Treća laboratorijska vježba iz OOUP

1. Tvornice (30% bodova)

Ova vježba razmatra oblikovanje tvornica koje ne ovise o konkretnim tipovima. Takve tvornice ćemo nazivati generičkim tvornicama, iako njihova izvedba ne mora biti povezana s generičkim programiranjem. Glavna prednost generičkih tvornica jest u tome što ih nije potrebno mijenjati kad želimo stvarati nove vrste komponenata. U svim vježbama ćemo razmatrati primjer s kućnim ljubimcima kojeg smo razmatrali i u prvoj laboratorijskoj vježbi. Htjet ćemo omogućiti stvaranje novih vrsta ljubimaca bez potrebe za mijenjanjem kôda koji inicira stvaranje. U C-u, ključni poziv bi bio:

```
struct Animal* p1=myfactory("cat", "Ofelija");
```

Zadatak generičke tvornice je asocirati simbolički identifikator `cat` s konkretnim podatkovnim tipom `struct Cat`. Na žalost, izvedbe generičkih tvornica čvrsto su vezane uz izvedbene detalje programskih jezika pa ćemo stoga pojedine jezike morati razmatrati ponaosob.

Obavezni dio vježbe uključuje izvedbu u C-u (1.1), te *barem jednu* od izvedbi u Pythonu (1.2), C++-u (1.3) i Javi (1.4). Naravno, uvijek su dobrodošli i studenti koji riješe više od obaveznog dijela.

1.1. Prvo ćemo pogledati kako bismo generičku tvornicu izveli u C-u. S obzirom da C ima vrlo ograničene mogućnosti introspekcije, jedini donekle portabilni način da taj cilj postignemo jest zapakirati konkretne objekte u dinamičke biblioteke (.dll,.so). Međutim, prije nego što uronimo u detalje, definirajmo naš cilj sljedećim ispitnim izvornim kôdom.

```
#include "myfactory.h"

#include <stdio.h>
#include <stdlib.h>

typedef char const* (*PTRFUN)(...);

struct Animal{
  PTRFUN* vptr;
  // vtable entries:
  // 0: char const* name(void* this);
  // 1: char const* greet();
  // 2: char const* menu();
};

// parrots and tigers defined in respective dynamic libraries

// animalPrintGreeting and animalPrintMenu similar as in lab 1

int main(int argc, char *argv[]){
  for (int i=0; i<argc/2; ++i){
    struct Animal* p=(struct Animal*)myfactory(argv[1+2*i], argv[1+2*i+1]);
    if (!p){
      printf("Creation of plug-in object %s failed.\n", argv[1+2*i]);
      continue;
    }

    animalPrintGreeting(p);
    animalPrintMenu(p);
    free(p);
  }
}
```

Zadatci vježbe su sljedeći:

1. Implementirati funkciju

   ```
   void* myfactory(char const* libname, char const* ctorarg);
   ```

   Funkcija treba i) otvoriti biblioteku zadanu prvim argumentom (`libname`), ii) učitati iz nje funkciju `create`, iii) pozvati create sa svojim drugim argumentom (`ctorarg`), te iv) dobiveni pokazivač vratiti pozivatelju. Zbog jednostavnosti, funkcija treba pretpostaviti da se tražena biblioteka nalazi u tekućem kazalu te dekorirati ime biblioteke tekućim kazalom './' i standardnom ekstenzijom `.so` (UNIX) ili `.dll` (Windows). Prije pisanja kôda preporučamo proučiti funkcije `dlopen` i `dlsym` (UNIX), odnosno `LoadLibrary` i `GetProcAddress` (Windows). Također, provjerite da li u potpunosti razumijete **značenje** deklaracije tipa `PTRFUN`. Neka prototip funkcije bude u `myfactory.h`, a izvedba u `myfactory.c`.
2. Implementirati funkcije `animalPrintGreeting` i `animalPrintMenu`. Implementacija će biti vrlo slična kao i u vježbi 1. Međutim, pripazite da sada do imena ljubimca dolazimo pozivom metode `name` (indeks 0 virtualne tablice). Sada bi se glavni program trebao moći prevesti (`gcc main.c myfactory.c -ldl`). Pokretanje dobivene izvršne datoteke treba rezultirati porukom `Creation of plug-in objects failed.`.
3. Implementirati dinamičke biblioteke `parrot.so` i `tiger.so` (odnosno `parrot.dll` i `tiger.dll` na Windowsima). Izvedba će ponovo biti vrlo slična izvedbama odgovarajućih funkcija u prvoj laboratorijskoj vježbi. Izvorni kod treba i) definirati konkretni tip ljubimca strukturom koja će osim pokazivača na virtualnu tablicu imati i pokazivač na ime. ii) implementirati funkcije ("metode"): `name`, `greet` i `menu`. iii) definirati virtualnu tablicu, te iv) definirati funkciju za stvaranje novih objekata na gomili s prototipom `void* create(char const* name);` Izvorni kôd dinamičkih biblioteka smjestite u datoteke `tiger.c` i `parrot.c` te ih prevedite (za `parrot.c`: `gcc -shared -fPIC parrot.c -o parrot.so`)
4. Ponovo testirati glavni program i otkloniti eventualne nedostatke. Sljedeća preporuka bi vam mogle pomoći da ne lutate previše. Neka se sve datoteke izvornog koda (`main.c`, `myfactory.c`, `myfactory.h`, `parrot.c`, `tiger.c`) nalaze u istom kazalu. Prevođenje i testiranje tada možete provesti lijepljenjem sljedećih naredbi u terminal (naravno, možete napraviti i skriptu).

```
gcc main.c myfactory.c -ldl
gcc -shared -fPIC tiger.c -o tiger.so
gcc -shared -fPIC parrot.c -o parrot.so
./a.out tiger mirko parrot modrobradi
```

