Programování

Debugging a testování

There are
10 types
of people
in the world:

Those who
understand binary
and those
who don't.

Martin Urza

Co je debugging?

- V počítačích nulté generace byly důvodem některých chyb zkraty, které způsoboval mimo jiné hmyz, jenž do těchto strojů zalézal.
- Odstraňování chyb tedy probíhalo někdy i tak, že se technici vydali na výlet do hlubin počítače, kde lovili brouky.
- Brouk se řekne anglicky bug. Debugging pak vlastně znamená odstraňování brouků z počítače (původně z počítače, který zabíral celou budovu).
- Za čas se přestalo tímto termínem označovat hledání chyb v počítačích; začal se používat pro hledání chyb v programech.

Co je testování?

- Testování slouží k odhalování chyb v aplikacích. Jde o velmi důležitou fázi při vývoje software, bez které nelze zajistit kvalitu výsledného produktu.
- Průběh testování je takový, že se tester snaží dostat aplikaci do co nejvíce stavů (běžných i extrémních) a objevit chyby.
- Rozhodně se nejedná o nějakou podřadnou činnost; jde o zodpovědnou, důležitou a náročnou práci, jež nemůže být zastána každým.
  - Testeři často bývají programátoři, případně analytici, kteří aplikaci navrhli (a nejlépe ví, jak má fungovat).
  - Testeři by ideálně neměli testovat vlastní práci.

Rozdíl mezi debuggingem a testováním

- Debuggování i testování jsou činnosti, které vedou k odhalování a opravování chyb v programu; přesto se jedná o různé činnosti.
- Při testování používají testeři program podobně, jak by ho používali uživatelé (v tom smyslu, že nemusí mít k dispozici nic víc než hotový program).
- Cílem debuggingu je typicky odstranění nalezených chyb. Tester tedy objeví v programu chybu a úkolem programátora je chybu lokalizovat (v kódu) a poté ji odstranit. Debugging je proces lokalizace chyby v kódu. K debuggování je třeba více zdrojů a nástrojů než k testování (např. zdrojový kód, ale nejen to).

Debugging - základní prostředky

- Základním kamenem debuggování je **breakpoint**, což je nástroj, který dává programátorovi vývojové prostředí. Tento nástroj slouží k označení nějaké řádky kódu; když k ní pak program při svém běhu dorazí, je na tom místě zastaven.
- Programátor může pozorovat stav takto zastaveného programu (stav jsou zejména hodnoty proměnných).
- Pozastavený program lze také **krokovat**; to znamená pouštět jej krok za krokem dál. Po každém takovém kroku se program znovu zastaví a chová se naprosto stejně jako po zastavení na breakpointu.
  - Jeden krok typicky odpovídá jedné řádce kódu.

Debugging - další prostředky

- Zde se nejedná přímo o nástroj, který by poskytovalo vývojové prostředí, ale o „figle“, kterých může využít programátor k dosažení podobných cílů.
- Jednou z možností jsou **ladící výpisy**, jenž spočívají v tom, že programátor nechá na nějakém místě kódu program něco vypsat, např. do souboru či na výstup.
  - To lze využít mimo jiné v případě, kdy program někde spadne, případně se zacyklí, ale programátor neví kde.
- Ladící výpisy lze nahradit breakpointy, což se může hodit v prostředí, které breakpointy nepodporuje.
- Další možnost jsou **výjimky**, kterými se ale budeme podrobněji zabývat až za nějaký čas.

Breakpointy v Pascalu

- Breakpoint do kódu umístíte volbou Breakpoint v menu Debug (zkratka ctrl+F8). Breakpoint je v tom případě umístěn na řádku kódu, kde je kurzor.
  - Odstranění breakpointu probíhá stejně jako přidávání, záleží jen na tom, jestli je na pozici kurzoru v tu chvíli zrovna breakpoint (pak je odstraněn), nebo není (pak je přidán).
- Položkou Breakpoint List v menu Debug lze zobrazit seznam všech breakpointů v kódu.
  - To se hodí v případě, že je program moc dlouhý a nelze jej snadno prohlédnout.
- Podobně to funguje i v jiných prostředích a jazycích.

Krokování

- Zastaví-li se program na breakpointu, je dále možné jej krokovat řádku po řádce. Krok lze vyvolat v menu Run položkou Step Over (F8).
- Je-li na řádce funkce či procedura, Step Over (jak již název napovídá) ji „přeskočí“ (a samozřejmě vykoná), aniž vejde do jejího kódu. To se může hodit, avšak je možné, že hledaná chyba bude právě v přeskočeném kódu.
- Chceme-li skočit do funkce či procedury na aktuální řádce a nepřeskočit ji, existuje možnost Trace Into v menu Run (zkratka F7).
- Během krokování lze používat alt+F5.

Watches

- V menu Debug je položka Watches, která otvírá okno se sledovanými proměnnými.
- Toto okno je defaultně prázdné, je třeba do něj dát proměnné, které chceme sledovat.
  - To lze udělat klikem pravým tlačítkem myši na okno watches, což vyvolá nabídku s možností „new watch“. Téhož výsledku lze dosáhnout stiskem ctrl+F7.
  - Napsáním jména proměnné do „new watch“ lze tuto proměnnou přidat do watches.
- Watches ukazují hodnoty všech sledovaných výrazů, které jsou aktualizovány v každém kroku krokování.
  - Watches se nejlépe využívají s krokováním.

