# LAB 6

## GDB

Department of Electronics Engineering
National Chiao Tung University

# Introduction to GDB

- GDB (GNU Debugger) is a debugging tool for UNIX system to debug C and C++ programs.

- Especially useful for segmentation fault , it can directly tell you where segmentation fault happened.

```
Program received signal SIGSEGV, Segmentation fault.
0x0000000000400d84 in child::printinfo (this=0x614c50) at demo.cpp:12
```

# How to execute

- To use GDB, we need to add the **-g** argument while compiling.
- Use **-q** argument can hide copyright related messages

```
17:44 2020PDA028@vda04 [~] >$ g++ -g demo.cpp
17:44 2020PDA028@vda04 [~] >$ gdb -q ./a.out
Reading symbols from /uhome/chome/2020PDA/2020PDA028/a.out...done.
(gdb)
```

```
17:49 2020PDA028@vda04 [~] >$ gdb ./a.out      without -q
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show c
and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /uhome/chome/2020PDA/2020PDA028/a.out...done.
(gdb)
```

# Common instructions

- You can use the full name or abbreviation to execute an instruction.
- list | l
- run | r
- breakpoint | break | b / continue | c
- next | n / step | s
- display | dis / print | p / watch
- info | I
- delete | d / quit | q

# List | l

- Command **l** can show the last 10 lines of code

```
(gdb) list 16
11              child(int a, string b) {value=a; id = b;}
12              void printinfo() {cout<<dad->value<<" "; cout<<dad->id;}
13              string id;
14              int    value;
15              child* dad;
16      };
17      int main(void)
18      {
19          child* y = new child(60,"root");
20          child* x = new child(50,"child");
(gdb)
```

# Run | r

- Use command **r** to start executing GDB
- Once a breakpoint is encountered or an error occurs, it will pause.

```
(gdb) r
Starting program: /uhome/chome/2020PDA/2020PDA028/./a.out
Program received signal SIGSEGV, Segmentation fault.
0x0000000000400d84 in child::printinfo (this=0x614c50) at demo.cpp:12
12              void printinfo() {cout<<dad->value<<" "; cout<<dad->id;}
(gdb)
```

```
(gdb) b main
Breakpoint 1 at 0x400b32: file demo.cpp, line 19.
(gdb) r
Starting program: /uhome/chome/2020PDA/2020PDA028/./a.out
Breakpoint 1, main () at demo.cpp:19
19              child* x = new child(50,"child");
(gdb)
```

# Breakpoint | b / Continue | c

- The way to set a breakpoint is

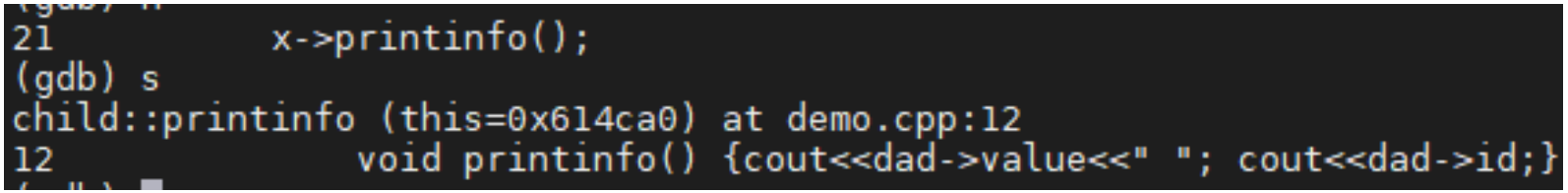
  **b** function name or **b** specific line
- You can use command **c** to continue execution from a pause caused by the breakpoint.

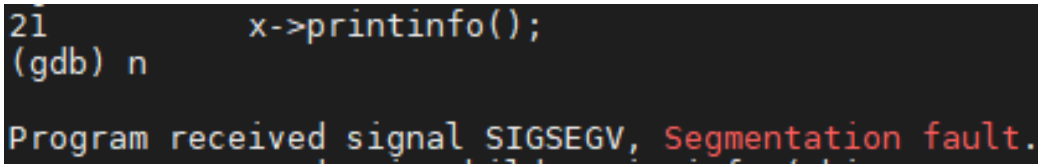
```
(gdb)
(gdb) b main
Breakpoint 1 at 0x400b32: file demo.cpp, line 19.
(gdb) b 21
Breakpoint 2 at 0x400bee: file demo.cpp, line 21.
(gdb)
```

```
(gdb) c
Continuing.
```

# Next | n / Step | s

- If you only want to execute one line of code each time, you can do it through **n / s**, don't need to set the breakpoint one by one.
- The difference between **n / s** is that once encounters the sub function, **s** will enter it while **n** will not.



