Guida al debugging

Luca Versari

Aprile 19, 2017

Indice

Compilazione

Compilazione standard

```
$ gcc programma.c -o programma
```

Compilazione con (quasi) tutti i warning

```
$ gcc -O2 -Wall programma.c -o programma
```

Compilazione per il debugging

```
$ gcc -ggdb programma.c -o programma
```

Compilazione per il debugging in C++

```
$ g++ -ggdb -D_GLIBCXX_DEBUG programma.cpp -o programma
```

Errori e warning comuni

Codice

```
1  #include <stdio.h>
2  int main() {
3      int a;
4      printf("%d\n", a);
5  }
```

Errori e warning comuni

Codice

```
1  #include <stdio.h>
2  int main() {
3      int a;
4      printf("%d\n", a);
5  }
```

Warning

```
prova.c: In function ‘main’:
prova.c:4:5: warning: ‘a’ is used uninitialized in
this function [-Wuninitialized]
```

Errori e warning comuni

Codice

```
1  #include <stdio.h>
2  int main() {
3      int a = 4;
4      if (a = 5) printf("%d\n", a);
5  }
```

Errori e warning comuni

Codice

```c
#include <stdio.h>
int main() {
    int a = 4;
    if (a = 5) printf("%d\n", a);
}
```

Warning

```
prova.c: In function ‘main’:
prova.c:4:5: warning: suggest parentheses around
assignment used as truth value [-Wparentheses]
```

Errori e warning comuni

Codice

```
1  #include <stdio.h>
2  int main() {
3      int a = 0;
4      scanf("%d\n", a);
5  }
```

Errori e warning comuni

Codice

```
1  #include <stdio.h>
2  int main() {
3      int a = 0;
4      scanf("%d\n", a);
5  }
```

Warning

```
prova.c: In function ‘main’:
prova.c:4:29: warning: format ‘%d’ expects argument
of type ‘int *’, but argument 2 has type ‘int’
[-Wformat=]
```

Errori e warning comuni

Codice

```
1  #include <stdio.h>
2  int main() {
3      int a = 5;
4      if (a == 5)
5          a++;
6          printf("%d\n", a);
7  }
```

Errori e warning comuni

Codice

```c
#include <stdio.h>
int main() {
    int a = 5;
    if (a == 5)
        a++;
        printf("%d\n", a);
}
```

Warning

```
prova.c: In function ‘main’:
prova.c:4:5: warning: this ‘if’ clause does not guard...
[-Wmisleading-indentation]
prova.c:6:9: note: ...this statement, but the latter is misleadingly
indented as if it is guarded by the ‘if’
```

Errori e warning comuni

Codice

```
1  #include <stdio.h>
2  int main() {
3      int a = 4
4      if (a == 5) printf("%d\n", a);
5  }
```

Errori e warning comuni

Codice

```
1  #include <stdio.h>
2  int main() {
3      int a = 4
4      if (a == 5) printf("%d\n", a);
5  }
```

Errore

```
prova.c: In function ‘main’:
prova.c:4:5: error: expected ‘,’ or ‘;’ before ‘if’
```

assert

Codice

```
1  #include <assert.h>
2  int main() {
3      int a = 4;
4      assert(a != 4);
5  }
```

Output

```
prova: prova.c:5: main: Assertion ‘a != 4’ failed.
Aborted (core dumped)
```

assert(0)

Codice

```
1  #include <stdio.h>
2  #include <assert.h>
3  int main() {
4      int a = -1;
5      if (a % 2 == 0) printf("pari\n");
6      else if (a % 2 == 1) printf("dispari\n");
7      else assert(0);
8  }
```

Output

```
prova: prova.c:7: main: Assertion '0' failed.
Aborted (core dumped)
```

assert(0) con messaggio

Codice

```
1  #include <stdio.h>
2  #include <assert.h>
3  int main() {
4      int a = -1;
5      if (a % 2 == 0) printf("pari\n");
6      else if (a % 2 == 1) printf("dispari\n");
7      else assert(!!!"messaggio");
8  }
```

Output

```
prova: prova.c:7: main: Assertion '!!!"messaggio"' failed.
Aborted (core dumped)
```

Macro utile

Codice

```
1  #include <stdio.h>
2  #define debug(fmt, ...) fprintf(stderr, \
3      __FILE__ ", line %d, function %s: " fmt \
4      "\n", __LINE__, __func__,  __VA_ARGS__);
5  int main() {
6      int a = 5;
7      debug("a = %d", a);
8  }
```

