Zadání úlohy do projektu z předmětu IPP 2016/2017
(Obecné a společné pokyny všech úloh jsou v `proj2017.pdf`)

CLS: C++ Classes

Zodpovědný cvičící: Radim Krčmář (ikrcmar@fit.vutbr.cz)

1 Detailní zadání úlohy

Vytvořte skript pro analýzu dědičnosti mezi třídami popsanými zjednodušenou syntaxí pro soubory programovacího jazyka C++11. Tento skript vytvoří strom dědičnosti mezi zadanými třídami, případně vypíše detaily o všech členech dané třídy.

1.1 Formát vstupu

Vstupem skriptu bude soubor se zjednodušenou syntaxí souborů jazyka C++. Myšlenkou zjednodušení je zachovat pouze konstrukce umožňující vyjádřit třídy a dědičnost. Syntaxe je zjednodušena následovně:

- V souboru jsou možné pouze definice/deklarace tříd a jejich členů (včetně redefinice a přetěžování členů tříd). Definice metod uvnitř tříd je naznačena pouze složenými závorkami (`{` a `}`), mezi nimiž mohou být pouze bílé znaky. Neuvažujte definice vnořených tříd. Každá deklarovaná třída či metoda bude rovnou i definována (tj. žádné dopředné deklarace nebo definice metody mimo třídu).
- V souboru se nenachází žádné příkazy preprocesoru, tedy není možné definování maker, ani `include` knihoven.
- Neuvažujte ani definici/deklaraci jmenných prostorů, šablon, nebudou definovány ani spřátelené třídy (`friend`), metody neměnící svůj objekt (`const`) a ani klíčová slova `explicit`, `final`, `override`.
- Neuvažujte virtuální dědění tříd, e.g. `class B : virtual A`.
- Pro jednoduchost neuvažujte definici vlastních operátorů uvnitř tříd.
- Nemusíte uvažovat definici/deklaraci typů. V rámci projektu se budou ve vstupech jako datové typy vyskytovat jen základní datové typy (s výjimkou `std::nullptr_t`, který bude vynechán), definované třídy a jejich členové, a ukazatelé/reference na tyto datové typy.
- Ve vstupním souboru nemusíte počítat s komentáři.

Používaná podmnožina jazyka C++ vystačí s klíčovými slovy `class`, `public`, `protected`, `private`, `using`, `virtual` a `static`, dále přiřazením pro čistě virtuální metody (`= 0`), základními datovými typy, hvězdičkou, ampersandem, identifikátory, dvojtečkou, středníkem, čárkou, vlnkou, závorkami a bílým znakem.

Můžete uvažovat, že formát vstupu bude vždy odpovídat naší zjednodušené syntaxi. Mohou však nastat sémantické chyby: neznámé typy, dědění z nedefinované třídy, apod.

1.2 Formát výstupu

Výstupem skriptu bude XML popisující **a)** strom dědičnosti mezi třídami definovanými na vstupu; nebo **b)** popis všech (tedy i zděděných) členů zadané třídy.

V případě **a)** bude výstup XML ve formátu dle následujícího příkladu:

```
<?xml version="1.0" encoding="UTF-8"?>
<model>
    <class name="A" kind="abstract"></class>
    <class name="B" kind="concrete">
        <class name="C" kind="concrete"></class>
    </class>
    <class name="D" kind="concrete">
        <class name="C" kind="concrete"></class>
    </class>
</model>
```

Element `model` je kořenový a vždy přítomný. Každý element `class` zastupuje jednu třídu ve stromu dědičnosti, jméno této třídy je uvedeno v atributu `name` tohoto elementu. Atribut `kind` nabývá hodnoty *abstract* v případě, že je třída jména v atributu `name` abstraktní, nebo *concrete*, pokud abstraktní není. Nachází-li se nějaký `class` uvnitř jiného elementu `class`, značí to, že vnitřní element dědí od vnějšího. V rámci výstupu tedy popisujete vícenásobnou dědičnost lesem, takže dědí-li některá třída současně od více tříd, bude se nacházet v tomto výpisu vícekrát (v příkladu třída C dědí současně od B a D) včetně případných podelementů.

