Debugging with GDB

HGU

gdb: a command-line debugging tool

GNU Debugger can :

- **Start** a program being debugged
- **Step** through it line by line
- **Pause** the program when it reaches certain conditions
  (breakpoint, user-defined condition, or crash)
- **Show** the status at the paused point
  (value of variables, stack frame, execution state)
- **Continue** the program after pause

GDB Commands

- Before using GDB, compile option -g is needed

```
$ gcc -g prog.c
```

- GDB uses the executable file

```
$ gcc a.out
```

- If the file crashed, core file dumped by OS could be running with GDB

```
$ gdb core a.out
```

Usual debug process with gdb

- Compiler source with –g flag –o <executable filename>
- Envoke gdb debugger with executable

  $ gdb <executable filename>
- in a gdb, use commands

  (gdb) **list** [<function_name>] (look at the source code lines)
  (gdb) **break** <function_name> (setup break point with function name)
  (gdb) **break** <line_number> (setup break point with line number)
  (gdb) **run** [optional argument for executable] (start program)
  (gdb) **next** or **step** (execute instruction one by one manner)
  (gdb) **print** <variable> (look at the contents of variables>
  (gdb) **continue** (run until next breakpoint is met)
  (gdb) **quit** (stop debugging)

GDB Commands

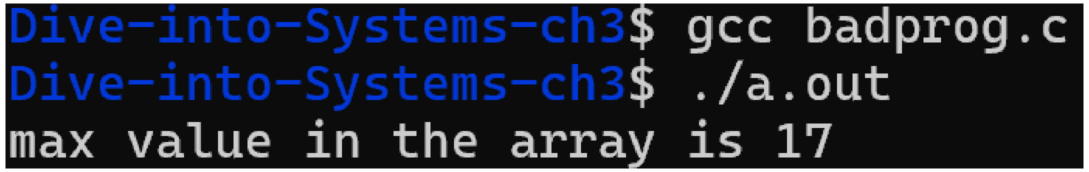
Commands	Description
break	Set a breakpoint
run	Start program running from the beginning
continue	Continue execution of the program until it hits a breakpoint
quit	Quit the GDB session
next	Allow program to execute the next line of C code and then pause it
step	Allow program to execute the next line of C code; if the next line contains a function call, step into the function and pause
list	List C source code around pause point or specified point
print	Print out the value of a program variable (or expression)
where	Print the call stack
frame	Move into the context of a specific stack frame

GDB Commands Uses

badprog.c

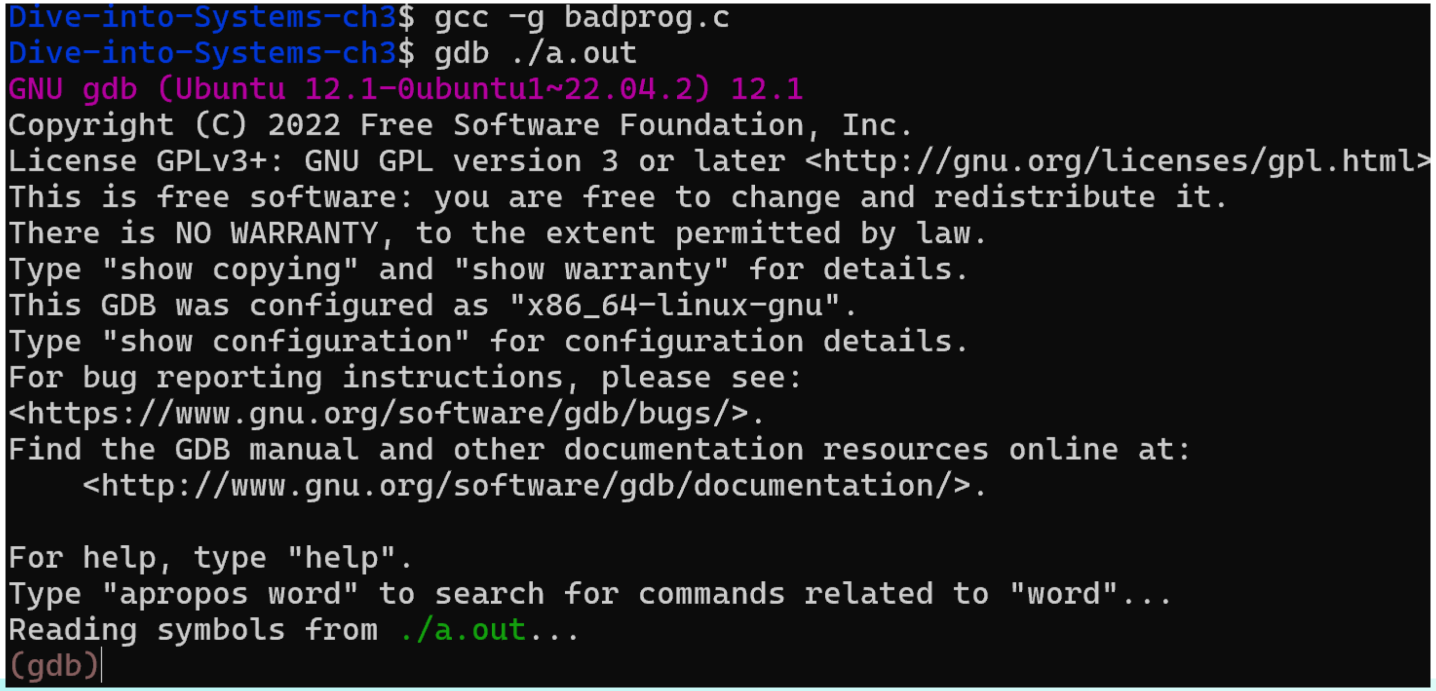
```c
// badprog.c finds the largest element in the array
#include stdio.h
#include stdlib.h
#include string.h

int findAndReturnMax(int *arr, int len, int max) {
    int i;
    if (!arr || (len=0) )
        return -1;
    max = arr[0];
    for ( i = 1; i <= len; i++ )
        if (max  arr[i])
            max = arr[i];
    return 0;
}

int main(int argc, char *argv[]) {
    int arr[5] = { 17, 21, 44, 2, 60 };
    int max = arr[0];

    if ( findAndReturnMax(arr, 5, max) != 0 ) {
        printf("strange error\n");
        exit(1);
    }
    printf("max value in the array is %d\n", max);

    return 0;
}
```

