Laboratorium Informatyka (I) AiR
Ćwiczenia z debugowania

Krzysztof Kluza, Janusz Miller

1 Debugowanie

Debugowanie, czy też po polsku odpluskiwanie, to proces polegający na kontrolowanym wykonaniu programu przy użyciu odpowiedniego narzędzia (tzw. debuggera) w celu redukcji błędów występujących w programie.

Jeśli zamierzasz debugować swój program, oznacza to, że nie działa on poprawnie. Najlepiej byłoby zatem pisać program tak, aby dobrze działał. Aby nie dopuścić do powstania sytuacji, w których konieczne byłoby debugowanie, warto pisać program:

- po dokładnym przemyśleniu jego budowy i sposobu działania,
- w taki sposób, aby jego kod był zrozumiały i czytelny, gdyż wtedy o wiele prościej będzie znaleźć w nim błędy.

Często na zajęciach (np. podczas oddawania programów) można odnieść wrażenie, że ignorowane jest działanie programu, natomiast dużą wagę przykłada się do kodu. Wiąże się to z odpowiedzią na pytanie, co jest ważniejsze: czytelność programu czy to, że poprawnie działa? Jeśli program jest czytelny, można łatwo naprawić błędy, tak aby dobrze działał. W przypadku nieczytelnego programu trudno w ogóle mówić o pewności, że działa poprawnie.

1.1 Kompilacja

Dla początkujących programistów sukcesem jest sam fakt, że program się kompiluje i działa. Jednakże dobrze napisany kod powinien być estetyczny i zrozumiały, ale przede wszystkim wolny od błędów. Zatem kompilacja to dopiero początek drogi...

Jednak zacząć należy od kompilacji – zawsze warto używać opcji: `-Wall -ansi -pedantic` – zostaną wtedy wyświetlone dodatkowe ostrzeżenia. Ich

wystąpienie warto przemyśleć, gdyż w przyszłości mogą stać się albo problemem dla innych programistów pracujących z naszym kodem, albo wręcz spowodować błąd programu.

Ćwiczenie

- Pobierz plik `d1.c`, nie przeglądaj go tylko zapisz na dysku i spróbuj skompilować bez opcji, po czym uruchom program...
- Jak możesz zdiagnozować powstała w nim jakaś nieskończona pętla. Wciśnij `Ctrl+C` aby zakończyć wykonywany program.
- Skompiluj program przy pomocy podanych opcji. Otwórz program w drugim terminalu i poszukaj podejrzanych linii. Po każdorazowym znalezieniu błędu powtórz kompilację aż do uzyskania braku ostrzeżeń.

1.2 Asercje

Znalezienie bugów w dużym programie często jest bardzo trudne, dlatego już na początku warto zatroszczyć się o mechanizmy ułatwiające proces odnajdywania błędów.

Efektywnym sposobem unikania błędów oprogramowania jest jego testowanie. Najprostszym przykładem testowania jest pisanie odpowiednich konstrukcji warunkowych, które sprawiają, że program reaguje w odpowiedni sposób na sytuacje wyjątkowe. Innym sposobem (na etapie tworzenia oprogramowania) są specjalne makra (tzw. asercje).

W przypadku występowania użycia w programie asercji, jeśli w czasie działania programu makro stwierdzi np. nieprawidłową wartość zmiennej – wyrzuci błąd, a ponieważ sami makro pisaliśmy, możemy łatwiej stwierdzić skąd błąd się mógł wziąć. Skompiluj program z listingu 1, a następnie uruchom go i zaobserwuj działanie asercji.

```
#include <assert.h>
int main() {
  int i = 0;
  assert(i!=5);
  i++;
  assert(i==0);
  return 0;
}
```

Listing 1: Przykładowy program z asercjami

Takie przerwanie programu i wyrzucenie błędu nie stanowi oczywiście problemu w trakcie pracy nad programem w wersjach testowych. Co natomiast począć, jeśli nie chcemy by wyrzuciło nam przypadkiem błąd w trakcie działania programu w wersji RTM? Wystarczy przed dołączeniem pliku nagłówkowego *assert.h* zdefiniować makro NDEBUG: `#define NDEBUG`.

Ćwiczenie

- W pliku `d1.c` zdefiniowana zostało makro NDEBUG. Zakomentuj je, a następnie skompiluj program i uruchom go.
- Zdiagnozuj problem na podstawie zatrzymanego przez asercję programu.
- Obejrzyj kod programu, a następnie zastanów się, jakie błędy elementy tego kodu mogą powodować i spróbuj je poprawić.

1.3 GNU Debugger

Gdy kod się kompiluje bez ostrzeżeń i błędów, uruchamia i działa nie powodując wycieków pamięci – nie oznacza to, że wszystko jest w porządku. Istnieje bowiem możliwość, że nie działa prawidłowo.

Debugger to narzędzie służące do dynamicznej analizy innych programów, które ułatwia odnalezienie i identyfikację zawartych w nich błędów. Debugger umożliwia zlokalizowanie instrukcji odpowiedzialnych za wadliwe działanie programu poprzez śledzenie wartości poszczególnych zmiennych, wykonywanie instrukcji krok po kroku, a także wstrzymywanie działania programu w zdefiniowanych miejscach.

GNU Debugger (gdb) to narzędzie będące częścią projektu GNU. Działa on w trybie tekstowym, ale istnieją zintegrowane środowiska programistyczne potrafiące prezentować wyniki działania GNU Debuggera.

1.3.1 Ćwiczenie[1]

- Pobierz kod programu w pliku `d2.c`, a następnie skompiluj program i uruchom go.
- Program generuje wyjątek w czasie działania. Aby odnaleźć, w którym miejscu się to dzieje użyjemy debuggera. Najpierw skompiluj program z opcją `-g`, która umożliwi potem jego debugowanie.

[1]Na podstawie: `http://www.tutorialspoint.com/gnu_debugger/gdb_debugging_example1.htm`

- Czas na debugowanie:
  - Załaduj program do debuggera: `gdb ./a.out`.
  - Uruchom go: `run`.
  - Program powinien zatrzymać się z wyrzuconym wyjątkiem. Wpisanie komendy `where` pokaże, w którym miejscu został wyrzucony wyjątek.
  - Komenda `list` pokaże najbliższe linie kodu w okolicy tego miejsca, gdzie został wyrzucony wyjątek.
  - Z kolei komenda `print` umożliwia podejrzenie wartości zmiennych, np. `print x`.

  Wymienione komendy działają poprawnie także po wpisaniu skróconej nazwy (dopóki jest ona jednoznaczna dla debuggera), np. w postaci jednoliterowej `p x` zamiast `print x`.

1.4 Analiza kodu

Ściągnij ze strony pliki `cw1.c` oraz `cw2.c`. Przeanalizuj kod, stwórz plik wejściowy, a następnie uruchom program. Spróbuj przy pomocy debuggera znaleźć miejsce powstania błędu w programie i poprawić je.

2 Oprogramowanie do debugowania i testowania aplikacji

Istnieje wiele narzędzi do debugowania i testowania aplikacji, niektóre z nich (dla języka C) wymienione zostały na poniższej liście:

- GNU DataDisplayDebuger[2] – graficzne środowisko do debugowania przzu użyciu gdb, czy dbx. Oprócz podświetlania składni, oferuje również graficzne wyświetlanie np. struktur danych w postaci grafów.
- Valgrind[3] – pomaga w diagnozwaniu błędów w zarządzaniu pamięcią (głównie do wykrywania wycieków pamięci).
- CUnit[4] – framework do testów jednostkowych dla języka C.
- Testwell CTC++f̃ootnotehttp://www.verifysoft.com/pl_ctcpp.html – narzędzie sprawdzające pokrycie kodu oprogramowania.
- Check[5] i CuTest[6] – kolejne frameworki do testów jednostkowych.
- Cmockery[7] – niewielka biblioteka do tworzenia testów jednostkowych.
- CMock[8] – narzędzie to tworzenia modułów i obiektów imitujących zachowanie pewnych funkcji, użyteczne do interakcyjnych testów jednostkowych.

[2] http://www.gnu.org/software/ddd/
[3] http://www.valgrind.org
[4] http://cunit.sourceforge.net/index.html
[5] http://check.sourceforge.net/
[6] http://cutest.sourceforge.net/
[7] http://code.google.com/p/cmockery/
[8] http://sourceforge.net/apps/trac/cmock/wiki