# Display | d / print | p / watch

- Command **d/p** can show the value of certain variable
- The difference is that **d** will automatically show the value each step, while **p** needs to be entered manually
- **Watch** can keep trace a certain variable, once its value change, GDB will pause.

```
Breakpoint 1, main () at demo.cpp:19
19          child* y = new child(60,"root");
(gdb) p y->value
$1 = -461158174
(gdb) n
20          child* x = new child(50,"child");
(gdb) p y->value
$2 = 60
(gdb)
```

```
(gdb) print age_@5
$17 = {0, 65535, 1, -10352, 32767}
```

```
Breakpoint 4, main () at demo.cpp:19
19          child* y = new child(60,"root");
(gdb) display y->value
2: y->value = -461158174
(gdb) n
20          child* x = new child(50,"child");
2: y->value = 60
(gdb) n
21          x->printinfo();
2: y->value = 60
(gdb)
```

# Info | i

- Command **info** can display the current status of some settings
- More information : help info

```
(gdb) info b
Num     Type           Disp Enb Address            What
9       breakpoint     keep y   0x0000000000400b32 in main() at demo.cpp:19
10      breakpoint     keep y   0x0000000000400bee in main() at demo.cpp:21
11      breakpoint     keep y   0x0000000000400b32 in main() at demo.cpp:15
(gdb) info dis
Auto-display expressions now in effect:
Num Enb Expression
3:   y  x->id
2:   y  y->value
(gdb)
```

# Delete | d / Quit | q

- Command **d** can remove some previous settings
- Command **q** is used to terminate GDB

```
(gdb) info b
Num     Type           Disp Enb Address            What
9       breakpoint     keep y   0x0000000000400b32 in main() at demo.cpp:19
10      breakpoint     keep y   0x0000000000400bee in main() at demo.cpp:21
11      breakpoint     keep y   0x0000000000400b32 in main() at demo.cpp:15
(gdb) delete br 9
(gdb) info b
Num     Type           Disp Enb Address            What
10      breakpoint     keep y   0x0000000000400bee in main() at demo.cpp:21
11      breakpoint     keep y   0x0000000000400b32 in main() at demo.cpp:15
(gdb)
```

```
(gdb) q
A debugging session is active.

        Inferior 1 [process 21784] will be killed.

Quit anyway? (y or n) y
15:47 2020PDA028@vda04 [~] >$
```

# Other commands

- **where** can tell which layers and lines you are now
- **finish** can execute rest part of the function and return to the upper layer.
- **return** will directly return function of upper layer.

```
(gdb) where
#0  strlen () at ../sysdeps/x86_64/strlen.S:137
#1  0x000000000040116c in person::person (this=0x7fffffffd790, name_=0x10000
    at bug.cpp:54
#2  0x0000000000400f85 in main (argc=1, argv=0x7fffffffdb48) at bug.cpp:35
(gdb) ret
Make selected stack frame return now? (y or n) y
#0  0x000000000040116c in person::person (this=0x7fffffffd790,
    name_=0x10000ffff <Address 0x10000ffff out of bounds>, age_=0) at bug.cp
54          int length = strlen(name_);
(gdb) where
#0  0x000000000040116c in person::person (this=0x7fffffffd790, name_=0x10000
    at bug.cpp:54
#1  0x0000000000400f85 in main (argc=1, argv=0x7fffffffdb48) at bug.cpp:35
(gdb)
```

# Example (1/8)

- Running the following program will cause segmentation fault, so use GDB to debug.