Expected result is 60, but



```
Dive-into-Systems-ch3$ gcc badprog.c
Dive-into-Systems-ch3$ ./a.out
max value in the array is 17
```

We will find out what's wrong using GDB



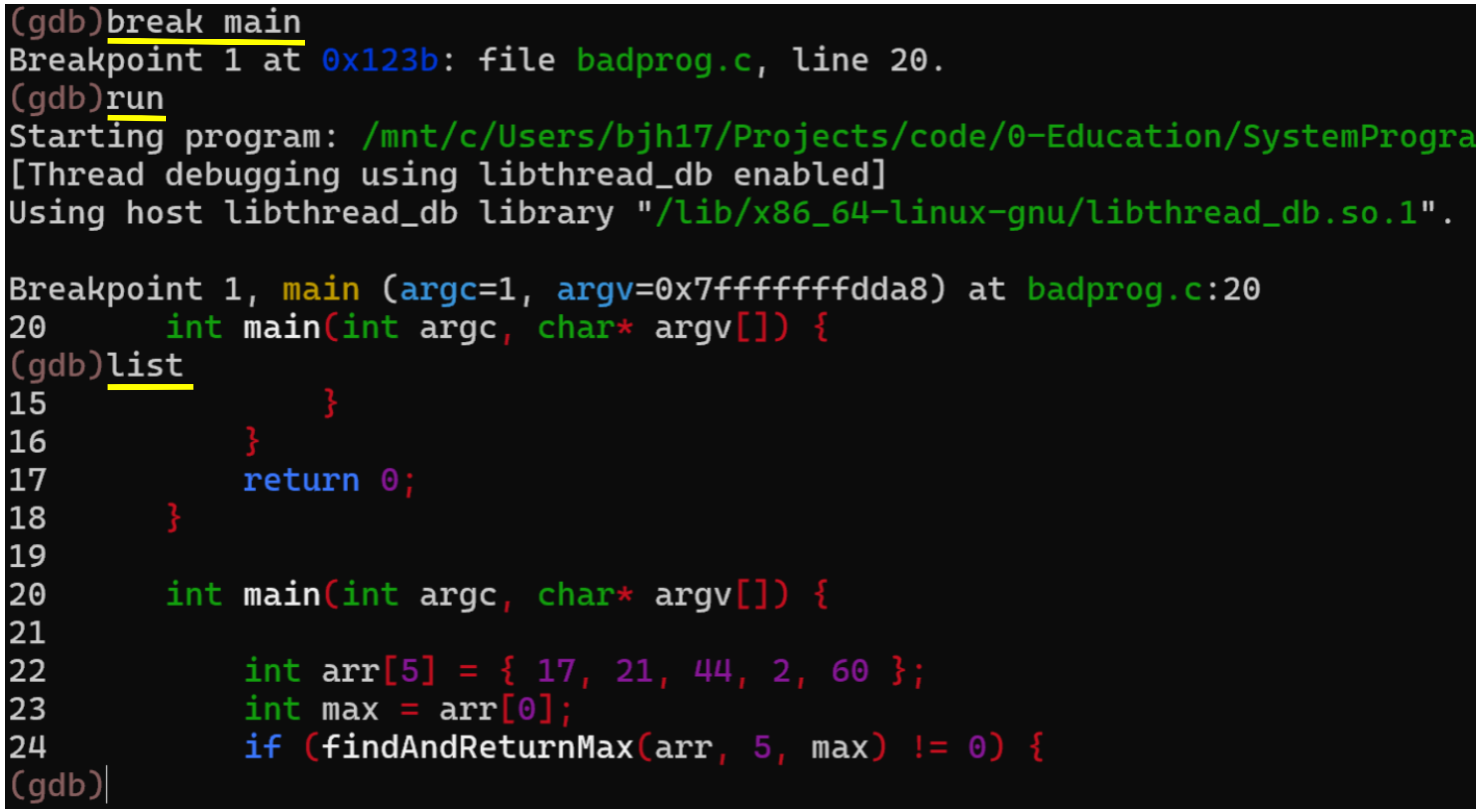
```
Dive-into-Systems-ch3$ gcc -g badprog.c
Dive-into-Systems-ch3$ gdb ./a.out
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./a.out...
(gdb)
```

GDB Commands Uses

badprog.c

```c
// badprog.c finds the largest element in the array
#include stdio.h
#include stdlib.h
#include string.h

int findAndReturnMax(int *arr, int len, int max) {
    int i;
    if (!arr || (len=0) )
        return -1;
    max = arr[0];
    for ( i = 1; i <= len; i++ )
        if (max  arr[i])
            max = arr[i];
    return 0;
}

int main(int argc, char *argv[]) {
    int arr[5] = { 17, 21, 44, 2, 60 };
    int max = arr[0];

    if ( findAndReturnMax(arr, 5, max) != 0 ) {
        printf("strange error\n");
        exit(1);
    }
    printf("max value in the array is %d\n", max);

    return 0;
}
```

Start from main() function
by setting breakpoint at main and 'run'



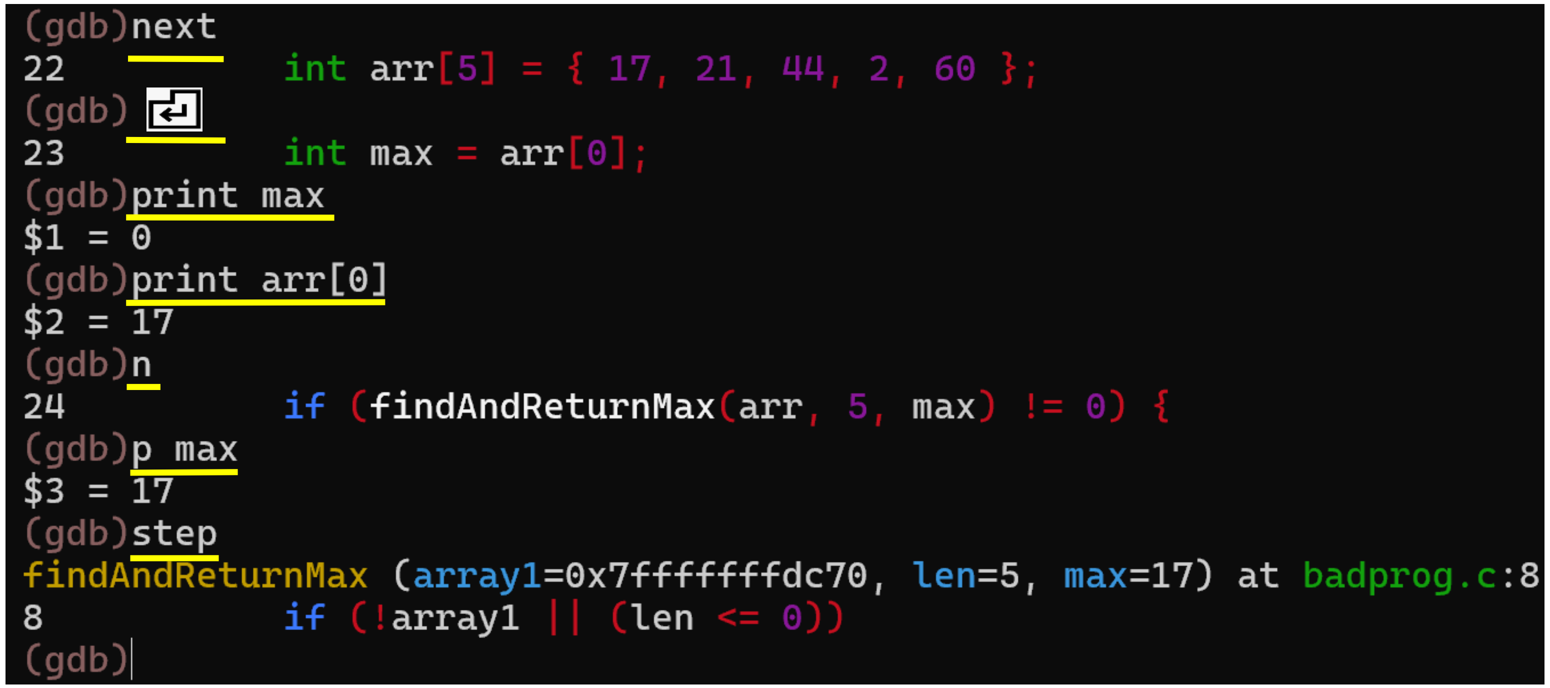
```
(gdb)break main
Breakpoint 1 at 0x123b: file badprog.c, line 20.
(gdb)run
Starting program: /mnt/c/Users/bjh17/Projects/code/0-Education/SystemProgra
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffdda8) at badprog.c:20
20      int main(int argc, char* argv[]) {
(gdb)list
15                      }
16                  }
17                  return 0;
18          }
19
20      int main(int argc, char* argv[]) {
21
22              int arr[5] = { 17, 21, 44, 2, 60 };
23              int max = arr[0];
24              if (findAndReturnMax(arr, 5, max) != 0) {
(gdb)
```

GDB Commands Uses

badprog.c

```
// badprog.c finds the largest element in the array
#include stdio.h
#include stdlib.h
#include string.h

int findAndReturnMax(int *arr, int len, int max) {
    int i;
    if (!arr || (len=0) )
        return -1;
    max = arr[0];
    for ( i = 1; i <= len; i++ )
        if (max  arr[i])
            max = arr[i];
    return 0;
}

int main(int argc, char *argv[]) {
    int arr[5] = { 17, 21, 44, 2, 60 };
    int max = arr[0];

    if ( findAndReturnMax(arr, 5, max) != 0 ) {
        printf("strange error\n");
        exit(1);
    }
    printf("max value in the array is %d\n", max);

    return 0;
}
```

Execute each one line of code (by next) examining program state (by print)



```
(gdb)next
22          int arr[5] = { 17, 21, 44, 2, 60 };
(gdb) ⏎
23          int max = arr[0];
(gdb)print max
$1 = 0
(gdb)print arr[0]
$2 = 17
(gdb)n
24          if (findAndReturnMax(arr, 5, max) != 0) {
(gdb)p max
$3 = 17
(gdb)step
findAndReturnMax (array1=0x7fffffffdc70, len=5, max=17) at badprog.c:8
8           if (!array1 || (len <= 0))
(gdb)
```

- many commands has their shortcuts (by alias command)
- Enter(⏎) means the most previous command

GDB Commands: function call is an instruction ?

badprog.c

```
// badprog.c finds the largest element in the array
#include stdio.h
#include stdlib.h
#include string.h

int findAndReturnMax(int *arr, int len, int max) {
    int i;
    if (!arr || (len=0) )
        return -1;
    max = arr[0];
    for ( i = 1; i <= len; i++ )
        if (max  arr[i])
            max = arr[i];
    return 0;
}

int main(int argc, char *argv[]) {
    int arr[5] = { 17, 21, 44, 2, 60 };
    int max = arr[0];

    if ( findAndReturnMax(arr, 5, max) != 0 ) {
        printf("strange error\n");
        exit(1);
    }
    printf("max value in the array is %d\n", max);

    return 0;
}
```

