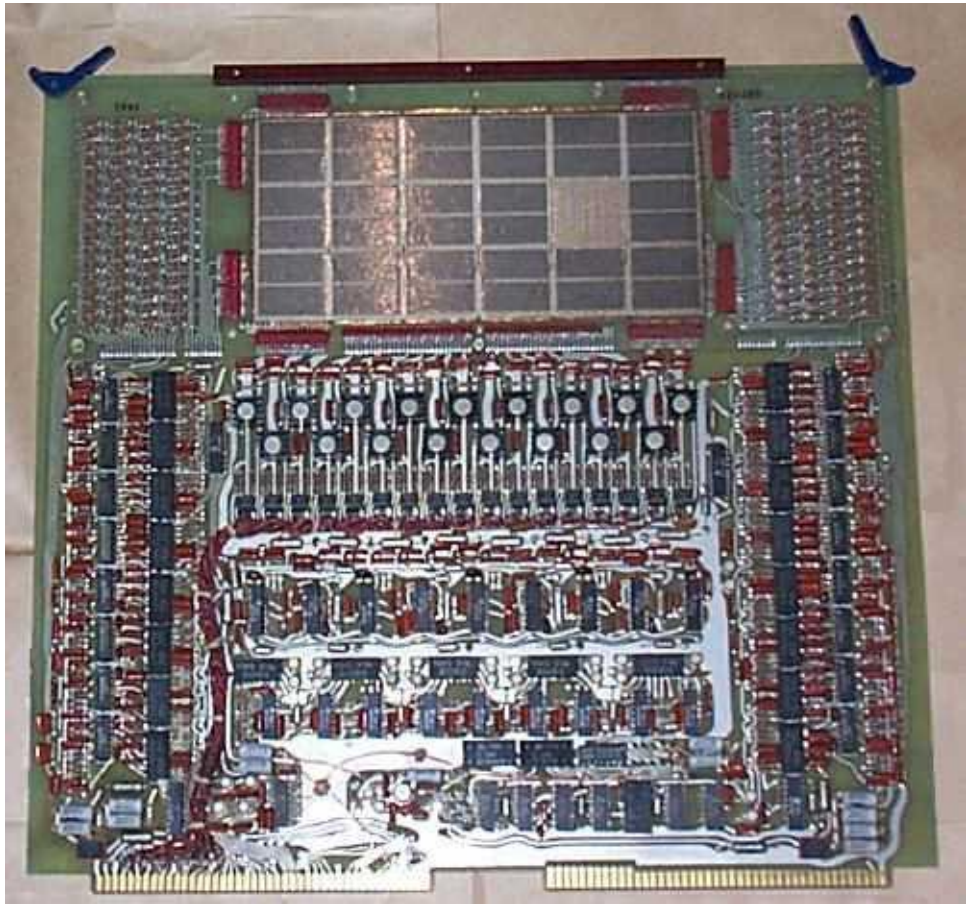


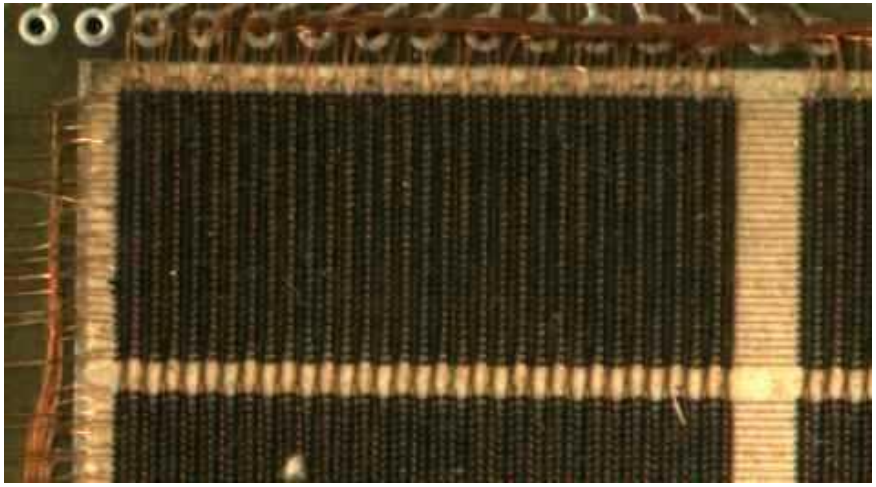


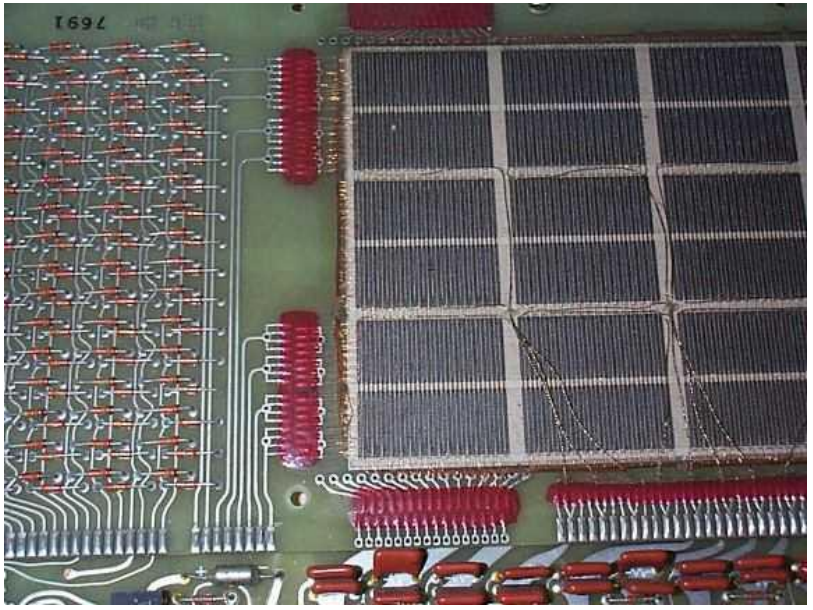


http://www.st.rim.or.jp/~nkomatsu/premicro/corememe.html
