1 Testování studentských programů systémem Progtest

Do systému progtest studenti odevzdávají zdrojový kód svého řešení. Zdrojový kód obecně může být zapsán v různých programovacích jazycích, které testovací subsystém zná (aktuálně pouze C/C++), autor úlohy má případně dále možnost množinu programovacích jazyků omezit (aktuálně to ale nemá smysl).

Testování odevzdaného programu může mít dva scénáře:

1. testování samostatného programu,
2. testování modulu, který se zapojí do testovacího prostředí.

2 Testování samostatného programu

Tento způsob předpokládá, že odevzdaný studentský program je plnohodnotný a samostatný program, který s okolím komunikuje pouze pomocí svého standardního vstupu a výstupu. Jedná se o formu, která je snazší pro začínající programátory, protože testovaný program lze snadno zkoušet i na domácím počítači bez nutnosti se starat o programové rozhraní vytvářeného řešení.

Testování probíhá podle obrázku 1. Autor úlohy připravuje:

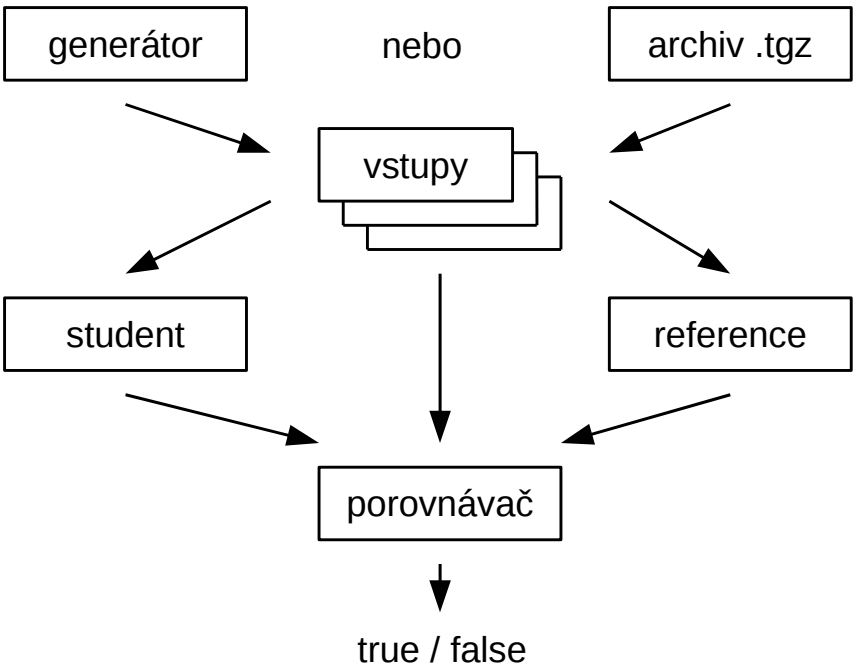
1. referenční řešení, tedy program vzorově implementující řešení zadaného problému,
2. sadu testovacích dat,
3. program, který porovnává referenční a studentský výstup (není nutný pro jednoduché úlohy, kde lze využít nějaký z předem připravených porovnávačů),
4. text zadání.

2.1 Referenční řešení

Implementované referenční řešení vytváří ze zadaných testovacích vstupů požadované výstupy. Správnost referenčního řešení je nezbytným předpokladem správné funkce celého automatizovaného testování úloh. Pokud je referenční řešení nesprávné (nebo i jen nefunkční pro některé vstupy), jsou tím postižení všichni studenti, kterým může být např. odmítáno jejich správně realizované řešení.

Při vytváření úloh je proto potřeba věnovat péči ověření ref. řešení. Samotné řešení autora není dostatečnou zárukou správnosti. Je proto vhodné, aby cvičící zkusili vyřešit zadanou úlohu a svým řešením správnost potvrdili.

Výzvy a odpovědi programu se mohou lišit pro jednotlivé jazykové mutace (česká varianta, anglická varianta, ...). Pokud je požadovaná funkce ve více



Obrázek 1: Testování samostatného programu

jazykových mutacích, musí referenční řešení použitý jazyk zohledňovat. Referenční řešení je v testovacím prostředí spouštěno s jedním parametrem – jazykem, kterým má program komunikovat. Parametrem je řetězec `CZE` pro český jazyk, `ENG` pro anglický jazyk, případně další zkratky pro jiné jazyky. Pokud je program spuštěn bez parametrů, měl by komunikovat ve výchozím jazyce (zřejmě `CZE`).

2.2 Testovací data

Studentský program je testován postupně několika testy. Testem se rozumí kontrola nějakého aspektu chování programu. Příkladem testu může být "test spávnosti pro velké množství vstupních dat" nebo "test reakce na nesprávná vstupní data".