- next vs. step

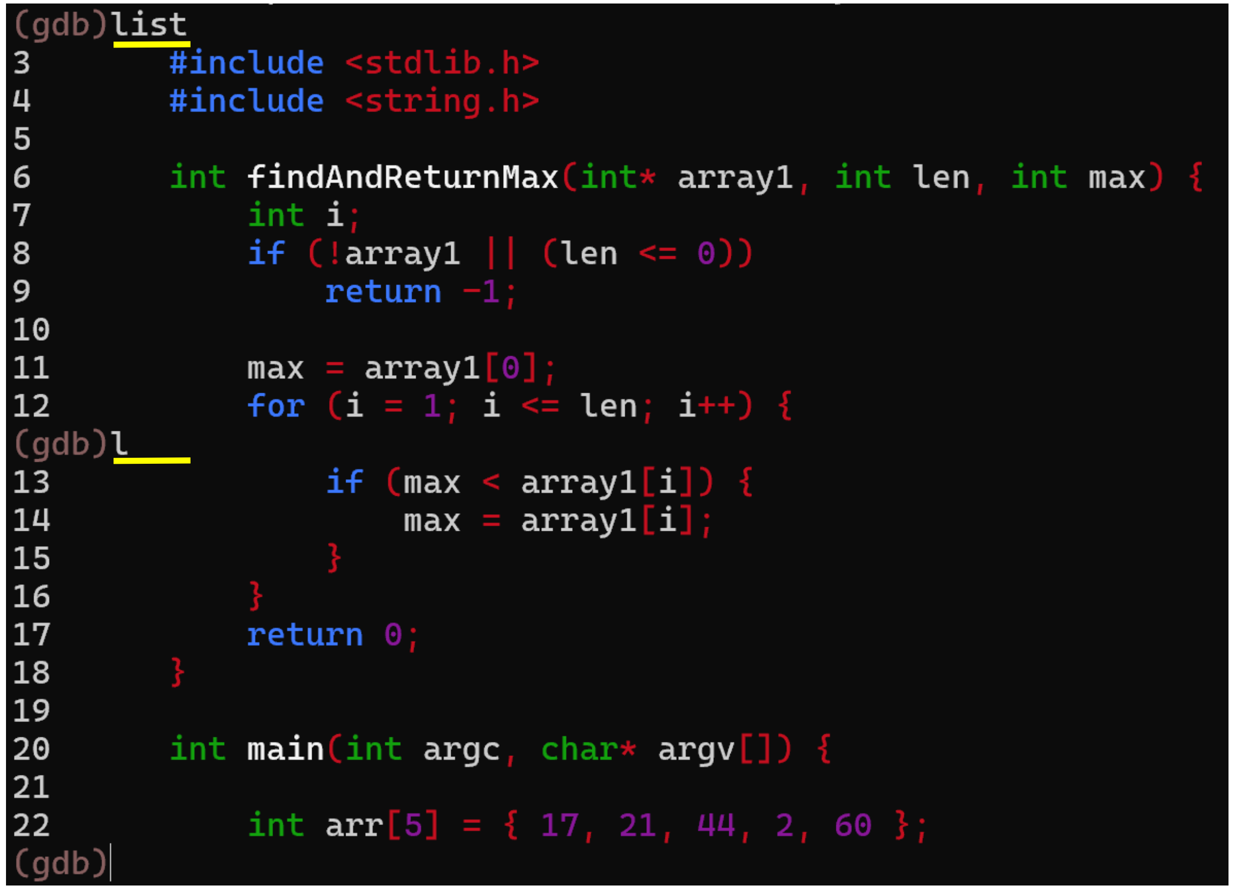
next	step
Execute one line	Execute one instruction (dive into a function)
Function call and its return is executed as part of next command	Step into called function, and pause before the first instruction of function

GDB Commands Uses

badprog.c

```
// badprog.c finds the largest element in the array
#include stdio.h
#include stdlib.h
#include string.h

int findAndReturnMax(int *arr, int len, int max) {
    int i;
    if (!arr || (len=0) )
        return -1;
    max = arr[0];
    for ( i = 1; i <= len; i++ )
        if (max  arr[i])
            max = arr[i];
    return 0;
}

int main(int argc, char *argv[]) {
    int arr[5] = { 17, 21, 44, 2, 60 };
    int max = arr[0];

    if ( findAndReturnMax(arr, 5, max) != 0 ) {
        printf("strange error\n");
        exit(1);
    }
    printf("max value in the array is %d\n", max);

    return 0;
}
```