http://www-6.ibm.com/jp/event/museum/rekishi/core.htmlより

coreメモリ

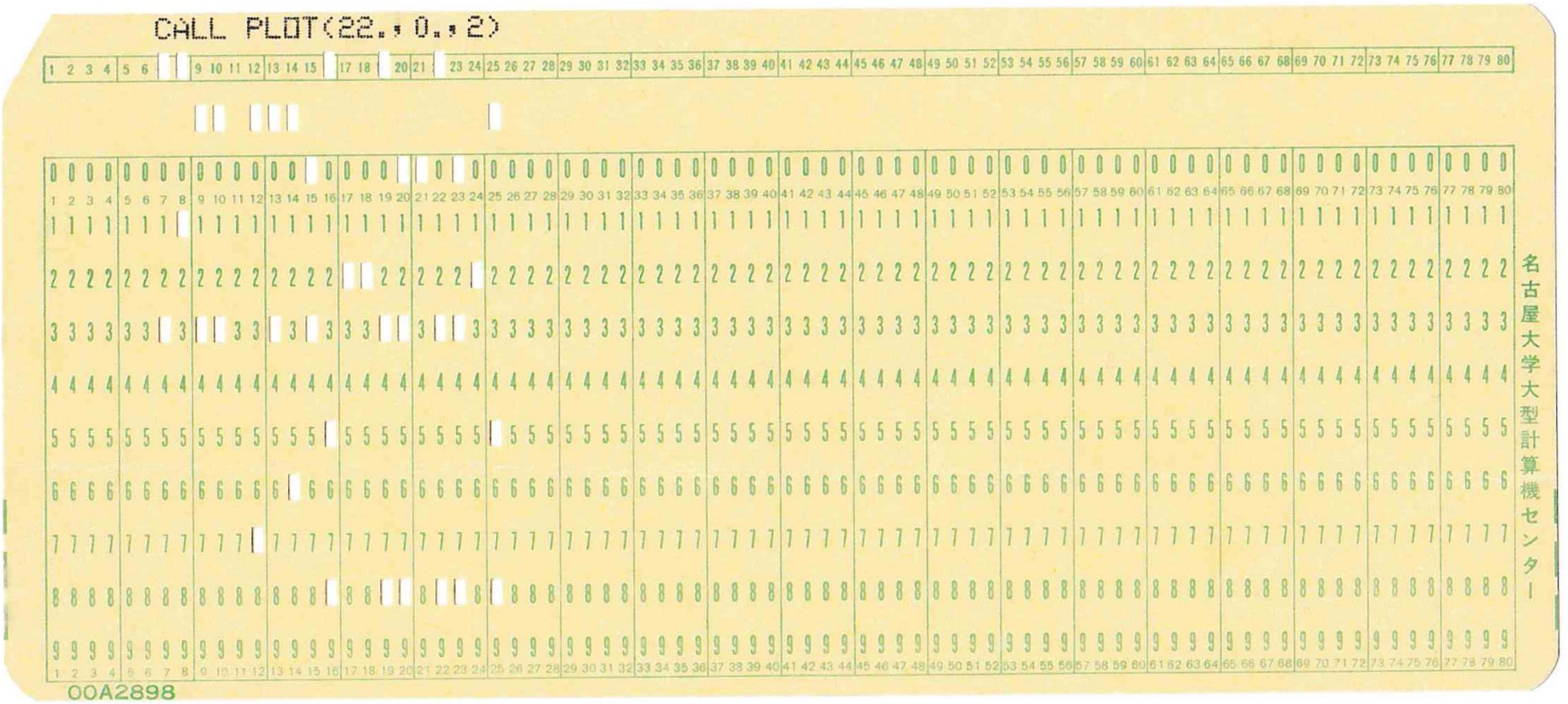






http://www.st.rim.or.jp/~nkomatsu/premicro/coremem.htmlより

ついでにパンチカード



デバッガの起動とコマンド (1/2)

- 起動方法
  - gdb 実行ファイル名 （coreファイル)
- コマンド
  - run
    - 実行ファイルを実行
  - backtrace
    - 現在止まっている位置にどのように到達したかを示す
  - list 行番号
    - 行番号のソースファイルを表示
  - print 変数名
    - 変数名の内容を表示
  - up (n)
    - 関数フレームを1(n)段up
  - frame
    - 現在のフレームを表示
  - frame n
    - n番目のフレームを選択
  - quit
    - gdbを終了

デバッガの起動とコマンド (2/2)

- コマンド
  - break 行番号
    - breakポイントの設定
  - delete (or clear) ブレークポイント番号
    - ブレークポイントの解除
  - next (or step)
    - 次の行を実行
  - continue
    - 次のブレークポイントまで実行
  - examine 変数
    - 変数のアドレスで示されるメモリ内容を表示
    - x/12 buf　　x/12b など

coreファイルを用いてデバッグ

```
ssh.ice.nuie.nagoya-u.ac.jp{mori}46: cc -o lec13-seg1 lec13-seg1.c
ssh.ice.nuie.nagoya-u.ac.jp{mori}47: ./lec13-seg1
セグメントエラー (coreを出力しました)
ssh.ice.nuie.nagoya-u.ac.jp{mori}48: gdb lec13-seg1 core.28576
GNU gdb (GDB) Red Hat Enterprise Linux (7.0.1-23.el5_5.1)
Copyright (C) 2009 Free Software Foundation, Inc.
            途中省略
Reading symbols from /home0/mori/gdb/lec13-seg1...(no debugging symbols
found)...done.
Reading symbols from /lib/libc.so.6...(no debugging symbols found)...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...(no debugging symbols found)...done.
Loaded symbols for /lib/ld-linux.so.2
Core was generated by `./lec13-seg1'.
Program terminated with signal 11, Segmentation fault.
#0  0x080483ea in func1 ()
(gdb) backtrace
#0  0x080483ea in func1 ()
#1  0x080483ba in main ()
(gdb)
```

list表示と変数表示

- デバッグ情報を含めてコンパイル
  - gcc -g lec13-seg1.c
  - -gオプションをつけることで実行ファイルにデバッグ情報が含まれる
- 異常終了した部分のリストを表示可能
- 任意の変数の値を調査可能

```
ssh.ice.nuie.nagoya-u.ac.jp{mori}52: gdb lec13-seg1
core.28576
GNU gdb (GDB) Red Hat Enterprise Linux (7.0.1-
23.el5_5.1)
Core was generated by `./lec13-seg1'.
Program terminated with signal 11, Segmentation fault.
#0  0x080483ea in func1 () at lec13-seg1.c:17
17        *retPtr = 10;
(gdb) list 17
12      static int func1()
13      {
14        int retVal;
15        int *retPtr;
16        retPtr = myAlloc();
17        *retPtr = 10;
18        retVal = *retPtr;
19        return retVal;
20      }
21
(gdb) print retPtr
$1 = (int *) 0x0
(gdb)
```

Segmentation faultを発生するプログラム lec13-seg2.c

```
#include <stdio.h>
static char * func1();
static char *myAlloc();

main()
{
  char *retPtr;
  retPtr = func1();
  printf("buffer %s¥n",retPtr);
}

static
char *func1()
{
  char *retPtr;
  retPtr = myAlloc();
  sprintf(retPtr,"%s","ABC");
  return retPtr;
}
```

```
static
char *myAlloc()
{
  char *retPtr;
  retPtr = 0;
  return retPtr;
}
```

myAllocで文字列を格納するメモリ領域を確保するはずなのに，確保せずに0ポインタを返している

gdbによるbacktrace

```
ssh.ice.nuie.nagoya-u.ac.jp{mori}58: gdb lec13-seg2 core.28614
GNU gdb (GDB) Red Hat Enterprise Linux (7.0.1-23.el5_5.1)

        途中略

Reading symbols from /lib/ld-linux.so.2...(no debugging symbols
found)...done.
Loaded symbols for /lib/ld-linux.so.2
Core was generated by `./lec13-seg2'.
Program terminated with signal 11, Segmentation fault.
#0  0x080483ea in func1 () at lec13-seg2.c:17
17          sprintf(retPtr,"%s","ABC");
(gdb) bt
#0  0x080483ea in func1 () at lec13-seg2.c:17
#1  0x080483ba in main () at lec13-seg2.c:8
(gdb)
```

