

# Tools to evaluate your program

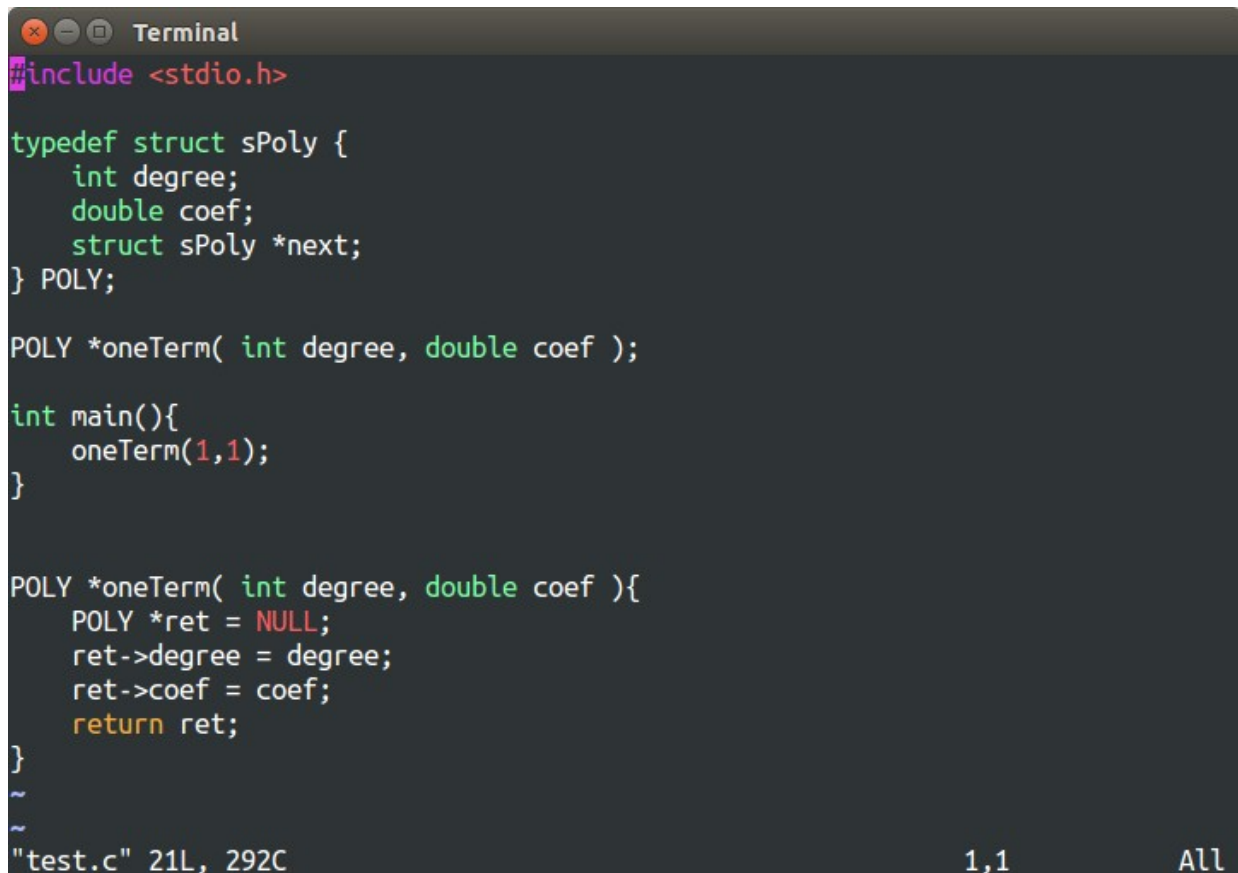
## 1. gdb: Trace program and find the segmentation fault

### a) Command

- run: Run the program
- print: Print the variable

### b) Example

- This program does not allocate memory space
- Cause "Segmentation fault"



```
Terminal
#include <stdio.h>

typedef struct sPoly {
    int degree;
    double coef;
    struct sPoly *next;
} POLY;

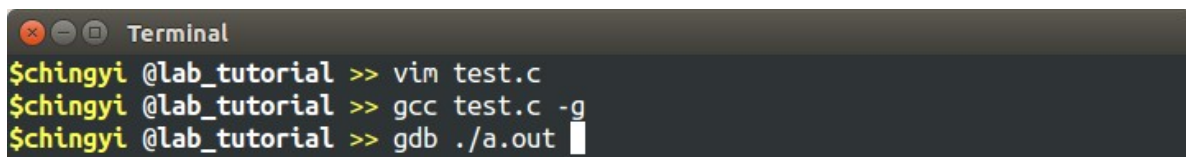
POLY *oneTerm( int degree, double coef );

int main(){
    oneTerm(1,1);
}

POLY *oneTerm( int degree, double coef ){
    POLY *ret = NULL;
    ret->degree = degree;
    ret->coef = coef;
    return ret;
}
~
~
"test.c" 21L, 292C                                1,1                                All
```

- Debugging procedure

- Re-compile with argument -g. For example: gcc lab11.c -g
- Call gdb, and the argument is the name of the executable



```
Terminal
$chingyi @lab_tutorial >> vim test.c
$chingyi @lab_tutorial >> gcc test.c -g
$chingyi @lab_tutorial >> gdb ./a.out
```

- You will enter gdb interface

```

Terminal
$chingyi @lab_tutorial >> gdb ./a.out
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 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:
<http://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...done.
(gdb)

```

- Type command "run", and press enter
- You will get the line with segmentation fault
  - You can print out the variable to check variable value
    - In "print ret", the value is (POLY\*) 0x0, which represents NULL
    - In "print ret->degree", it shows "Cannot access memory at address 0x0", which means you cannot use this variable (The regular reason is without malloc())

```

Terminal
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:
<http://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...done.
(gdb) run
Starting program: /home/chingyi/lab_tutorial/a.out

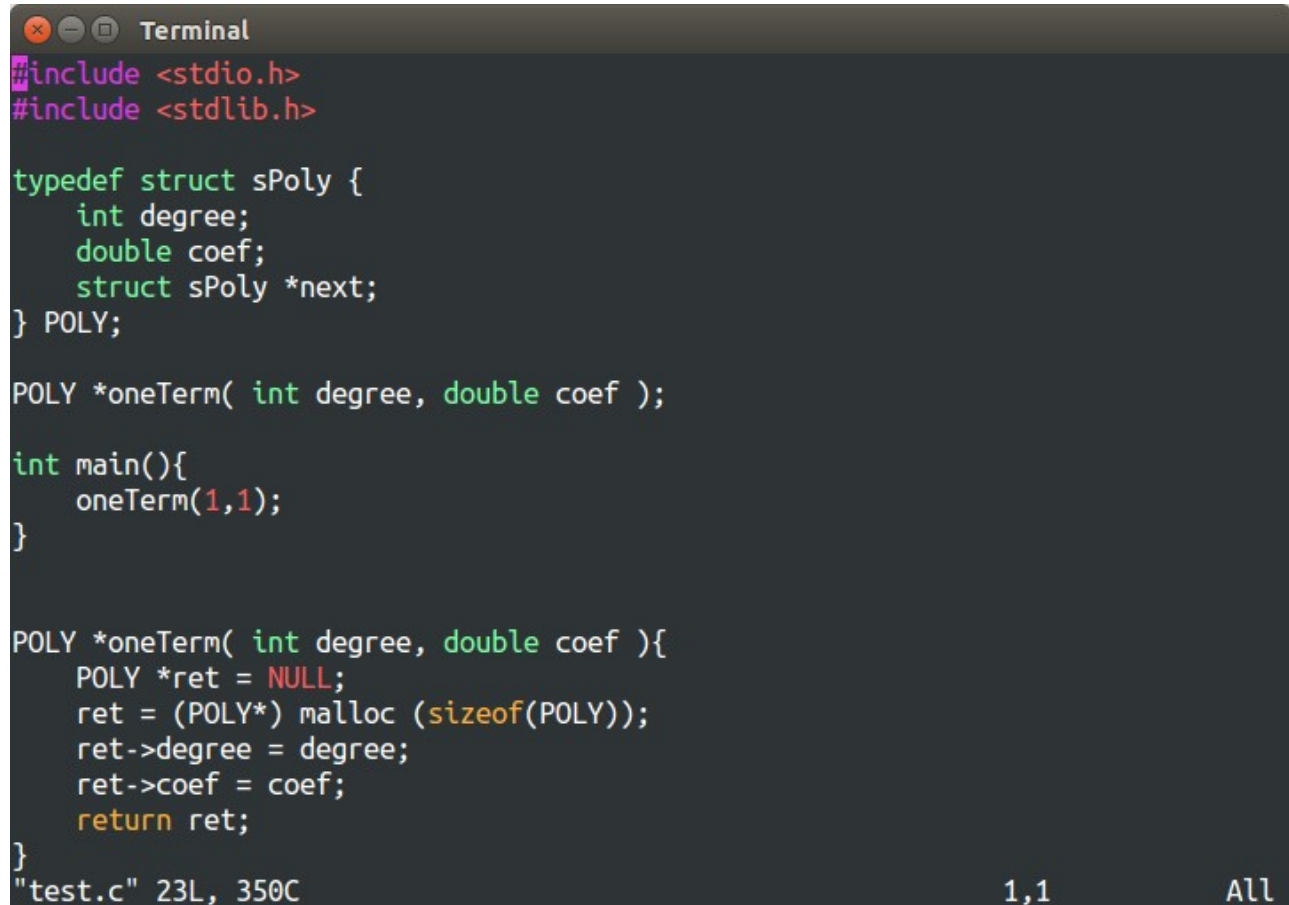
Program received signal SIGSEGV, Segmentation fault.
0x000000000400520 in oneTerm (degree=1, coef=1) at test.c:18
18      ret->degree = degree;
(gdb) print ret
$1 = (POLY *) 0x0
(gdb) print ret->degree
Cannot access memory at address 0x0
(gdb)