List a Function



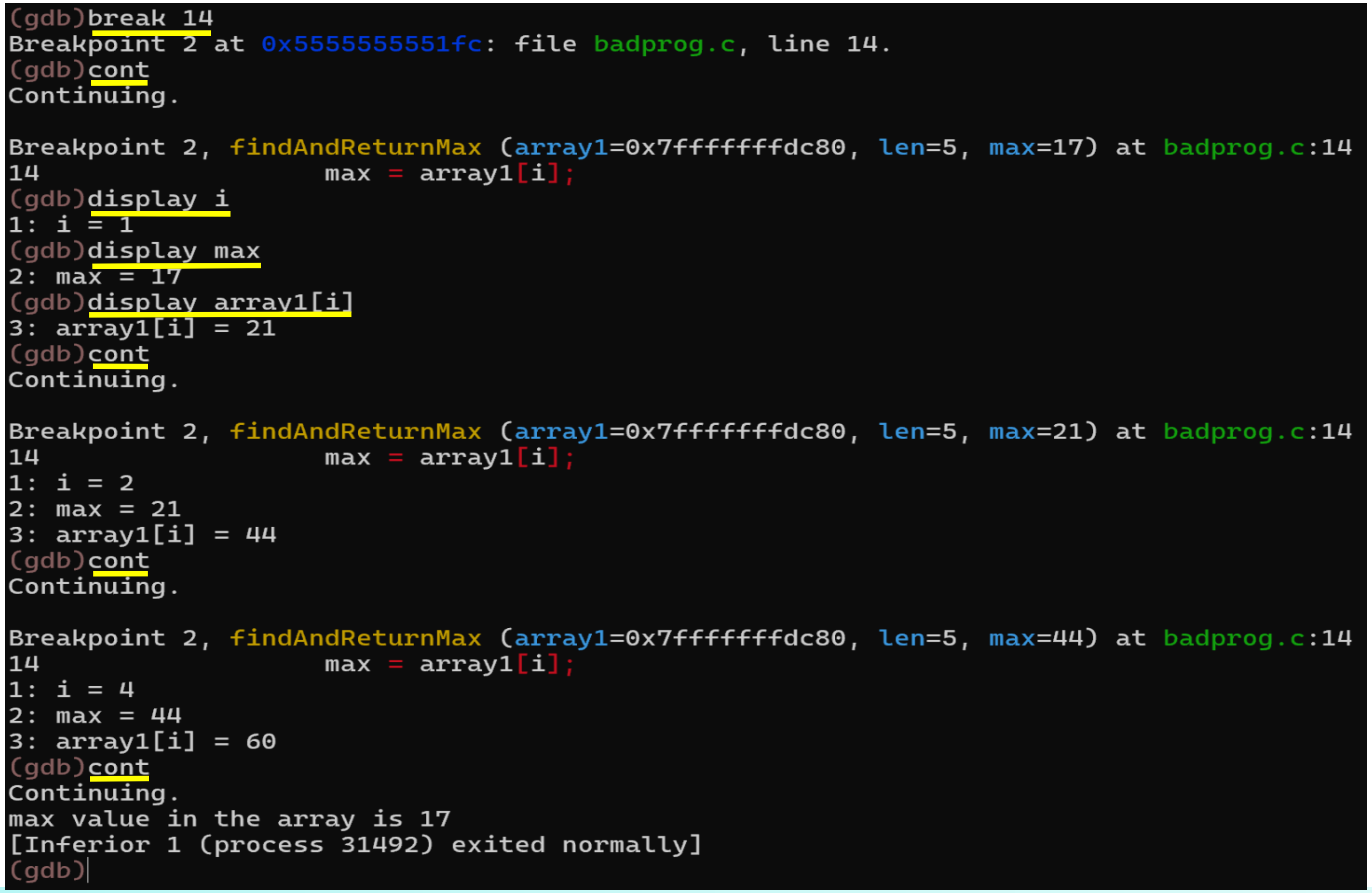
```
(gdb)list
3       #include <stdlib.h>
4       #include <string.h>
5
6       int findAndReturnMax(int* array1, int len, int max) {
7           int i;
8           if (!array1 || (len <= 0))
9               return -1;
10
11          max = array1[0];
12          for (i = 1; i <= len; i++) {
(gdb)l
13              if (max < array1[i]) {
14                  max = array1[i];
15              }
16          }
17          return 0;
18      }
19
20      int main(int argc, char* argv[]) {
21
22          int arr[5] = { 17, 21, 44, 2, 60 };
(gdb)
```

GDB Commands Uses

badprog.c

```c
// badprog.c finds the largest element in the array
#include stdio.h
#include stdlib.h
#include string.h

int findAndReturnMax(int *arr, int len, int max) {
    int i;
    if (!arr || (len=0) )
        return -1;
    max = arr[0];
    for ( i = 1; i <= len; i++ )
        if (max  arr[i])
            max = arr[i];
    return 0;
}

int main(int argc, char *argv[]) {
    int arr[5] = { 17, 21, 44, 2, 60 };
    int max = arr[0];

    if ( findAndReturnMax(arr, 5, max) != 0 ) {
        printf("strange error\n");
        exit(1);
    }
    printf("max value in the array is %d\n", max);

    return 0;
}
```

Display the same set of variables



```
(gdb)break 14
Breakpoint 2 at 0x5555555551fc: file badprog.c, line 14.
(gdb)cont
Continuing.

Breakpoint 2, findAndReturnMax (array1=0x7fffffffdc80, len=5, max=17) at badprog.c:14
14                   max = array1[i];
(gdb)display i
1: i = 1
(gdb)display max
2: max = 17
(gdb)display array1[i]
3: array1[i] = 21
(gdb)cont
Continuing.

Breakpoint 2, findAndReturnMax (array1=0x7fffffffdc80, len=5, max=21) at badprog.c:14
14                   max = array1[i];
1: i = 2
2: max = 21
3: array1[i] = 44
(gdb)cont
Continuing.

Breakpoint 2, findAndReturnMax (array1=0x7fffffffdc80, len=5, max=44) at badprog.c:14
14                   max = array1[i];
1: i = 4
2: max = 44
3: array1[i] = 60
(gdb)cont
Continuing.
max value in the array is 17
[Inferior 1 (process 31492) exited normally]
(gdb)
```

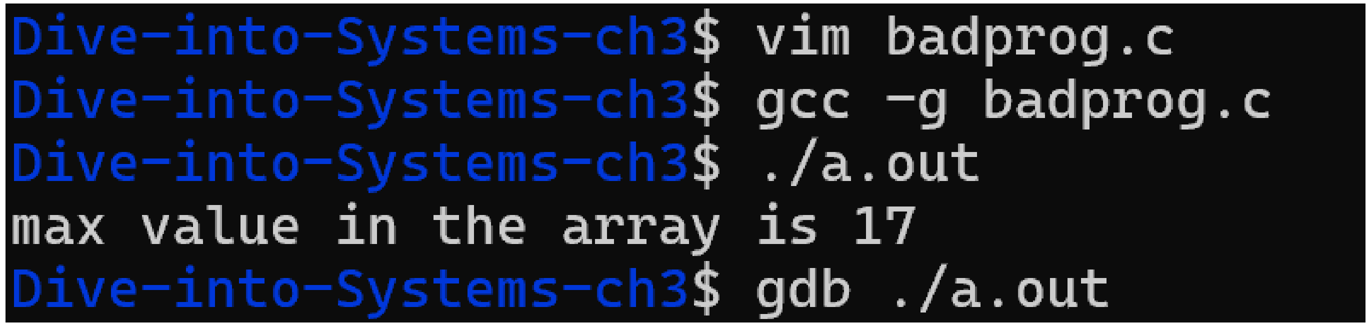
GDB Commands Uses

badprog.c

```
// badprog.c finds the largest element in the array
#include stdio.h
#include stdlib.h
#include string.h

int findAndReturnMax(int *arr, int len, int max) {
    int i;
    if (!arr || (len=0) )
        return -1;
    max = arr[0];
    for ( i = 1; i < len; i++ )
        if (max  arr[i])
            max = arr[i];
    return 0;
}

int main(int argc, char *argv[]) {
    int arr[5] = { 17, 21, 44, 2, 60 };
    int max = arr[0];

    if ( findAndReturnMax(arr, 5, max) != 0 ) {
        printf("strange error\n");
        exit(1);
    }
    printf("max value in the array is %d\n", max);

    return 0;
}
```

Now, we found the first bug!