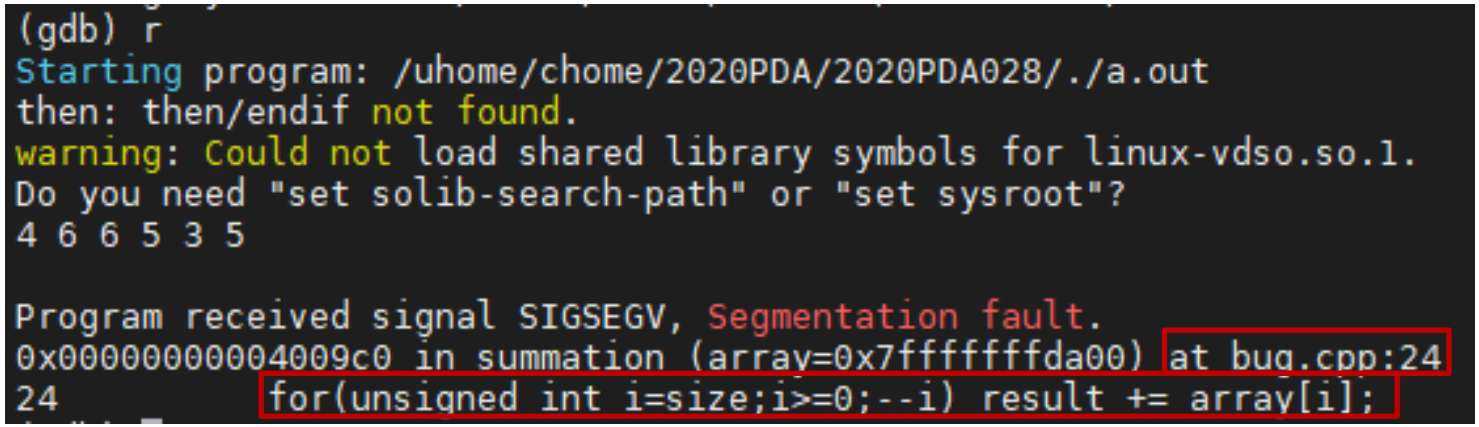
```
 6 #define size 6
 7 int summation(int array[]);
 8
 9 int main()
10 {
11     srand(time(NULL));
12     int array[size];
13     for(unsigned int i=0;i<size;++i) array[i] =  rand()%10+1;
14     for(unsigned int i=0;i<size;++i) cout<<array[i]<<" ";
15     cout<<endl;
16     int val=summation(array);
17     cout<<val<<endl;
18     return 0;
19 }
20
21 int summation(int array[])
22 {
23     int result=0;
24     for(unsigned int i=size;i>=0;--i) result += array[i];
25     return result;
26 }
```

```
22:19 2020PDA028@vda04 [~] >$ ./a.out
9 9 10 6 4 1
Segmentation fault (core dumped)
22:19 2020PDA028@vda04 [~] >$
```

```
g++ -g bug.cpp
gdb -q ./a.out
```

# Example (2/8)

- First, use **r** to find out which line cause segmentation fault.



# Example (3/8)

- Then, use **where** to find where we call the function,

  set a **breakpoint** and **restart**.

```
(gdb) where
#0  0x00000000004009c0 in summation (array=0x7fffffffda00) at bug.cpp:24
#1  0x0000000000400972 in main () at bug.cpp:16
(gdb) b 16
Breakpoint 1 at 0x400966: file bug.cpp, line 16.
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /uhome/chome/2020PDA/2020PDA028/./a.out
then: then/endif not found.
warning: no loadable sections found in added symbol-file system-supplied
warning: Could not load shared library symbols for linux-vdso.so.1.
Do you need "set solib-search-path" or "set sysroot"?
7 2 7 9 8 7

Breakpoint 1, main () at bug.cpp:16
16          int val=summation(array);
(gdb)
```

# Example (4/8)

- Use **s** enter sub-function, and use **watch i** to check value of i whenever it changes.

```
Breakpoint 1, main () at bug.cpp:16
16          int val=summation(array);
(gdb) s
summation (array=0x7fffffffda00) at bug.cpp:23
23          int result=0;
(gdb) s
24          for(unsigned int i=size;i>=0;--i) result += array[i];
(gdb) watch i
Hardware watchpoint 4: i
(gdb) c
Continuing.
Hardware watchpoint 4: i

Old value = 4156023511
New value = 6
0x00000000004009ae in summation (array=0x7fffffffda00) at bug.cpp:24
24          for(unsigned int i=size;i>=0;--i) result += array[i];
```

# Example (5/8)

- Find that -1 & unsigned cause this error.

```
(gdb) c
Continuing.
Hardware watchpoint 4: i

Old value = 1
New value = 0
0x00000000004009c9 in summation (array=0x7fffffffda00) at bug.cpp:24
24          for(unsigned int i=size;i>=0;--i) result += array[i];
(gdb) c
Continuing.
Hardware watchpoint 4: i

Old value = 0
New value = 4294967295
0x00000000004009c9 in summation (array=0x7fffffffda00) at bug.cpp:24
24          for(unsigned int i=size;i>=0;--i) result += array[i];
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x00000000004009c0 in summation (array=0x7fffffffda00) at bug.cpp:24
24          for(unsigned int i=size;i>=0;--i) result += array[i];
(gdb)
```

# Example (6/8)

- Remove unsigned part and compile again, find program can execute successfully, but final answer seems wrong.
- Therefore use **watch** to trace variable of sub-function.

```
(gdb) r
Starting program: /uhome/chome/2020PDA/2020PDA028/./a.out
then: then/endif not found.
warning: Could not load shared library symbols for linux-vdso.so.1.
Do you need "set solib-search-path" or "set sysroot"?
5 8 3 2 8 8
4196370
[Inferior 1 (process 27019) exited normally]
(gdb)
```

# Example (7/8)

- Find that due to the wrong index i, cause array read undefined space.
- So modify it to the correct index.

```
Breakpoint 1, main () at bug.cpp:16
16          int val=summation(array);
(gdb) s
summation (array=0x7fffffffda00) at bug.cpp:23
23          int result=0;
(gdb) s
24          for(int i=size;i>=0;--i) result +=
(gdb) watch result
Hardware watchpoint 6: result
(gdb) watch i
Hardware watchpoint 7: i
(gdb) c
Continuing.
```

```
(gdb) c
Continuing.
Hardware watchpoint 7: i

Old value = -138943785
New value = 6
0x00000000004009ae in summation (
24          for(int i=size;i>=0;-
(gdb) c
Continuing.
Hardware watchpoint 6: result

Old value = 0
New value = 4196336
```

# Example (8/8)

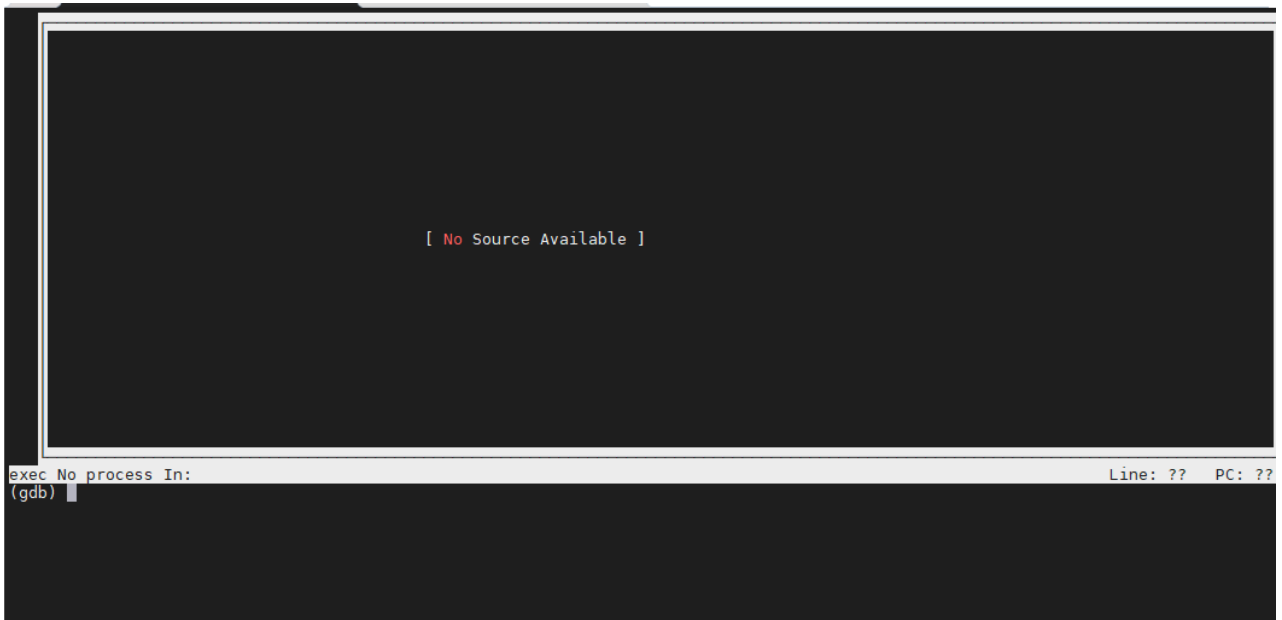
- Compile again, and find final answer is correct.