```

## 2. valgrind: Check memory leakage and invalid access

### a) Example for memory leakage

- This is a program without free the node



```
Terminal
#include <stdio.h>
#include <stdlib.h>

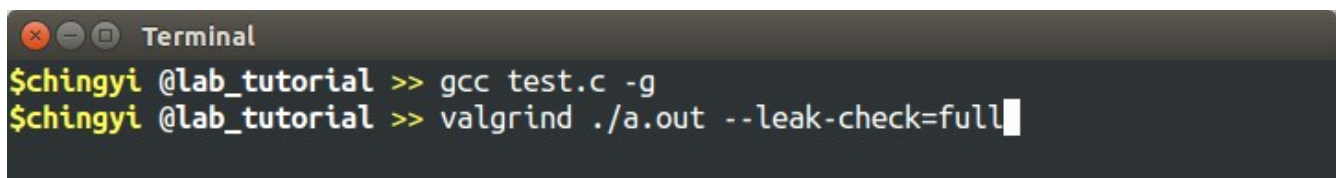
typedef struct sPoly {
    int degree;
    double coef;
    struct sPoly *next;
} POLY;

POLY *oneTerm( int degree, double coef );

int main(){
    oneTerm(1,1);
}

POLY *oneTerm( int degree, double coef ){
    POLY *ret = NULL;
    ret = (POLY*) malloc (sizeof(POLY));
    ret->degree = degree;
    ret->coef = coef;
    return ret;
}
"test.c" 23L, 350C                                     1,1      All
```

- Re-compile with "-g" also
- Use the tool "valgrind". There are two arguments, first is the name of executable, usually "./a.out", second is "--leak-check=full"



```
Terminal
$chingyi @lab_tutorial >> gcc test.c -g
$chingyi @lab_tutorial >> valgrind ./a.out --leak-check=full
```

- There will be a leak summary
  - Ideally, there should be all 0

```
Terminal
$chingyi @lab_tutorial >> gcc test.c -g
$chingyi @lab_tutorial >> valgrind ./a.out --leak-check=full
==5278== Memcheck, a memory error detector
==5278== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==5278== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==5278== Command: ./a.out --leak-check=full
==5278==
==5278== HEAP SUMMARY:
==5278==   in use at exit: 24 bytes in 1 blocks
==5278==   total heap usage: 1 allocs, 0 frees, 24 bytes allocated
==5278==
==5278== LEAK SUMMARY:
==5278==   definitely lost: 24 bytes in 1 blocks
==5278==   indirectly lost: 0 bytes in 0 blocks
==5278==   possibly lost: 0 bytes in 0 blocks
==5278==   still reachable: 0 bytes in 0 blocks
==5278==   suppressed: 0 bytes in 0 blocks
==5278== Rerun with --leak-check=full to see details of leaked memory
==5278==
==5278== For counts of detected and suppressed errors, rerun with: -v
==5278== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
$chingyi @lab_tutorial >>
```

### 3. Checking uninitialized

- a) Example for accessing uninitialized value
  - In this program, it used head->degree without initialization

```
Terminal
#include <stdio.h>
#include <stdlib.h>

typedef struct sPoly {
    int degree;
    double coef;
    struct sPoly *next;
} POLY;

POLY *oneTerm( int degree, double coef );

int main(){
    POLY *head;
    head = oneTerm(1,1);
    printf("head degree = %d", head->degree);
    return 0;
}

POLY *oneTerm( int degree, double coef ){
    POLY *ret = NULL;
    ret = (POLY*) malloc (sizeof(POLY));
    return ret;
}
```

3,0-1 All

- There will be error in valgrind

```
Terminal
==5811==
==5811== Conditional jump or move depends on uninitialised value(s)
==5811==    at 0x4E8161C: vfprintf (vfprintf.c:1660)
==5811==    by 0x4E8B3D8: printf (printf.c:33)
==5811==    by 0x4005B1: main (test.c:15)
==5811==
==5811== head->degree = 0
```

- Ideally also, the ERROR SUMMARY should have no errors

```
==5811== Rerun with --leak-check=full to see details of leaked memory
==5811==
==5811== For counts of detected and suppressed errors, rerun with: -v
==5811== Use --track-origins=yes to see where uninitialised values come from
==5811== ERROR SUMMARY: 6 errors from 6 contexts (suppressed: 0 from 0)
$chingyi @lab_tutorial >> 
```