Každý test se skládá z opakovaného spuštění studentského programu pro nějaká vstupní data. Typicky se pro jeden test připravuje 5 až 20 instancí vstupních dat. Vstupní data musí být připravena autorem problému. Vstupní data mohou mít různé podoby:

- archiv (.tgz), který obsahuje soubory (bez adresářů). Každý soubor představuje jednu instanci vstupních dat, která budou předaná testovanému programu, tedy studentský program bude spuštěn `počet souborů` krát.

- generovaná data, tedy program, který po svém spuštění vygeneruje v zadaném adresáři soubory obsahující vstupy pro testovaný program. Generátor může být realizován v různých programovacích jazycích (shell, Perl, C/C++, Python). Generátor je spuštěn testovacím prostředím, na příkazové řádce jsou generátoru předané potřebné parametry, kde mají být soubory se vstupními daty vytvořeny:

  ```
  generator <dst_path> <lang> <seed>
  ```

  `dst_path` je jméno adresáře, kde mají být uložené soubory s testovacími daty. Typicky generátor testovacích dat vytvoří v zadaném adresáři soubory `0000`, `0001`, ...

  `lang` je zkratka používaného jazyka. Obecně, vstup studentského programu se může lišit podle toho, zda student studuje v české, anglické nebo i jiné mutaci předmětu. Testovací prostředí jako parametr předá hodnotu tohoto jazyka. Pro české mutace bude předán parametr `CZE`, pro anglické parametr `ENG` a pro případné další jazyky odpovídající zkratky.

  `seed` je číselná hodnota udávající inicializaci náhodného generátoru, který je (případně) v generátoru použit. Pokud generátor náhodná čísla nepotřebuje, může tento parametr ignorovat. Pokud generátor náhodná čísla potřebuje, může ponechat generátor náhodných čísel neinicializovaný, ale pak budou vždy generovaná "stejná" náhodná data

při testování všech studentů. To může být na škodu, studenti např. mohou při znalosti náhodných dat švindlovat a místo korektního řešení do programu přidávat výjimky, kterými obejdou známá zadání vstupů. Lepší řešení je generátor náhodných čísel zadanou hodnotou inicializovat. Tím bude generátor generovat pro každé spuštění jiné hodnoty, ale při znalosti hodnoty `seed` lze generátor spustit později znovu a získat stejná testovací data. To se hodí např. při zjišťování příčin, proč byl program odmítnut.

návratový kód generátor by měl vrátit hodnotu 0 pro úspěch, nenulu pro selhání.

2.3 Porovnávač

Úkolem porovnávače je zkontrolovat (porovnat) výstup studentského a referenčního programu. Podle charakteru úlohy může být porovnání různě obtížné:

přesná shoda výstup studentského a referenčního programu musí být zcela stejný. Tedy musí souhlasit i bílé znaky, odřádkování apod. Tento porovnávač je v testovacím systému již hotový, autor úlohy jej pouze vybere pro použití.

přesná shoda, nezapočítává bílé znaky výstup referenční a studentský se může odlišovat v bílých znacích (tedy např. počet za sebou jdoucích mezer nemusí souhlasit). Tento porovnávač je opět v testovacím prostředí již implementován.

přesná shoda, nezapočítává malé rozdíly des. čísel výstup referenční a studentský se může odlišovat v hodnotách desetinných čísel. Porovnávač hledá ve výstupu desetinná čísla (tj. čísla , která mají des. tečku nebo exponent) a porovnává jejich hodnoty. Vyžaduje shodu v nečíselném výstupu a toleruje rozdíl v hodnotách des. čísel menší než jedno promile. Tento porovnávač je opět v testovacím prostředí již implementován.

přesná shoda, nezapočítává malé rozdíly des. čísel a bílé znaky výstup referenční a studentský se může odlišovat v hodnotách desetinných čísel a v sekvencích bílých znaků (mezer). Porovnávač hledá ve výstupu desetinná čísla (tj. čísla , která mají des. tečku nebo exponent) a porovnává jejich hodnoty. Vyžaduje shodu v nečíselném výstupu a toleruje rozdíl v hodnotách des. čísel menší než jedno promile. Tento porovnávač je opět v testovacím prostředí již implementován.

program v sh/Perl/C/C++... Pokud pro vyhodnocení správnosti nepostačuje žádný z porovnávačů uvedených výše, lze implementovat porovnávač vlastní. Příkladem může být zadání, kde je po studentech požadován program na faktorizaci zadaného čísla. Pokud je úloha zadaná bez dalších omezení, není pořadí prvočíselných činitelů na výstupu důležité. Porovnávač by tedy měl porovnat výstup referenční a studentský

bez ohledu na pořadí nalezených faktorů. V takovém případě lze implementovat vlastní program, který bude hrát roli porovnávače:

`comparator <input> <ref_out> <stud_out>`

`input` udává jméno souboru se vstupními daty (tak, jak byl připraven generátorem),

`ref_out` udává jméno souboru s referenčním výstupem,

`stud_out` udává jméno souboru se studentským výstupem,

návratový kód porovnávač vrací hodnotu 0 pro shodu (tj. studentský výstup vyhovuje), 1 pro neúspěch (studentský výstup nevyhovuje),

standardní výstup porovnávač může na svém std. výstupu předat další dodatečnou informaci o nalezeném rozdílu, tento dodatečný výstup může být zpřístupněn jako nápověda studentovi.

2.4 Text zadání

Kromě vlastního popisu zadaného problému je potřeba zdůraznit některá specifika při odevzdávání automatickému hodnotícímu systému:

- je vhodné zadání doplnit ukázkou běhu pro několik instancí vstupních dat,
- je vhodné ukázkový běh (data použitá pro demonstraci) zahrnout do testování jako první test,
- je vhodné k zadání připojit sadu testovacích dat (archiv), do archivu umístit vstupní data (tak, jak je vygeneruje generátor pro základní test) a výstupy referenčního programu,
- zejména pro první úlohy je potřeba zdůraznit, že se odevzdává zdrojový kód,
- je potřeba zdůraznit, že porovnávač je stroj, který trvá na shodném výstupu jako je výstup referenční. Tedy že záleží na každém znaku,
- je nutné zdůraznit roli odřádkování, typicky zdůraznit, že odřádkování je i za poslední řádkou výstupu programu,
- je důležité zdůraznit, že některé funkce se v odevzdávaném programu nesmí používat. Mj. je zakázáno použití funkce `system`. Pokud mají studenti doporučeno na konec programu přidávat volání:

  ```
  system("pause");
  ```

  je důležité v textu zdůraznit, že toto volání nemají používat při odevzdávání zdrojového programu. Lze využít preprocesoru a skutečnosti, že testovací prostředí definuje proměnnou `__PROGTEST__`:

  ```
  #ifndef __PROGTEST__
  system("pause"); // nyni ok
  #endif /* __PROGTEST__ */
  ```

2.5 Další parametry testování

Pro testování lze nastavit další omezující parametry, kterými lze systémem black-box testování usuzovat na použité algoritmy a pod. Například nastavením vhodného omezení na dobu běhu a vhodného rozsahu úlohy si lze vynutit řešení, které používá asymptoticky lepšího algoritmu (např. logaritmického místo lineárního).

Omezení lze nastavit na dobu běhu,velikost dostupné paměti a velikost dostupného zásobníku:

doba běhu určuje časové omezení na jedno spuštění studentského programu v daném testu. Doba je zadaná v sekundách. Pokud studentský program nestihne výpočet v zadaném čase, je násilně ukončen a daný test je ohodnocen jako nesplněný. Při zadávání úlohy je potřeba odhadnout tento časový limit. Vodítkem bývá doba běhu ref. řešení. Praktické zkušenosti ukazují, že nastavená max. doba běhu by měla být cca 10x delší než doba, kterou pro běh požaduje ref. řešení. Zároveň není vhodné nastavovat tento limit vysoko. Pokud by byl nastaven vysoko, mohl by nesprávný studentský program blokovat hodnotící systém na dlouhou dobu. Rozumná horní mez je cca 20 sec, pokud je potřeba více, je potřeba uvažovat o snížení rozsahu vstupních dat.

velikost paměti určuje maximální velikost dostupné paměti, kterou může proces dynamicky alokovat. Zadávaná hodnota je dost hrubá. Za běhu lze omezovat pouze velikost VM daného procesu, velikost VM pro daný proces je určena ze zadaného limitu a odhadnuté velikosti spouštěného programu a linkovaných knihoven. Pokud je zadaná velikost paměti nulová, nebude použito žádné omezení.

velikost zásobníku určuje max. dostupnou hloubku zásobníku.

kontrola mudflap určuje, zda bude použit paměťový debugger mudflap pro kontrolu alokací paměti. Pokud je volba aktivní, je studentský program překládán s dodatečným kódem, který za běhu kontroluje používané ukazatele a testuje, zda nedochází k zápisům mimo alokované proměnné. Navíc je na konci programu kontrolováno, zda byly uvolněné všechny dynamicky alokované bloky paměti.

Paměťový debugger je na testovaný program velmi přísný. Pokud je zjištěn přístup k nealokované paměti, je program ukončen (segmentation fault). Chyby jsou detekované i v situacích, kdy se program zdá být funkční a za běžného režimu nepadá. Typicky program funguje, ale pracuje s indexem pole o 1 vyšším než je alokovaný rozsah. Takový program bude většinu času pracovat správně, při kontrole paměťovým debuggerem bude pravidelně hodnocený jako nesprávný.

Při použití paměťového debuggeru je potřeba počítat s vyšší paměťovou náročností a delší dobou běhu (podle charakteru programu se oba parametry mohou zvětšit i 20x). Pokud je paměťový debugger použit, je rozumné pracovat s menším objemem vstupních dat.

2.6 Ukázka

V aresáři `priklad1` je ukázka zadání úlohy, která využívá tento typ testování. Jedná se o úlohu určení parametrů trojúhelníku na základě velikostí jeho stran.

`reference.cpp` ukázkové (referenční) řešení, implementované pro českou a anglickou jazykovou mutaci. Referenční řešení není studentům dostupné.

`zadani_CZE.txt` ukázka zadání úlohy v českém jazyce. Použité kódování je UTF-8. Součástí zadání je ukázka reakce programu na několik základních příkladů. Tento text je zobrazen studentům.

`test1.pl` generátor testovacích dat pro první test (základní test pro data podle ukázky). Testovací data jsou pevně daná. Ke skriptu studenti nemají přímo přístup, vidí ale testovací data v ukázce.

`test2.pl` generátor testovacích dat pro druhý test (test mezních hodnot). Testované hodnoty zkoušejí např. správnost trojúhelníkových nerovností (zkoušejí právě rovnost). Testovací data jsou připravena podle určitého vzoru, ale jsou parametrizovaná náhodnými hodnotami, aby pro různé studenty byly vstupy pokaždé trochu jiné. Ke skriptu studenti nemají přístup, konkrétní hodnoty si ale mohou nechat zobrazit v případě neúspěchu.

`test3.pl` generátor testovacích dat pro třetí test (kontrola ošetření vstupů).

`test4.pl` generátor testovacích dat pro čtvrtý test (kontrola náhodnými vstupy).

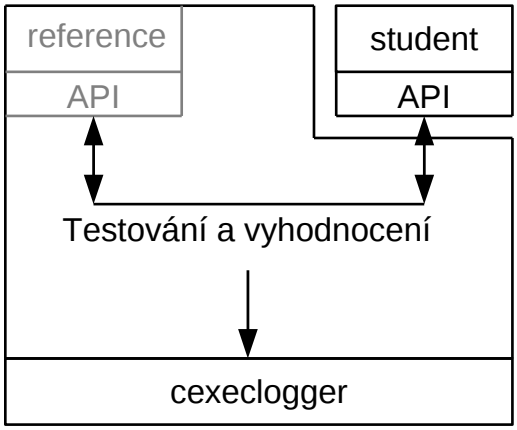
`CZE` adresář s páry vstupních data a jim odpovídajících očekávaných výstupních dat. Tento adresář (zabalený v .tgz archivu) je studentům dostupný pro jejich testování. Vstupní data odpovídají vstupům z testu č. 1, studenti si tedy mohou svůj program doma otestovat stejně jako to dělá automat v základním testu.

3 Testování modulu, který se zapojí do testovacího prostředí

Testování samostatných programů se hodí pro jednoduché úlohy, které jsou typické pro úvodní kurzy programování. Výhodou je relativní jednoduchost řešení, kdy student v základních úlohách pouze implementuje funkci `main`.

Pro pokračování, kdy si studenti osvojují základy modulárního programování a OOP, je tento jednoduchý postup nepoužitelný. Správně vyřešená úloha (např. z OOP při procvičování polymorfismu) vyžaduje řešení korektní nejen po stránce funkční, ale i po stránce navržení a realizace tříd. Pro testování takových úloh je možné použít tento typ testování. Princip je znázorněn na obrázku 2.

Při přípravě tohoto zadání je potřeba:



Obrázek 2: Testování modulu, který se zapojí do testovacího prostředí

- navrhnout rozhraní mezi testovacím prostředím a studentským modulem. Toto rozhraní má typicky podobu sady funkcí nebo několika tříd. Požadované rozhraní je možné vložit do zadání úlohy, případně je možné po studentech požadovat návrh takového rozhraní na základě textového popisu a ukázky použití,
- implementovat testovací prostředí, které využívá rozhraní výše a pomocí tohoto rozhraní kontroluje studentskou implementaci,
- implementovat referenční řešení,
- vytvořit text zadání.

3.1 Testovací prostředí

TBD