フレーム番号

backtraceで関数の呼び出し状況をチェック

17行目でエラーが発生

listを表示

```
(gdb) bt
#0  0x080483ea in func1 () at lec13-seg2.c:17
#1  0x080483ba in main () at lec13-seg2.c:8
(gdb) list 17
12      static
13      char *func1()
14      {
15        char *retPtr;
16        retPtr = myAlloc();
17        sprintf(retPtr,"%s","ABC");
18        return retPtr;
19      }
20
21      static
(gdb)
```

retPtrがどうも怪しい

retPtrを調べてみる

frame操作

```
(gdb) print retPtr
$1 = 0x0
(gdb) f
#0  0x080483ea in func1 () at lec13-seg2.c:17
17          sprintf(retPtr,"%s","ABC");
(gdb) up
#1  0x080483ba in main () at lec13-seg2.c:8
8           retPtr = func1();
(gdb) down
#0  0x080483ea in func1 () at lec13-seg2.c:17
17          sprintf(retPtr,"%s","ABC");
(gdb) frame 0
#0  0x080483ea in func1 () at lec13-seg2.c:17
17          sprintf(retPtr,"%s","ABC");
(gdb) frame 1
#1  0x080483ba in main () at lec13-seg2.c:8
8           retPtr = func1();
```

retPtrをprint

現在はfunc1()にいるようだ

一つ上へ → func1を呼び出したmain()へ

一つ下へ

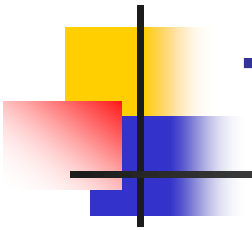
frame番号0を選択

frame番号1を選択

ブレークポイント lec13-seg3.c

```
#include <stdio.h>

main()
{
  char buf[256];
  char *p;
  sprintf(buf, "%s","ABC¥n");
  p = buf;
  printf( "%s", p);
}
```



ブレークポイントを設定

```
ssh.ice.nuie.nagoya-u.ac.jp{mori}101: gdb lec13-seg3
(gdb) list
1       #include <stdio.h>
2
3       main()
4       {
5         char buf[256];
6         char *p;
7         sprintf(buf,"%s","ABC¥n");
8         p = buf;
9         printf( "%s", p);
10       }
(gdb) b 7
Breakpoint 1 at 0x80483b8: file lec13-seg3.c, line 7.¥
(gdb) run
Starting program: /home0/mori/gdb/lec13-seg3
Breakpoint 1, main () at lec13-seg3.c:7
7         sprintf(buf,"%s","ABC¥n");
```

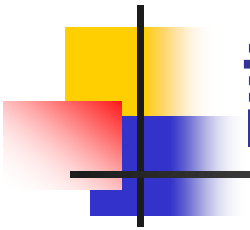
実行制御とメモリ内容表示

```
(gdb) n
8         p = buf;
(gdb) print p
$1 = 0x69e600 "U¥211¥345WVS¥350¥260x"
(gdb) n
9         printf( "%s", p);
(gdb) print p
$2 = 0xbfffe720 "ABC¥n"
(gdb) x /12bx p
0xbfffe720:     0x41    0x42    0x43    0x0a    0x00    0x08    0x00    0x00
0xbfffe728:     0xd8    0x72    0xfe    0xb7
(gdb) s
ABC
10      }
(gdb) s
0x006c4e9c in __libc_start_main () from /lib/libc.so.6
(gdb) info break
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x080483b8 in main at lec13-seg3.c:7
        breakpoint already hit 1 time
```

s step実行

よく利用される GDB コマンド (centos: man gdbより)

- break [file:]function
  - プレークポイントを (file内の) functionに設定します。
- run [arglist]
  - プログラムの実行を開始します(もしあれば arglistを引数として)。
- 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 expr
  - 式の値を表示します。
- c
  - プログラムの実行を再開します。(たとえばブレークポイントで実行を中断した後で)
- 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。
- help [name]
  - GDB コマンド nameについての情報や、GDB を使う上での一般的な情報を表示します。
- quit
  - GDB を終了します。



課題

- emacsの中でgdbを使用する方法を各自調べなさい
- はじめの一歩
  - 起動方法
    - M-x gdb
    - その後gdbの引数を聞かれるので、
      gdb lec13-seg1
      のようにタイプする
  - ブレークポイントの設定 C-x space
  - ブレークポイントの解除 C-x C-a C-d
  - 1行進む C-x C-a C-s (関数呼び出し時も1行ごと)
  - 1行進む C-x C-a C-n
  - 次のブレークポイントまで実行 C-x C-a C-r