5. Predložiti rješenje koje bi klijentima biblioteke omogućilo fleksibilnost pri alociranju memorijskog prostora za novi objekt. Vaše rješenje mora omogućiti stvaranje objekata na stogu odnosno unutar odvojeno alociranog memorijskog prostora, kao što je bilo traženo i u prvom zadatku prve vježbe. Uputa 1: generičku tvornicu potrebno je prekrojiti. Uputa 2: biblioteke trebaju definirati dvije dodatne funkcije; prva funkcija (možete je zvati sizeof) vraća veličinu objekata u bajtovima; druga funkcija (možete je zvati construct) inicijalizira objekt u memorijskom prostoru kojeg zadaje pozivatelj. Uputa 3: memoriju na stogu možete zauzeti lokalnim poljem znakova ili pozivom funkcije alloca (ili _malloca); na Windowsima funkcionira samo ova druga opcija.
6. Omogućite zadavanje identične funkcije `main` ali na način da izvedba strukture `Animal` ne bude vidljiva kodu iz datoteke `main.c`. Upute:
   - definicije tipa `PTRFUN`, strukture `Animal` te funkcija `animalPrintGreeting` i `animalPrintMenu` premjestite u datoteku `animal.c`.
   - u glavni program dodajte redak `#include "animal.h"`
   - predložite sadržaj zaglavlja `animal.h` tako da program bude korektan
   - razmislite o vezi s neprozirnim pokazivačima sa stranica "NIO, C" **predavanja** o načelima oblikovanja
7. Prevedite novi program ovim naredbama:

```
gcc -c myfactory.c animal.c
g++ main2.c myfactory.o animal.o
```

   Sada bi se linker trebao pobuniti da ne može razriješiti funkcije koje je preveo gcc. Izmijenite datoteke `animal.h` i `myfactory.h` tako da riješite problem. Vaše izmjene trebaju biti takve da sve komponente možemo prevesti i gcc-om, kao i ranije. Pomoć: ključni elementi rješenja su makro `__cplusplus` i deklaracija `extern "C"`.

1.2. Sada ćemo razmotriti izvedbu generičkih tvornica u Pythonu. Kako vježba ne bi bila prelaka, razmotrit ćemo malo teži problem. Potrebno je napisati program koji će napraviti sljedeće: i) iz svake datoteke izvornog koda u kazalu plugins instancirati po jednog ljubimca, te ii) sve instancirane ljubimce predstaviti načinom glasanja i omiljenim obrokom. Ispitni program bi izgledao ovako:

```
def test():
  pets=[]
  # obiđi svaku datoteku kazala plugins
  for mymodule in os.listdir('plugins'):
    moduleName, moduleExt = os.path.splitext(mymodule)
    # ako se radi o datoteci s Pythonskim kodom ...
    if moduleExt=='.py':
      # instanciraj ljubimca ...
      ljubimac=myfactory(moduleName)('Ljubimac '+str(len(pets)))
      # ... i dodaj ga u listu ljubimaca
      pets.append(ljubimac)

  # ispiši ljubimce
  for pet in pets:
    printGreeting(pet)
    printMenu(pet)
```

Ispis ispitnog programa treba izgledati ovako:

```
Ljubimac 0 pozdravlja: Sto mu gromova!
Ljubimac 0 voli brazilske orahe.
Ljubimac 1 pozdravlja: Mijau!
Ljubimac 1 voli mlako mlijeko.
```

Upute:

1. Ljubimce predstavi razredima `tiger` i `parrot` koje ćeš smjestiti u istoimene module kazala plugins. Kao i do sada, ljubimci trebaju definirati konstruktor koji u argumentu prima ime ljubimca, te metode `name`, `greet` i `menu`. Pojedine ljubimce možeš opisati s 9 redaka Pythona.
2. Funkcija `myfactory` treba koristiti funkciju `import_module` modula `importlib`, te ugrađenu funkciju `getattr`. Za definiciju funkcije dovoljna su 3 retka uključujući i zaglavlje.

1.3. U C++-u možemo dinamički stvarati komponente iz dinamičkih biblioteka, ali tim se ovdje nećemo baviti jer smo to već obradili u odjeljku koji se odnosi na programski jezik C. Ovdje ćemo međutim pokazati jednu drugu mogućnost, a ta je da tvornica stvara objekte konkretnih razreda koji su prevedeni i ugrađeni u izvršnu datoteku, ali bez da bude ovisna o njima. Ključ za ostvarivanje te funkcionalnosti je mogućnost da lokalne varijable u dosegu datoteke inicijaliziramo povratnom vrijednošću funkcije za koju će se prevoditelj pobrinuti da bude pozvana na samom početku izvđenja programa, prije ulaska u funkciju `main()`. Naš zadatak će biti analogan zadatku u Pythonu: proiterirati svim ukompajliranim ljubimcima te svakog od njih predstaviti načinom glasanja i omiljenim obrokom. U izvedbi valja koristiti sljedeće pretpostavke.

1. Svi ljubimci izvode razred Animal:

```
class Animal{
public:
  virtual char const* name()=0;
  virtual char const* greet()=0;
  virtual char const* menu()=0;
};
```

2. Svaki konkretni tip treba izvesti u zasebnoj komponenti. Konkretni tipovi trebaju definirati: