

實驗一 Linux environment & debugging tool

9617145 許晏峻 資工 4C

Project 1:

```
(gdb) p i
$7 = 11
(gdb) l
12
13     void get_args(int ac, char **av)
14     { int i;
15
16         num_inputs = ac - 1;
17         for (i = 0; i < num_inputs; i++)
18             x[i] = atoi(av[i+1]);
19     }
20
21     void scoot_over(int jj)
(gdb) l 8
3 //
4 // usage: insert_sort num1 num2 num3 ..., where the numi are the n
5 // be sorted
6 #include <stdio.h>
7
8     int x[10], // input array
9         y[10], // workspace array
10        num_inputs, // length of input array
11        num_y = 0; // current number of elements in y
12
(gdb)
0 --- [1] --- 2 --- 3 ---
[ hEaD's WoRkStAtIoN ]
```

上圖：當 $i=11$ 時，第 18 行 assign $x[i]$ 值，但在第 8 行宣告 x 空間只有 10，因此會 stack overflow

```
50         for (num_y = 0; num_y < num_inputs; num_y++)
(gdb) c
Continuing.

Breakpoint 2, insert (new_y=1597) at insert_sort.c:31
31         if (num_y == 0) { // y empty so far, easy case
(gdb) p num_inputs
$1 = 16
(gdb)
```

上圖：迴圈判斷 $\text{num_y} < \text{num_inputs}$ ($\$1=16$)，但迴圈中每次都會 if 判斷 $\text{num_y}=0$ ，少了一個 $=$ ，變成 assign $\text{num_y}=0$ ，因此會無窮迴圈

```
(gdb) s
24      for (k = num_y-1; k > jj; k++)
(gdb) p k
$2 = 1
(gdb) p jj
$3 = 0
(gdb)
```

上圖：迴圈判斷 $k > jj$ ，但每次 $k+1$ ，會陷入無窮迴圈，應改成 $k--$

```
(gdb) b 67
Breakpoint 1 at 0x4007a5: file insert_sort.c, line 67.
(gdb) r 1597 144 3 8 610 2 233 34 21 5 13 55 89 377 1 987
Starting program: /amd/cs/96/9617145/MP/lab1/insert_sort/insert_sort 1597

Breakpoint 1, main (argc=17, argv=0x7fffffe3c8) at insert_sort.c:67
67      print_results();
(gdb) c
Continuing.
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597

Program exited with code 020.
(gdb) l
62      }
63
64      int main(int argc, char ** argv)
65      {
66          get_args(argc,argv);
67          process_data();
68          print_results();
69      }
```

上圖：中斷點設在最後發現還有錯誤，因為 `int main()` 沒有回傳一個 `int`，加上 `return 0;` 即可

Project 2:

```
(gdb) r 16
Starting program: /amd/cs/96/9617145/MP/lab1/fibonacci_list/fibonacci 16

Breakpoint 1, fibonacci (n=16) at fibonacci.c:32
32      return (Fib[n]-fibonacci(n) + fibonacci(n-1));
(gdb)
```

上圖：`fibonacci(n=16)` 中又 call 一次 `fibonacci(n=16)`，很明顯會造成無限遞迴

```

Breakpoint 1, main (argc=2, argv=0x7fffffff460) at fibonacci.c:17
17         for (i=0; i<=N; i++) ;
(gdb) s
18         printf("%d, ", fibonacci(i));
(gdb) n i
$1 = 17
(gdb)

```

上圖：for(...)後多加了一個；，導致下一行的 printf(...)變成迴圈執行完才執行到，此時 i=17，已超過預計應該印出的陣列位置 Fib[0]~Fib[16]

Project 3:

```

Breakpoint 1, is_prime (p=11) at prime.c:18
18         for(i = 3; i <= to; i+=3) {
(gdb) c
Continuing.

Breakpoint 2, is_prime (p=11) at prime.c:20
20         return 0;
(gdb) l
15
16         if ((p & 1)==0)
17             return 0;
18         for(i = 3; i <= to; i+=3) {
19             if ((p % i)==0);
20             return 0;
21         }

```

上圖：11 是質數，但卻 return 0，是錯誤的，因為第 19 行的 if 多加了一個；，導致 return 0 一定會被執行到

```

Breakpoint 2, is_prime (p=25) at prime.c:19
19         if ((p % i)==0)
(gdb) s
18         for(i = 3; i <= to; i+=3) {
(gdb) s
22         return 1;
(gdb) p i
$1 = 6
(gdb) p to
$2 = 5
(gdb)

```

上圖：25 不是質數，但卻 return 1，是錯誤的，因為第 18 行的地方 i+=3，應該是做 i+=2 判斷每個奇數做(p%i)才對

```

Breakpoint 1, main (argn=2, argv=0x7fffffff4f0) at prime.c:65
65     for (i = 0; i < total; j++); {
(gdb) l
60     work ((void *) id);
61
62
63     printf ("Number of prime numbers between 2 and %d: %d\n", i, total);
64
65     for (i = 0; i < total; j++; {
66         printf ("%d ", primes[i]);
67     }
68
69     printf("\n");
(gdb) c
Continuing.

```

上圖：在第 65 行後就進入無窮迴圈，因為迴圈判斷式用 $i < \text{total}$ ，但每次的 control 是 $j++$ ，應改成 $i++$ 。此外，for(...)後也多加一個; 因此 printf(...)會再迴圈結束後才印出，應該拿掉

```

Number of prime numbers between 2 and 1024 171
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71
233 239 241 251 257 263 269 271 277 281 283 293 307 311
87 491 499 503 509 521 523 541 547 557 563 569 571 577
769 773 787 797 809 811 821 823 827 829 839 853 857 859

Program exited normally.
(gdb) l 40
35     if ((start % 2) == 0)
36         start++;
37     for (i = start; i < Max_num; i += 2) {
38         if (is_prime (i)) {
39             total++;
40             primes[total] = i;
41         }
42     }
43     return NULL;
44 }
(gdb)

```

上圖：程式已經可以正常執行，答案也正確，但是卻少印一個質數，因為第 39 行的 $\text{total}++$ ，少算第一個質數 $\text{primes}[0]=2$ ，或著也可以說 total 代表的是最後一個 prime 的陣列位置，因此質數數量應是 $\text{total}+1$ ，所以在迴圈結束後可再加一個 $++\text{total}$