Fix and recompile.



```
Dive-into-Systems-ch3$ vim badprog.c
Dive-into-Systems-ch3$ gcc -g badprog.c
Dive-into-Systems-ch3$ ./a.out
max value in the array is 17
Dive-into-Systems-ch3$ gdb ./a.out
```

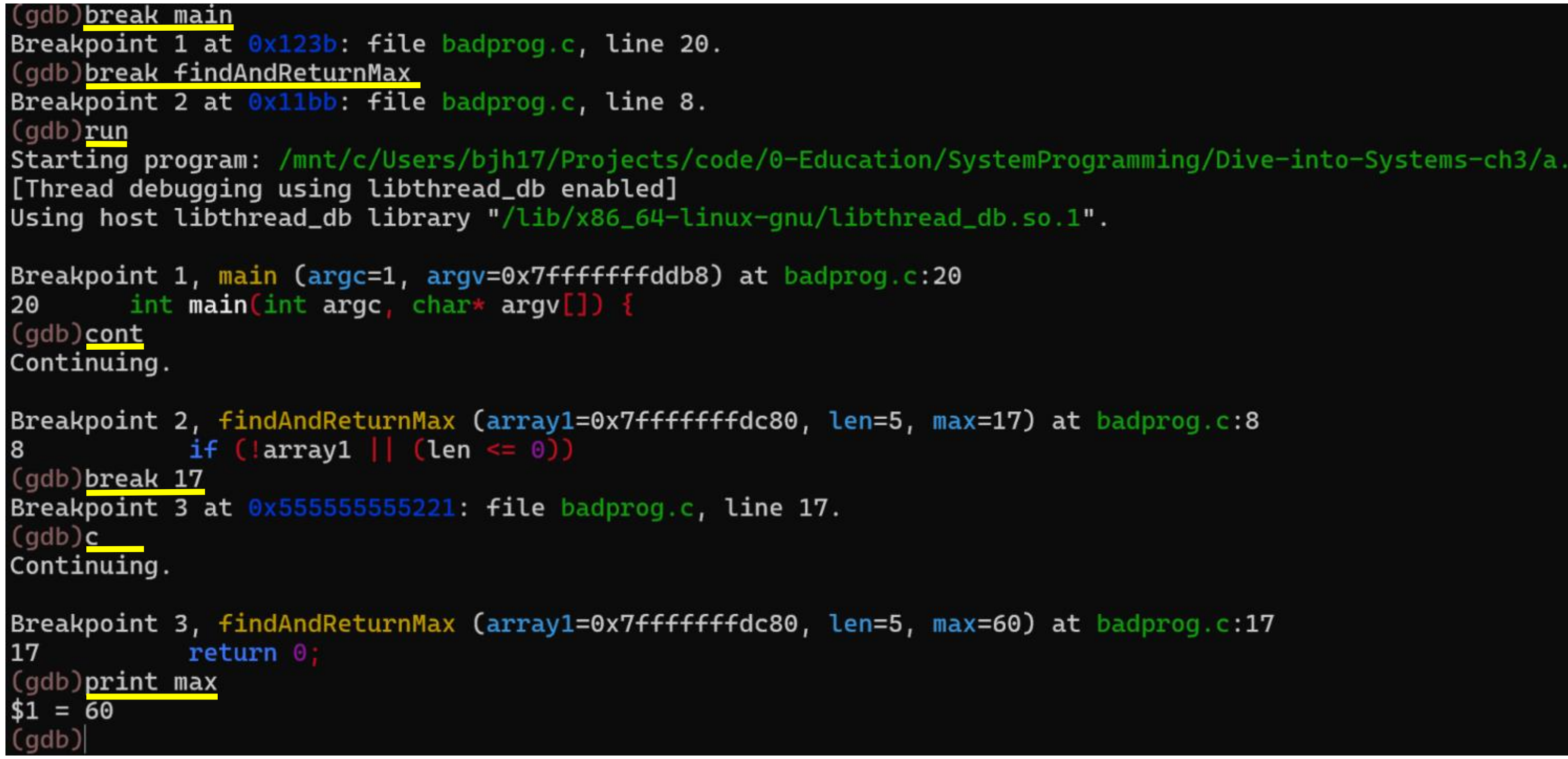
But the result is still wrong..
Resume debugging again!

GDB Commands Uses

badprog.c

```
// badprog.c finds the largest element in the array
#include stdio.h
#include stdlib.h
#include string.h

int findAndReturnMax(int *arr, int len, int max) {
    int i;
    if (!arr || (len=0) )
        return -1;
    max = arr[0];
    for ( i = 1; i < len; i++ )
        if (max  arr[i])
            max = arr[i];
    return 0;
}

int main(int argc, char *argv[]) {
    int arr[5] = { 17, 21, 44, 2, 60 };
    int max = arr[0];

    if ( findAndReturnMax(arr, 5, max) != 0 ) {
        printf("strange error\n");
        exit(1);
    }
    printf("max value in the array is %d\n", max);

    return 0;
}
```

Go back to the point we quit



```
(gdb)break main
Breakpoint 1 at 0x123b: file badprog.c, line 20.
(gdb)break findAndReturnMax
Breakpoint 2 at 0x11bb: file badprog.c, line 8.
(gdb)run
Starting program: /mnt/c/Users/bjh17/Projects/code/0-Education/SystemProgramming/Dive-into-Systems-ch3/a.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffddb8) at badprog.c:20
20      int main(int argc, char* argv[]) {
(gdb)cont
Continuing.

Breakpoint 2, findAndReturnMax (array1=0x7fffffffdc80, len=5, max=17) at badprog.c:8
8           if (!array1 || (len <= 0))
(gdb)break 17
Breakpoint 3 at 0x555555555221: file badprog.c, line 17.
(gdb)c
Continuing.

Breakpoint 3, findAndReturnMax (array1=0x7fffffffdc80, len=5, max=60) at badprog.c:17
17          return 0;
(gdb)print max
$1 = 60
(gdb)
```

- The findAndReturnMax() has found the correct max value, 60.