  => finish.

```
(gdb) r
Starting program: /uhome/chome/2020PDA/2020PDA028/./a.out
then: then/endif not found.
warning: Could not load shared library symbols for linux-vdso.so.1.
Do you need "set solib-search-path" or "set sysroot"?
8 6 6 6 7 10
43
[Inferior 1 (process 34719) exited normally]
(gdb)
```

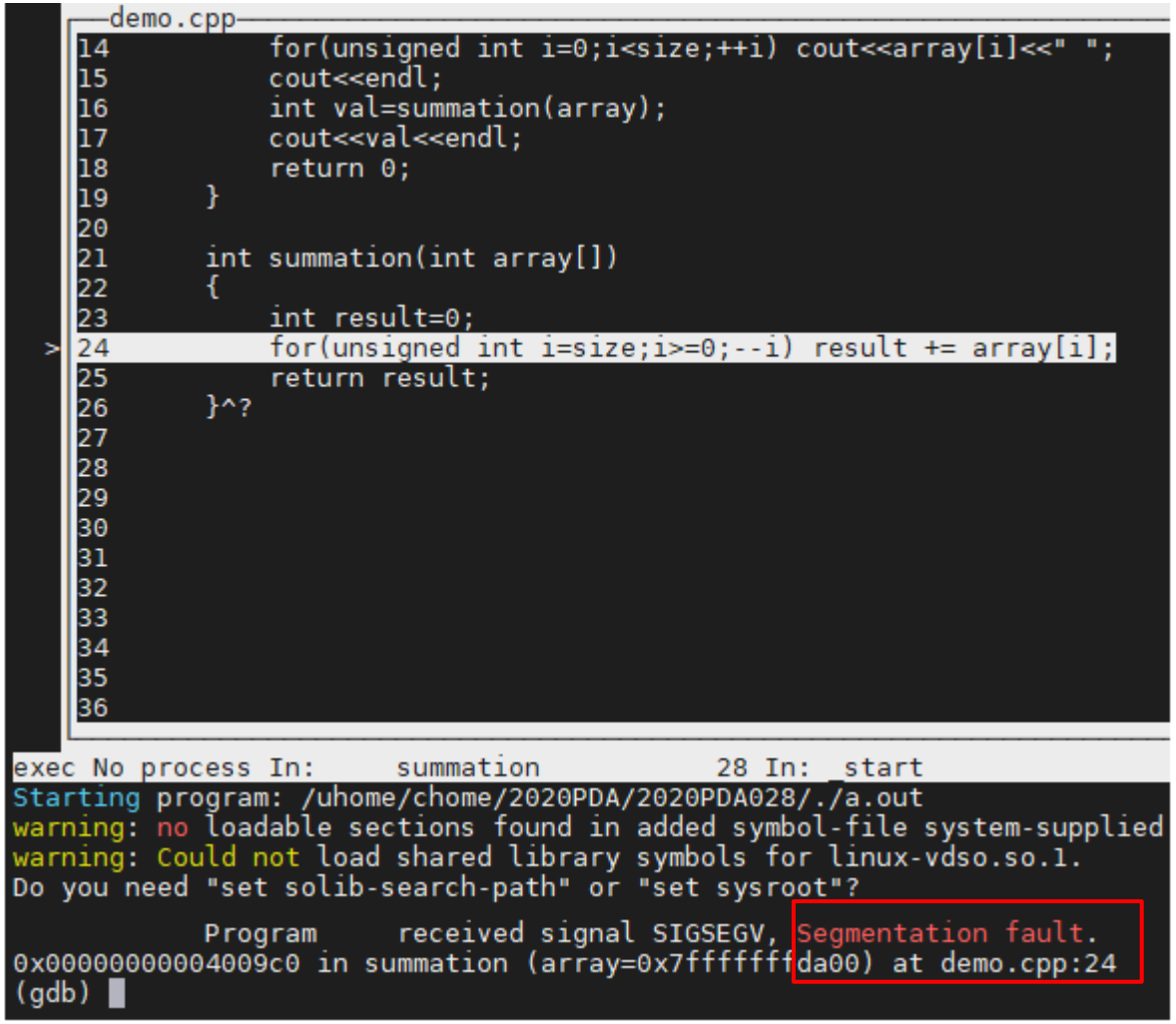
# GUI (1/5)

- You can also use GUI interface to get more intuitive information, all instructions are same as previous one.
- After gdb ./executable file, you can use ctrl+x+a to enter GUI interface.



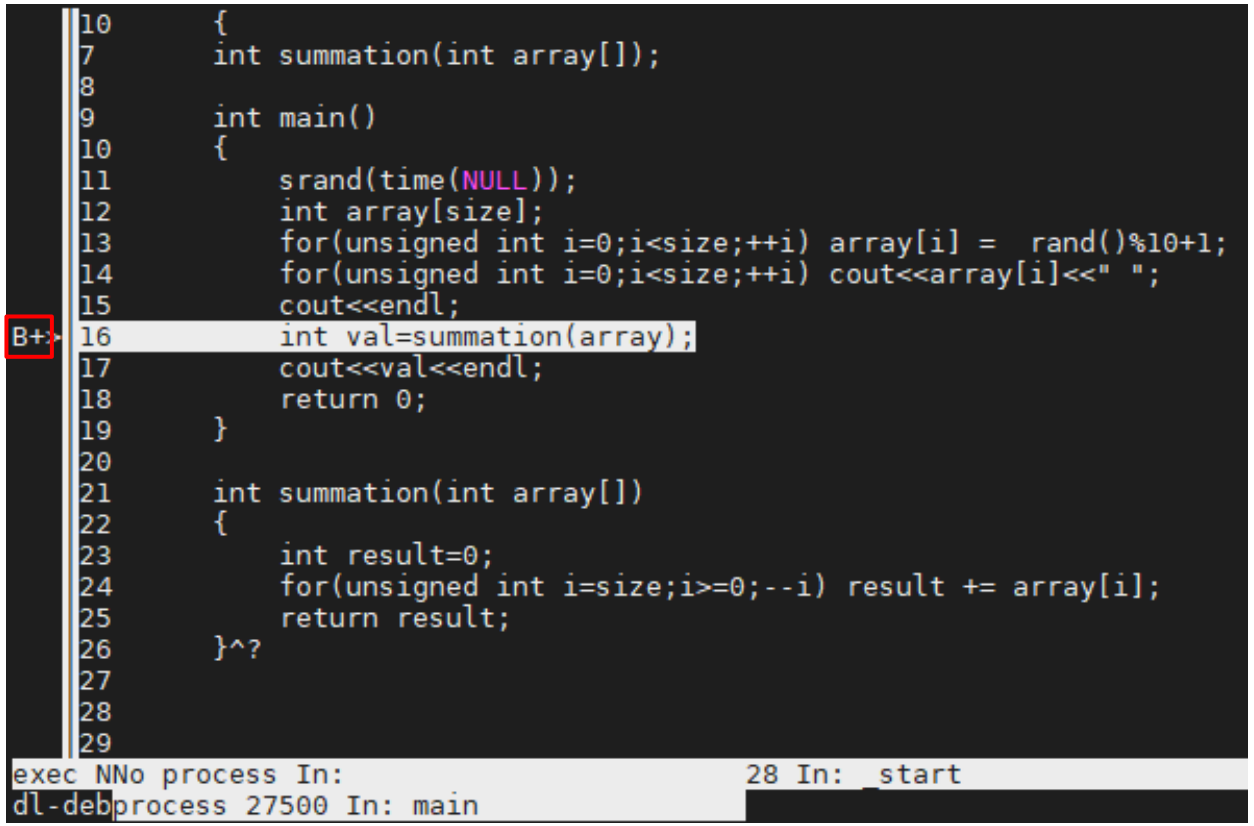
# GUI (2/5)