V případě **b)** bude formát výstupu vycházet z následujícího příkladu:

```
<?xml version="1.0" encoding="UTF-8"?>
<class name="B" kind="abstract">
    <inheritance>
        <from name="C" privacy="public" />
    </inheritance>
    <private>
        <attributes>
            <attribute name="question" type="char32_t *" scope="instance">
                <from name="C" />
            </attribute>
        </attributes>
    </private>
    <protected>
        <methods>
            <method name="the_answer" type="int" scope="static">
                <arguments>
                </arguments>
            </method>
        </methods>
    </protected>
    <public>
        <attributes>
            <attribute name="myatt" type="int *" scope="instance">
                <from name="C" />
```

```
            </attribute>
        </attributes>
        <methods>
            <method name="is_real" type="bool" scope="instance">
                <virtual pure="yes" />
                <arguments>
                    <argument name="cls" type="A &amp;amp;" />
                </arguments>
            </method>
        </methods>
    </public>
</class>
```

Následuje popis jednotlivých elementů a jejich atributů:

- `class` zastupuje vyhledávanou třídu (viz přepínač `details`), atribut `name` udává jméno dané třídy. Atribut `kind` má stejný význam a rozsah hodnot jako ve formátu pro strom dědičnosti.
- `inheritance` obaluje elementy `from`. Tyto slouží pro určení, z jaké třídy (`name`) se dědí (jen přímá dědičnost) a s jakou úrovní přístupu (`privacy`). Atribut `privacy` může nabývat hodnot *public*, *private* a `protected`.
- Elementy `public`, `private` a `protected` obalují definice metod a atributů třídy, které jsou zde podle své úrovně přístupu. Každý z elementů `public`, `private` a `protected` se smí v elementu `class` nacházet nejvýše **jednou**, a to právě tehdy, když takový element obsahuje alespoň jeden podelement.
- `attributes` obaluje elementy `attribute` a `methods` obaluje elementy `method`.
- `attribute` znázorňuje každou ze členských proměnných třídy. Atribut `name` udává jméno proměnné, `type` její datový typ a `scope`, zdali je daná proměnná statická (v tom případě `scope` nabývá hodnoty *static*) nebo vázaná k instancím této třídy (hodnota *instance*). Tehdy a jen tehdy, je-li proměnná zděděna z nějaké jiné třídy, obsahuje element `attribute` podelement `from` s atributem `name`, který udává jméno třídy, ve které je proměnná původně definována.
- `method` značí metodu dané třídy. Má stejné atributy se stejným významem jako u `attribute`, stejně tak může obsahovat element `from`, opět se stejným významem jako u `attribute`. Je-li metoda virtuální (polymorfní), bude obsažen také element `virtual` s atributem `pure` s hodnotou buď *yes*, nebo *no* dle toho, zda metoda je, nebo není čistě virtuální. Dále element `method` **vždy** obsahuje podelement `arguments`.
- `arguments` je element obalující elementy typu `argument`. Každý element `argument` značí jeden argument metody, obsahuje atributy `name` a `type` se stejným významem jako např. u elementu `attribute`. Neuvažujte metody s proměnným počtem argumentů ani s implicitními hodnotami argumentů.

V rámci odvozování členů třídy se může stát, že dojde ke konfliktu mezi členy z několika děděných tříd. To znamená, že není možné přesně určit, kterou z implementací daného členu má dědící třída zvolit (tj. v rámci stromu dědičnosti by ke členu určitého jména vedly alespoň dvě různé cesty).

V případě, že došlo ke konfliktu popsanému výše, ukončete skript s návratovou hodnotou 21.

Tento skript bude pracovat s těmito parametry:

- `--help` Viz společné zadání všech úloh.
- `--input=`*file* Vstupní textový soubor *file*, který obsahuje popis tříd jazyka C++ podle popsaných omezení. Předpokládejte kódování ASCII. Chybí-li tento parametr, je uvažován standardní vstup.
- `--output=`*file* Výstupní soubor *file* ve formátu XML v kódování UTF-8. Není-li tento parametr zadán, bude výstup vypsán na standardní výstup.
- `--pretty-xml=`k Výstupní XML bude formátováno tak, že každé nové zanoření bude odsazeno o k mezer oproti předchozímu. Není-li k zadáno, uvažujte $k = 4$. Pokud tento parametr není zadán, je formátování výstupního XML volné (doporučujeme mít na každém řádku maximálně jeden element).
- `--details=`*class* Místo stromu dědičností mezi třídami se na výstup vypisují údaje o členech třídy se jménem *class*. Formát je popsán výše. Pokud argument *class* není zadán, vypisují se detaily o všech třídách v daném souboru, kde kořenem XML souboru je `model`. Pokud *class* neexistuje, bude na výstup vypsána pouze XML hlavička.

Poznámky:

- Zděděné privátní atributy/metody sice nejsou přístupné, ale v C++ stále způsobují konflikty.
- Třída je abstraktní, dokud nejsou implementovány všechny, i kvůli dědění nepřístupné, čistě virtuální členské funkce.
- Budou testovány i nepřípustné typy, třeba `uint64_t number;`.
- `std::is_fundamental` může pomoci získat 19 základních typů naší podmnožiny C++.
- V následujícím příkladě je ve třídě `C` za původní třídu atributu `x` považována třída `A`.

```
class A { protected: int x; };
class B : protected A {};
class C : B { public: using B::x; };
```

- Testujte vstupy s kompilátorem C++ – váš program by měl dospět ke stejným výsledkům ohledně dědičnosti.

Reference:

- Classes (I) - C++ Tutorials. *Cplusplus.com* [online]. ©2000-2015 [cit. 2016-02-05]. Dostupné z: `http://www.cplusplus.com/doc/tutorial/classes/`
- ISO/IEC N3337. Working Draft, Standard for Programming Language C ++. 2012. Dostupné z: `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3337.pdf`

2 Bonusová rozšíření

- **CFL** (Konflikty, 2 body): Bude-li spolu s přepínačem `--details` zadán i přepínač `--conflicts` a nastane-li v rámci dědičnosti konflikt při odvozování členů nějaké třídy, zaznamenejte tento konflikt (formát výpisu níže) a pokračujte v odvozování pro danou třídu již bez tohoto konfliktního členu (tedy tento člen již nebude vypsán v `attributes` nebo `methods`).

  Pro výpis konfliktů v elementu `class` v původním formátu výpisu přibude element `conflicts` s následující syntaxí:

```
<conflicts>
    <member name="myattr">
        <class name="B">
            <[privacy]>
                <[memberkind]></[memberkind]>
            </[privacy]>
        </class>
        <class name="C">
            <[privacy]>
                <[memberkind]></[memberkind]>
            </[privacy]>
        </class>
    </member>
</conflicts>
```

  Element `member` zastupuje daný konfliktní člen, atribut `name` je jeho jméno. Každý `class` element uvnitř `member` zastupuje třídu, z které bylo děděno a která obsahuje člen daného jména, čímž se účastní na konfliktu. Obsah elementu `class` je přizpůsoben tomu, jaké má konfliktní člen uvnitř této třídy vlastnosti (tedy `[privacy]` bude jedno z `public`, `private`, `protected`, `[memberkind]` bude jedno z `attribute` nebo `method`, podle toho, jestli se v původní třídě jednalo o atributu nebo metodu, s vlastnostmi stejnými jako v původním formátu výpisu). Element `[privacy]` v tomto případě nebude obsahovat jiné členy než člen konfliktní.

  Pokud dojde k tomu, že konfliktní člen se nachází jen v jedné bázové třídě, bude element `conflicts` obsahovat pouze jeden záznam o tomto členu.

  V případě, že dědící třída, v níž nastal konflikt, figuruje jako rodičovská i v jiných vztazích, neuvažujte, že by obsahovala žádný z konfliktních členů.

  Při konfliktu s polymorfní metodou (mající více implementací pro různé typy argumentů) vypište všechny implementace.

  Příklad: Mějme rozšíření známého diamantu:

```
class A { public: int x; };
class B: public A {};
class C: public A {};
class D: public B, public C {};
class E: public D {};
```

Třída `E` neobsahuje konflikt, jelikož konfliktní člen z `D` byl odebrán a konflikt pro třídu `D` bude vypadat následovně:

```
<conflicts>
     <member name="x">
         <class name="A">
             <public>
                <attribute name="x" type="int" scope="instance" />
             </public>
        </class>
     </member>
</conflicts>
```

3 Poznámky k hodnocení:

Výsledný XML soubor bude porovnáván s referenčními XML soubory nástrojem JExamXML na porovnání XML souborů, který se umí správně vypořádat například s různým odsazením elementů. Více viz stránka IPP:ProjectNotes na Wiki předmětu.

V úloze je zakázáno použít generátory syntaktických analyzátorů.