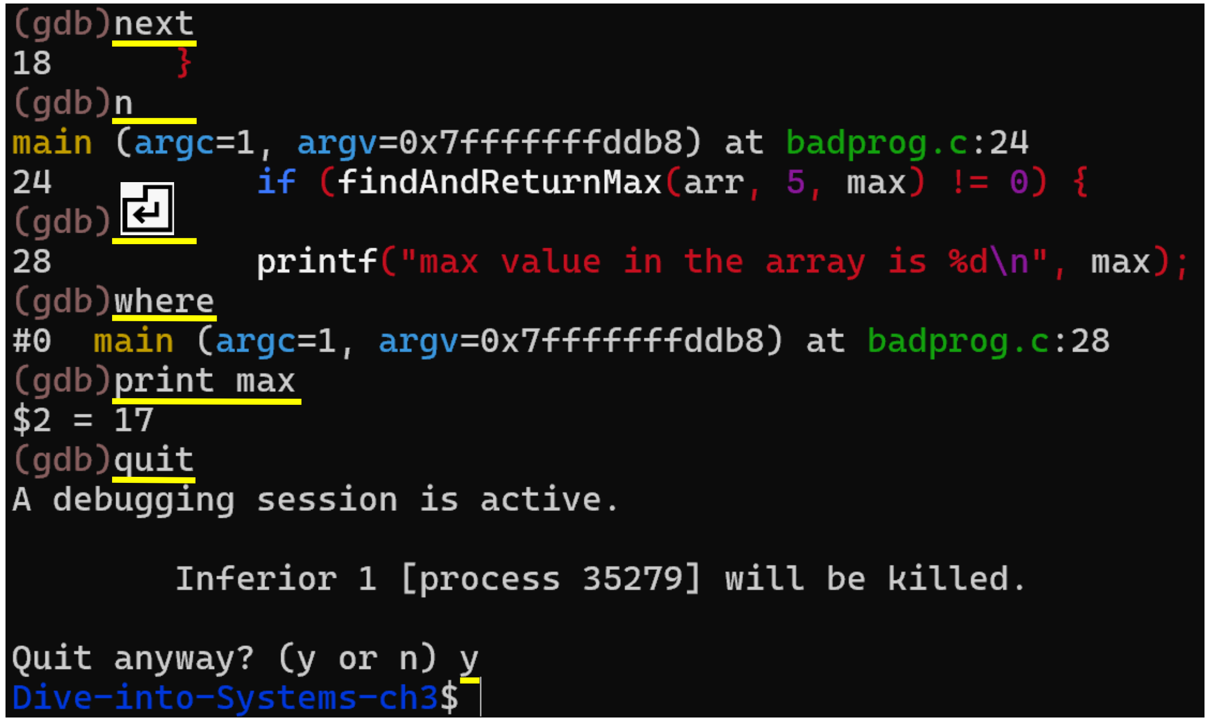
GDB Commands Uses

badprog.c

```
// badprog.c finds the largest element in the array
#include stdio.h
#include stdlib.h
#include string.h

int findAndReturnMax(int *arr, int len, int max) {
    int i;
    if (!arr || (len=0) )
        return -1;
    max = arr[0];
    for ( i = 1; i < len; i++ )
        if (max  arr[i])
            max = arr[i];
    return 0;
}

int main(int argc, char *argv[]) {
    int arr[5] = { 17, 21, 44, 2, 60 };
    int max = arr[0];

    if ( findAndReturnMax(arr, 5, max) != 0 ) {
        printf("strange error\n");
        exit(1);
    }
    printf("max value in the array is %d\n", max);

    return 0;
}
```

We found the second bug!

The max value doesn't return back to the main



```
(gdb)next
18          }
(gdb)n
main (argc=1, argv=0x7fffffffddb8) at badprog.c:24
24            if (findAndReturnMax(arr, 5, max) != 0) {
(gdb)
28            printf("max value in the array is %d\n", max);
(gdb)where
#0  main (argc=1, argv=0x7fffffffddb8) at badprog.c:28
(gdb)print max
$2 = 17
(gdb)quit
A debugging session is active.

        Inferior 1 [process 35279] will be killed.

Quit anyway? (y or n) y
Dive-into-Systems-ch3$
```

- How would you fix it? (hint: pass-by-pointer)

Exercise (1)

segfault.c

```c
/* Finds the largest element in the array */
int initfunc(int *array, int len) {
    for (int i=1; i < len; i++)
        array[i] = i;
    return 0;
}
int func(int *array1, int len, int max) {
    max = array1[0];
    for (int i=1; i < len; i++)
        if (max  array1[i])
            max = array1[i];
    return 0;
}
int main(int argc, char *argv[]) {
    int *arr = malloc(100);
    int max = 6;
    if (initfunc(arr, 100) != 0 ) {
        printf("init error\n");
        exit(1);
    }
    if ( func(arr, 100, max) != 0 ) {
        printf("func error\n");
        exit(1);
    }
    printf("max value in the array is %d\n", max);
    exit(0);
}
```

Practice with this case!

Compile / Execute	
$ gcc -g -o segfault segfault.c $ ./segfault	Expected Resut:100
	Executed Result: Segmentation fault

Exercise (2)

segfault.c

```c
/* Finds the largest element in the array */
int initfunc(int *array, int len) {
    for (int i=1; i < len; i++)
        array[i] = i;
    return 0;
}
int func(int *array1, int len, int max) {
    max = array1[0];
    for (int i=1; i < len; i++)
        if (max  array1[i])
            max = array1[i];
    return 0;
}
int main(int argc, char *argv[]) {
    int *arr = malloc(100);
    int max = 6;
    if (initfunc(arr, 100) != 0 ) {
        printf("init error\n");
        exit(1);
    }
    if ( func(arr, 100, max) != 0 ) {
        printf("func error\n");
        exit(1);
    }
    printf("max value in the array is %d\n", max);
    exit(0);
}
```

Debugging	
$ gdb ./segfault	-
(gdb) run	Program received signal SIGSEGV, Segmentation fault.
(gdb) where	#0 0x00005555555551b8 in initfunc (array=0x0, len=100) at segfault.c:8 #1 0x000055555555526b in main (argc=1, argv=0x7fffffffdda8) at segfault.c:21
(gdb) print i	$1 = 1
(gdb) print array[i]	Cannot access memory at address 0x4
(gdb) print array	$3 = (int *) 0x0
(gdb) frame 1	#1 0x000055555555526b in main (argc=1, argv=0x7fffffffdda8) at segfault.c:21 21 if (initfunc(arr, 100) != 0) {
(gdb) print arr	$4 = (int *) 0x0
(gdb) quit	

GDB Common Commands

Four Commands group :

- Controlling program execution
- Examine the execution point
- Manipulating breakpoints
- Printing program state and evaluating expressions

GDB Common Commands

- Controlling program execution

Commands	Arguments	Description
break, b	<func-name>	Set a breakpoint at start of <func-name>
	<line>	Set a breakpoint at <line>
run, r	- | <program args>	Start program running from the beginning
continue, cont	-	Continue execution of the program until it hits a breakpoint
step, s	- | <count>	Execute the next line or next <count> lines
next, n	- | <count>	Execute the next instruction or <count> instructions (function call is treated as a single line)
until	<line>	Execute the program until it reaches the specified <line>
quit, q	-	Exit GDB

GDB Common Commands

- Examine the execution point

Commands	Arguments	Description
list, l	- | <line> | <start> <end> | <func-name>	List program source code around <line> or beginning of <func-name> Also, can list <start> through <end>
where, backtrace, bt	-	Show the contents of the stack
frame	<frame-num>	Move into the context of stack frame number <frame-num>

GDB Common Commands

- Manipulating breakpoints

Commands	Arguments	Description
break, b	<func-name> | <line>	List program source code around <line> or beginning of <func-name> Also, can list <start> through <end>
enable	<bpnums …>	Disable one or more breakpoints
disable	<bpnums …>	Enable one or more breakpoints
ignore	<bpnum> <bum>	Don’t pause at <bpnum> the next <num> times it’s hit
delete	- | <bpnum>	Delete all or <bpnum> breakpoint
clear	<line> | <func-name>	Delete breakpoint at <line> or <func-name>
condition	<bpnum> <exp>	Set <bpnum> breakpoint to break only when <exp> is true

GDB Common Commands

- Printing program state and evaluating expressions

Commands	Arguments	Description
print, p	<exp>	Display the value of <exp> in different formats
display	<exp>	Display value of <exp> at every breakpoint
x	<memory address>	Display the contents of a <memory address>. Auto dereferencing.
whatis	<exp>	Display the data type of an <exp>
set	<variable> = <exp>	Sets <variable> to <exp> for assign/change the value
info	<status>	Lists information about program state and debugger state

Get Help

help	<topic> | <command>	Shows help documentation for topic or command

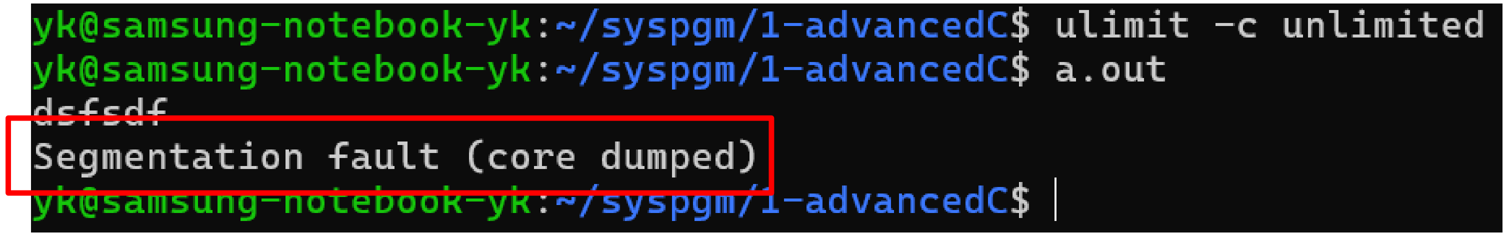
Core File

- Definition: Dump of Program including call stack, CPU registers, and information about others resources including open files and sockets at the time of program crach.
  - Developers use core files to investigate why a program crashed.
- How Core files are Generated
  - Generated automatically by the OS Kernel when a program crashes
- How to use core (Debugging with gdb)

  $ gdb <executable> <core>
- Managing Core: When Core file does not created
  - configuration file setup (/proc/sys/kernel/core_pattern)

  $ echo "./core" | sudo tee /proc/sys/kernel/core_pattern
  - checking core file size limt ($ ulimit –c 0 => Disable Core file dump)

  $ ulimit –c unlimited



```
yk@samsung-notebook-yk:~/syspgm/1-advancedC$ ulimit -c unlimited
yk@samsung-notebook-yk:~/syspgm/1-advancedC$ a.out
dsfsdf
Segmentation fault (core dumped)
yk@samsung-notebook-yk:~/syspgm/1-advancedC$
```

References

Dive into Systems (https://diveintosystems.org/)

- 3.1. Debugging with GDB (https://diveintosystems.org/book/C3-C_debug/gdb.html#_debugging_with_gdb)
- 3.2. GDB Commands in Detail (https://diveintosystems.org/book/C3-C_debug/gdb_commands.html#_gdb_commands_in_detail)

GDB cheetsheet (https://web2.qatar.cmu.edu/~msakr/15213-f08/recitations/gdbcheatsheet.txt)

Development Tool - GDB

(https://developer.apple.com/library/archive/documentation/DeveloperTools/gdb/gdb/gdb_9.html)