- Command **r**



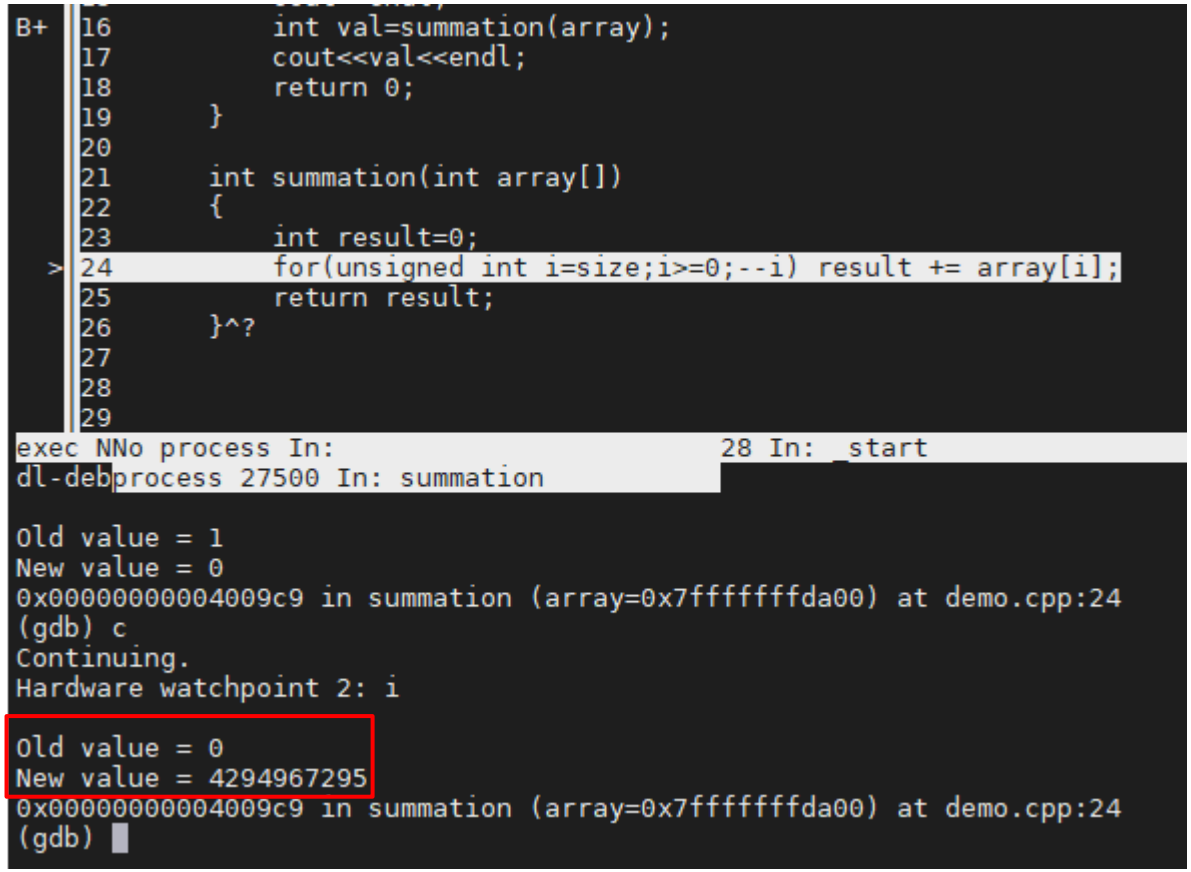
# GUI (3/5)

- Use where to find where sub-function was called.
- Set breakpoint and restart.



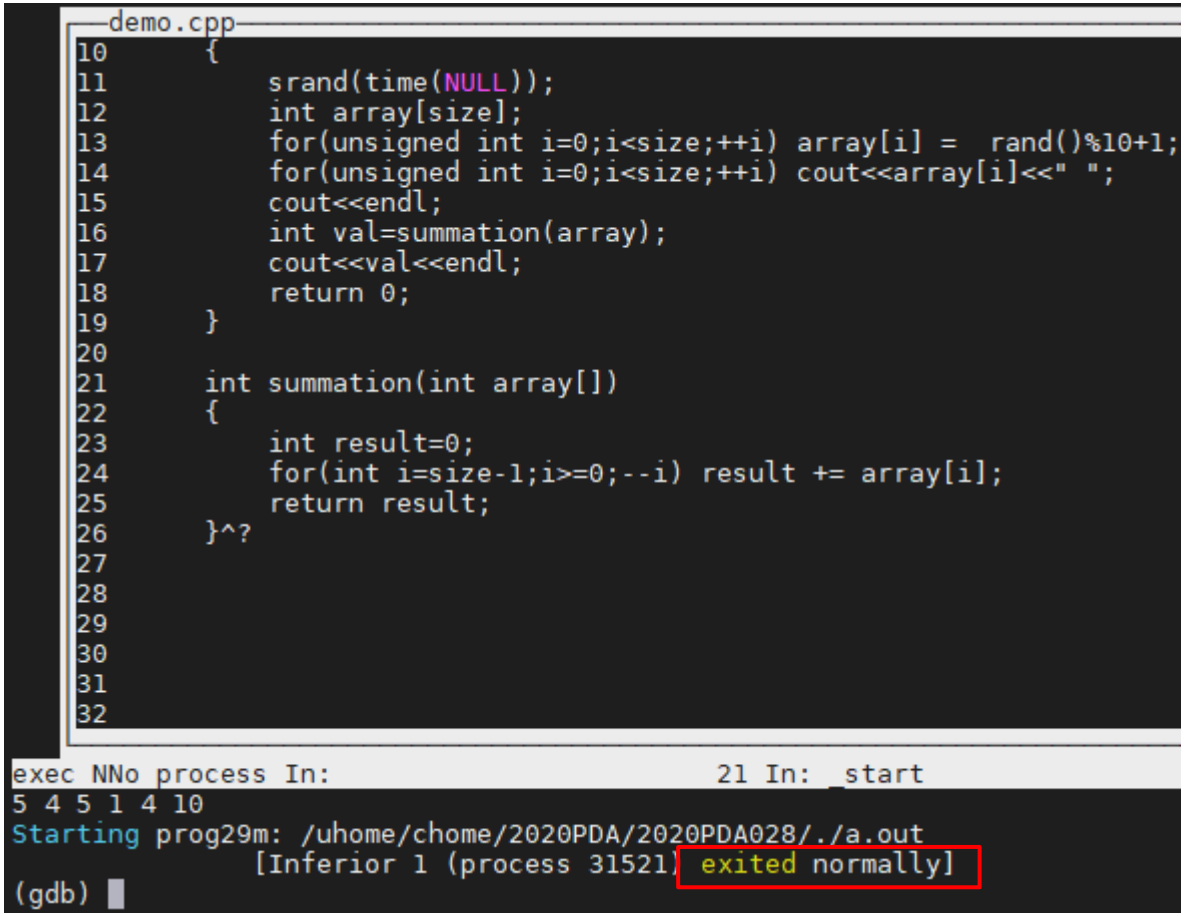
# GUI (4/5)