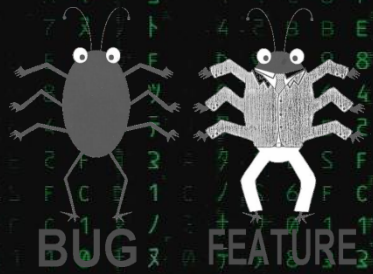
Cvičení krokování

- Zkuste si krokování ve svém překladači na kódu, který je na tomto slajdu.
- Dejte breakpoint na první řádku kódu, spusťte program a krokujte jej až do jeho konce.
  - Povšimněte si, že krokování skutečně odpovídá chování programu, a např. v podmínce je navštívena jen jedna větev.
- Poté, co kód odkrokujete naprázdno, přidejte proměnné bar a baz do watches a odkrokujte celý program znovu. Sledujte, jak se hodnoty proměnných mění a v každém kroku se aktualizují.
  - Nově aktualizované proměnné jsou označeny.

```
program foo;

var
  bar : integer;
  baz : integer;

begin
  bar := 2;
  baz := 2 * bar;
  bar := 2 * baz;
  writeln(bar);

  if (2 > bar)
    then writeln('++')
    else writeln('--');

  while (2 < bar)
    do bar := bar div 2;

  writeln(bar);
end.
```

Další debuggovací nástroje Pascalu

- V menu Debug lze najít mnoho zajímavých nástrojů, které zatím nebudeme podrobně rozebírat, avšak je dobré o nich vědět.
- Disassemble ukazuje kód programu v assembleru, což je nízkoúrovňový jazyk, jehož příkazy jsou přímo instrukce procesoru.
- Registers ukazuje tabulku, která zachycuje obsah registrů procesoru.
- Floating point unit umožňuje zobrazit stav registrů FPU (jednotka pro práci s desetinnými čísly).
- Vector unit ukazuje stav SSE registrů procesoru, což je jednotka pro práci s vektory čísel.



Testování aplikace

- Testování je velmi komplexní proces; je to vlastně asi podobor programování.
- Existuje mnoho postupů a způsobů testování. Jejich rozbor je výrazně nad rámec této přednášky, proto je zjednodušíme jen na návod k testování jednoduché aplikace.
- Nejjednodušší způsob testování spočívá asi v tom, že děláte s programem přibližně to, co by dělal uživatel a sledujete, zda program dělá to, co dělat má.
- **Testování je doopravdy důležité. Měla by mu být věnována patřičná pozornost a čas.**
  - Programy téměř nikdy nejsou napsány bez chyb.

Chyby v testování

- Asi nejobvyklejší chybou při testování je, když tester zkouší pouze pozitivní vlastnosti programu, tedy zda dělá to, co dělat má, ale už nezkouší, zda nedělá nic, co dělat nemá.
  - Příklad takového špatného testování velmi jednoduché aplikace, která má zjišťovat, jestli je číslo na vstupu prvočíslo, by bylo testovat pouze prvočísla a nepodívat se, jestli program správně funguje i pro ostatní čísla.
  - Tento příklad je dost jednoduchý, nicméně složitější případy na stejném principu se dějí často.
- Další častou chybou je, když tester ignoruje extrémy a testuje pouze „běžné situace“, nikoliv nezvyklé.
  - Uživatelé dělají v reálu hrozné věci ;o))

Cvičení

- Otestujte alespoň čtyři programy, které jste napsali v rámci minulých přednášek.
- Písemně zaznamenejte, jak jste testování prováděli, na jakých datech jste testovali a proč právě na nich.
- Pokud objevíte nějaké chyby, odstraňte je. K tomu využijte znalostí o debuggingu, které jste načerpali na začátku přednášky.
- I v případě, že žádné chyby neodhalíte, ověřte limity programu. Máte-li například zadávat programu čísla, zjistěte, zda mohou být libovolně velká, nebo jestli pro ně platí nějaké omezení. Má-li program přijímat řetězce, vymyslete a testujte omezující kritéria.

Cvičení

- Program uvedený vpravo by měl dostat na vstupu řetězec a na výstupu jej vrátit, avšak bez mezer.
- V tomto programu je několik chyb. Na to, jak je program krátký, je jich tam hodně.
- Najděte v programu všechny chyby a opravte je, aby fungoval. Ačkoliv by bylo snazší napsat vše znovu, nedělejte to, nýbrž upravte existující verzi (a přitom zachovejte použitý algoritmus).
  - Zjistíte, že najít chyby je těžší než je opravit.

```
program trim;

var
  input : string;
  output: string;
  loop1 : integer;

begin
  readln(input);
  output := '';

  for loop1 := 1 to length(output)
    do
    if (' ' = input[loop1])
      then output := input[loop1] + output;

  writeln(input);
  readln();
end.
```

Rekapitulace

- Měli byste vědět, že **testování je důležité**, už proto, že napsat program na první pokus bez chyby, je de facto nemožné, nejedná-li se o aplikaci naprosto triviální.
- Dále byste měli mít představu, jak správně testovat a čemu se naopak vyvarovat. Toto by mělo platit pro jednodušší a menší aplikace (u větších je to výrazně složitější).
- Též byste měli být schopni lokalizovat chyby pomocí ladících nástrojů Pascalu (breakpointy, krokování, watches, ...).
- Nezapomeňte na možnost používání ladících výpisů.