Output

```
prova.c, line 7, function main:  a = 5
```

Indice

Invocazione

Compilazione e invocazione normale

```
$ gcc programma.c -o programma
$ ./programma argomento < input.txt
```

Compilazione e invocazione con gdb

```
$ gcc programma.c -ggdb -o programma
$ gdb ./programma
GNU gdb (GDB) 7.12.1
[...]
(gdb) run argomento < input.txt
```

Primo esempio di uso di gdb

Codice

```
1  void print_list(struct list_t* L) {
2      struct list_t* ptr;
3      for (ptr = L; ptr != NULL; ptr = ptr->next)
4          printf("%d\n", ptr->val);
5  }
6  L->next->next = rand(); print_list(L); // L lista
```

GDB

```
$ gdb ./prova
GNU gdb (GDB) 7.12.1
[...]
(gdb) run
Program received signal SIGSEGV, Segmentation fault.
0x0000000000345234 in print_list (L=0x24eb0) at prova.c:4
4       printf("%d\n", ptr->val);
```

Secondo esempio di uso di gdb

Codice

```
1  int f(int a) {
2      if (a > 0) return f(a-1)+1;
3      int* b;
4      return *b;
5  }
6  int main() {printf("%d\n", f(10000));}
```

GDB

```
$ gdb ./prova
GNU gdb (GDB) 7.12.1
[...]
(gdb) run
Program received signal SIGSEGV, Segmentation fault.
0x000000000040051d in f (a=0) at prova.c:4
4       return *b;
```

Stack trace

GDB

```
$ gdb ./prova
GNU gdb (GDB) 7.12.1
[...]
(gdb) run
Program received signal SIGSEGV, Segmentation fault.
0x000000000040051d in f (a=0) at prova.c:4
4       return *b;
(gdb) backtrace
#0 0x000000000040051d in f (a=0) at prova.c:4
#1 0x0000000000400514 in f (a=1) at prova.c:2
#2 0x0000000000400514 in f (a=2) at prova.c:2
#3 0x0000000000400514 in f (a=3) at prova.c:2
#4 0x0000000000400514 in f (a=4) at prova.c:2
#5 0x0000000000400514 in f (a=5) at prova.c:2
[...]
```

Comandi utili

Comandi utili

`break file:line`	mette in pausa l'esecuzione a `file:linea`.
`break funzione`	mette in pausa quando si esegue `funzione`.
`continue`	riprende l'esecuzione
`step`	esegue la riga successiva
`next`	come `step`, saltando chiamate a funzione
`watch variabile`	mette in pausa quando cambia `variabile`
`print espressione`	stampa `espressione`
`display espressione`	stampa automaticamente `espressione`
`quit`	esce da gdb

Inoltre, `break ... if condizione` o `watch ... if condizione` interrompono l'esecuzione solo se vale `condizione`.

Esempio

Esempio

```
(gdb) break f if a == 0
Breakpoint 1 at 0x400501:  file prova.c, line 2.
(gdb) run
Starting program:  /home/luca/tmp/prova
Breakpoint 1, f (a=0) at prova.c:2
2       if (a > 0) return f(a-1)+1;
(gdb) step
4       return *b;
(gdb) print b
$1 = (int *) 0x0
(gdb) step
Program received signal SIGSEGV, Segmentation fault.
0x000000000040051d in f (a=0) at prova.c:4
4       return *b;
```

Indice

Controllo accesso a variabili non inizializzate

Memcheck

```
$ valgrind --tool=memcheck ./prova
==5519== Memcheck, a memory error detector
[...]
==5519== Command:  ./prova
==5519==
==5519== Use of uninitialised value of size 8
==5519== at 0x40051D: f (prova.c:4)
==5519== by 0x400513:  f (prova.c:2)
==5519== by 0x400513:  f (prova.c:2)
[...]
```

Controllo tempi di esecuzione

Callgrind

```
$ valgrind --tool=callgrind ./prova
==5693== Callgrind, a call-graph generating cache profiler
[...]
$ kcachegrind callgrind.out.5693
```

Apre una finestra con i tempi di esecuzione di ogni funzione, il grafo delle chiamate tra funzioni e altre informazioni utili.