- Use watch to check the variable you want to trace.



# GUI (5/5)

- Repeat above steps, until all bugs are removed.



# Exercise

- Exercise – 1
- Exercise – 2

# Exercise 1 - Description

- Download the wrong.cpp from E3.
- Compile and run it.
- Fix the wrong code.
- **Demo 1. Show TA how you use GDB to find the bug, explain it.**
- **Demo 2. Fix the wrong code and show the correct result to TA.**

# Exercise

- Exercise – 1
- Exercise – 2

# Exercise 2-1 - Description

- In mathematics, a polynomial is an expression of finite length constructed from variables and constants, using only the operations of addition, subtraction, multiplication, and non-negative integer exponents.

- For example, $4.5x^2 - x + 5$ is a polynomial, but $x^2 + \frac{4}{x} + 3$ is not, because its second term's exponent is negative (-1).

- In this problem, your job is to implement the following polynomial member functions and **try to debug with GDB tool**.

- **In this exercise, you can only modify polynomial.cpp**

# Exercise 2-2 - Specification

- You must implement the PolySeq class with the following public member functions and friend function:

| Function | Description |
|---|---|
| ~PolySeq( ) | Destructor. |
| istream &operator>>(); | Read in the polynomial coefficient. |
| operator+(); | Return p1 + p2. |
| operator*(); | Return p1 * p2; |
| operator=(); | Assignment ex: p3 = p1 + p2. |
| Derivative(); | Return the derivative of polynomial function. |
| double Integral(int, int); | Return the result of definite integral. $\int_{low\_bound}^{up\_bound} P(x)dx$ |
| double getValue(double); | Return the result of the polynomial with the specified parameter. |

# Exercise 2-3 - INPUT

```
3                 #The highest degree of P1.
3 -2 1 0          #The coefficient of P1.(x^3, x^2, x^1, x^0) respectively.
2                 #The highest degree of P2.
9 -4 1            #The coefficient of P2.(x^2, x^1, x^0) respectively.
2 3               #x1, x2
```

# Exercise 2-4 - Output

- The output should print the following integers in order.

  (1) The sum of the first and the second polynomials with parameter x1.

  (2) The product of the first and the second polynomials with parameter x1.

  (3) The derivative of the first polynomial with parameter x1.

  (4) The result of the definite integral of the second polynomial with parameter lower bound x1 and upper bound x2.

| Sample Output | |
|---|---|
| 47 | p1+p2, with x=x1 |
| 522 | p1*p2, with x=x1 |
| 29 | the derivative of p1, with x=x1 |
| 48 | the integral of p2, with x1 and x2 |

# **Exercise 2-5** Compile & Run & Demo

- Compile
  - g++ -std=c++11 main.cpp polynomial.cpp -I . -o Lab06
  - or using makefile
- Run
  - ./Lab06 [input filename]
- Ask TA for Demo