Druga laboratorijska vježba iz Oblikovnih obrazaca u programiranju:
načela oblikovanja, strategija, promatrač

0. Proceduralni stil i načela oblikovanja (5% bodova, bilo koji jezik)

Prevedite i isprobajte priloženi dopunjeni program s predavanja (str.~``Logička načela: NBP i proceduralni stil?").

```
#include <iostream>
#include <assert.h>
#include <stdlib.h>

struct Point{
  int x; int y;
};
struct Shape{
  enum EType {circle, square};
  EType type_;
};
struct Circle{
   Shape::EType type_;
   double radius_;
   Point center_;
};
struct Square{
   Shape::EType type_;
   double side_;
   Point center_;
};
void drawSquare(struct Square*){
  std::cerr <<"in drawSquare\n";
}
void drawCircle(struct Circle*){
  std::cerr <<"in drawCircle\n";
}
void drawShapes(Shape** shapes, int n){
  for (int i=0; i<n; ++i){
    struct Shape* s = shapes[i];
    switch (s->type_){
    case Shape::square:
      drawSquare((struct Square*)s);
      break;
    case Shape::circle:
      drawCircle((struct Circle*)s);
      break;
    default:
      assert(0);
      exit(0);
    }
  }
}
int main(){
  Shape* shapes[4];
  shapes[0]=(Shape*)new Circle;
  shapes[0]->type_=Shape::circle;
  shapes[1]=(Shape*)new Square;
  shapes[1]->type_=Shape::square;
```

```
    shapes[2]=(Shape*)new Square;
    shapes[2]->type_=Shape::square;
    shapes[3]=(Shape*)new Circle;
    shapes[3]->type_=Shape::circle;

    drawShapes(shapes, 4);
  }
```

Dodajte metodu moveShapes koja pomiče oblike zadane prvim argumentom za translacijski pomak određen ostalim argumentima. Ispitajte dodanu funkcionalnost.

Dodajte razred Rhomb. Dodajte jedan objekt tipa Rhomb u listu objekata u main(). Sjetite se, sad moramo promijeniti i drawShapes().

Ovo je domino-efekt (krutost), kojeg ćemo kasnije pokušati zauzdati. Za probu, zaboravite adekvatno promijeniti moveShapes(). Isprobajte ponovo. Sad bi moveShapes trebao "puknuti". To je krhkost uzrokovana redundancijom. Ni to ne želimo imati u programu.

1. Nadogradnja bez promjene u C-u (20% bodova, C)

Napišite u C-u funkciju `mymax` koja pronalazi najveći element zadanog polja. Vaša implementacija treba biti primjenljiva na polja elemenata svih mogućih tipova: cijelih brojeva, pokazivača ili struktura te omogućiti rad s različitim vrstama usporedbi. Kako biste ostvarili nadogradivost bez promjene, funkcija `mymax` treba primiti pokazivač na kriterijsku funkciju koja vraća 1 ako je njen prvi argument veći od drugoga, a 0 inače. Funkciju oblikujte prema primjeru funkcije `qsort` standardne biblioteke:

```
const void* mymax(
  const void *base, size_t nmemb, size_t size,
  int (*compar)(const void *, const void *));
```

Definirajte kriterijske funkcije za usporedbu cijelih brojeva, znakova i znakovnih nizova. Nazovite te kriterijske funkcije `gt_int`, `gt_char` i `gt_str`. U izvedbi funkcije `gt_str`, posao delegirajte funkciji `strcmp`.

Pokažite da vaša funkcija može pronaći najveće elemene sljedećih nizova:

```
int arr_int[] = { 1, 3, 5, 7, 4, 6, 9, 2, 0 };
char arr_char[]="Suncana strana ulice";
const char* arr_str[] = {
   "Gle", "malu", "vocku", "poslije", "kise",
   "Puna", "je", "kapi", "pa", "ih", "njise"
};
```

2. Nadogradnja bez promjene primjenom predložaka (10% bodova, C++)

U prethodnom zadatku smo vidjeli da se nadogradnja bez promjene u C-u može postići delegiranjem posla preko pokazivača na funkciju. Međutim,

takvim mehanizmom ne bismo mogli ostvariti nadogradivost s obzirom na različite vrste pretraživanih spremnika. U ovom zadatku ćemo istražiti kako takvu funkcionalnost postići predlošcima.

Za početak, izvedite identičnu funkcionalnost iz prethodnog zadatka, ali na način da nadogradivost bez promjene ostvarite funkcijskim predloškom (engl. template function). Oblikujte vaš predložak prema primjeru funkcije `find_if` standardne biblioteke:

```
template <typename Iterator, typename Predicate>
Iterator mymax(
  Iterator first, Iterator last, Predicate pred){
// ...
}
```

Obratite pažnju na to da drugi argument funkcije mymax treba pokazivati *iza* posljednjeg elementa polja, kao što je i uobičajeno u bibliotekama C++-a. Prednost takve konvencije je u tome što omogućava jasno predstavljanje praznih nizova: u tom slučaju imamo `first==last`.

Umjesto kriterijskim funkcijama koje primaju pokazivače na podatke (kao u prvom zadatku), ljepši kod dobit ćete s kriterijskim funkcijama koje primaju podatke ili reference na njih. Iskoristite mogućnost da pri pozivu predloška ne navedete parametre predloška, nego prepustite prevoditelju da ih pogodi sam, kao u sljedećem primjeru:

```
int arr_int[] = { 1, 3, 5, 7, 4, 6, 9, 2, 0 };
int main(){
  const int* maxint = mymax( &arr_int[0],
    &arr_int[sizeof(arr_int)/sizeof(*arr_int)], gt_int);
  std::cout <<*maxint <<"\n";
}
```

Pokažite da vaš predložak možete primijeniti i na stringove te na standardne spremnike C++-a `vector` i `set`. Komentirajte prednosti i nedostatke ove implementacije u odnosu na implementaciju iz prethodnog zadatka.

3. Nadogradnja bez promjene u Pythonu (15% bodova, Python)

Python je jezik koji omogućava fleksibilnije izražavanje od C-a i C++-a. Stoga je logično da ćemo i ovdje htjeti postići podršku različitih algoritama usporedbe te podršku za različite načine pohrane objekata. Vaš zadatak je napisati funkciju `mymax` koja pronalazi najveći element zadanog spremnika. Vaša funkcija treba biti primjenljiva na sve pobrojive objekte (engl. iterable object), odnosno na sve vrste spremnika i "spremnika" koje možemo obići naredbom `for` (liste, rječnike, generatore, ...). Pored toga, vaša funkcija treba omogućiti zadavanje svih zamislivih načina usporedbe elemenata.

S obzirom na to da je Python dinamički jezik, usporedbu ćemo modelirati funkcijskim argumentom `key` koji elemente pobrojivog objekta preslikava u objekte nad kojima je uređaj - dobro definiran. Za ugrađene tipove tako nećemo morati raditi ništa (jer su operatori usporedbe nad njima definirani), osim ako budemo htjeli izmijeniti kriterij usporedbe. Neka struktura vaše funkcije bude:

```
def mymax(iterable, key):
  # incijaliziraj maksimalni element i maksimalni ključ
  max_x=max_key=None

  # obiđi sve elemente
  for x in iterable:
    # ako je key(x) najveći -> ažuriraj max_x i max_key

  # vrati rezultat
  return max_x
```

Pokažite kako biste jednim pozivom vaše funkcije u listi stringova pronašli najdulju riječ. Argument `key` zadajte neimenovanom funkcijom koju ćete zadati ključnom rječju `lambda`. Sljedeći primjer ilustrira rad s bezimenim funkcijama:

```
# napravi bezimenu funkciju i poveži je s imenom f
  f = lambda x: 2*x+3

  # primijeni bezimenu funkciju
  f(3) # 9
```

Izmijenite zaglavlje funkcije `mymax` tako da omogućite njeno pozivanje sa samo jednim argumentom te neka se u tom slučaju kao kriterij koristi uređaj elemenata pobrojivog objekta. Ovu funkcionalnost ostvarite na način da argumentu `key` podrazumijevano dodijelite funkciju identiteta koju ćete zadati prikladnom neimenovanom funkcijom. Vaša funkcija trebala bi moći obraditi sljedeće upite:

```
maxint = mymax([1, 3, 5, 7, 4, 6, 9, 2, 0])
maxchar = mymax("Suncana strana ulice")
maxstring = mymax([
  "Gle", "malu", "vocku", "poslije", "kise",
  "Puna", "je", "kapi", "pa", "ih", "njise"])
```

Pronađite najskuplji proizvod u rječniku D koji sadrži cjenik pekare na uglu ulice:

```
D={'burek':8, 'buhtla':5}
```

Zadatak ostvarite tako da funkciji `mymax` kao ključ pošaljete metodu `get` rječnika `D`. Objasnite kako i zašto metodu možemo koristiti kao slobodnu funkciju.

Neka je kolekcija osoba zadana listom čiji su elementi parovi `(ime,prezime)`. Pronađite posljednju osobu prema leksikografskom poretku primjenom funkcije `mymax`. Pomoć: nad n-torkama Pythona je definiran uređaj tako da je `x<y` ako je `x[0]<y[0]` ili `x[0]==y[0] and x[1:]<y[1:]`, pri čemu `x[1:]` označava sve elemente n-torke počevši od indeksa 1 na dalje.

4. Generiranje slijeda brojeva i statističkih pokazatelja (15% bodova, bilo koji jezik)

Razmatramo komponentu `DistributionTester` čiji zadatak je generirati prikladni niz cijelih brojeva te ispisati 10., 20., ..., i 90. **percentil** njihove distribucije. Generiranje cijelih brojeva trebalo bi biti podržano na svaki od sljedećih načina:

- slijedno, u ovisnosti o zadanim granicama intervala i koraku uvećanja;
- slučajno, u ovisnosti o zadanim parametrima normalne distribucije i željenom broju elemenata;
- kao Fibonaccijev niz u ovisnosti o zadanom ukupnom broju elemenata.

Komponenta `DistributionTester` također treba podržavati određivanje p-tog percentila distribucije zadanog niza cijelih brojeva na svaki od sljedećih načina:

- kao element čiji je položaj u sortiranom nizu (počevši od 1) najbliži položaju percentila `n_p` definiranog s `n_p = p*N/100 + 0.5`, gdje N odgovara broju elemenata; primjerice, 80. percentil niza (1,10,50) bi u tom slučaju bio 50 (**detaljnije**).
- kao interpoliranu vrijednost između elemenata `v[i]` i `v[i+1]` za čije percentilne položaje vrijedi `p(v[i]) < p < p(v[i+1])`; percentilni položaj elementa `v_i` na rednom broju `i` računamo kao `p(v[i]) = 100*(i-0.5)/N`, gdje N odgovara broju elemenata, a redni broj `i` počinje od jedan; traženu interpoliranu vrijednost `v(p)` za zadani percentil `p` određujemo izrazom `v(p) = v[i] + N * (p-p(v[i]))*(v[i+1]-v[i])/100`; za percentile koji su manji od p(v[1]) odnosno veći od p(v[N]) vraćamo `v[1]` odnosno `v[N]`; primjerice, 80. percentil niza (1,10,50) bi u tom slučaju bio 46 (**detaljnije**).

Komponenta `DistributionTester` mora biti oblikovana na način da omogućava uključivanje drugih načina stvaranja cijelih brojeva i računanja percentila, i to bez potrebe za mijenjanjem same komponente.

Oblikujte rješenje problema u skladu s oblikovnim obrascem Strategija, i demonstrirajte funkcionalnost rješenja prikladnim ispitnim programom. Ispitni program treba stvoriti primjerak razreda `DistributionTester`, prikladno

ga konfigurirati, te pokrenuti obradu koja rezultira ispisom percentila distribucije.

5. Fleksibilno učitavanje i prikazivanje slijeda brojeva (15% bodova, bilo koji jezik)

Potrebno je ostvariti programsko rješenje sa sljedećim komponentama.

`SlijedBrojeva` je komponenta koja interno pohranjuje kolekciju cijelih brojeva. Pri stvaranju te komponente, kolekcija je prazna. Komponentu treba oblikovati na način da elemente dobiva od nekog izvora brojeva. U sustavu trebaju postojati različite implementacije izvora brojeva: `TipkovnickiIzvor` koji od korisnika učitava broj po broj s tipkovnice te `DatotecniIzvor` koji brojeve čita iz datoteke. Neka izvori svoje iscrpljivanje signaliziraju vraćanjem vrijednosti -1 (ili na neki drugi prikladan način). U svim ostalim slučajevima očekuje se da izvori uvijek generiraju nenegativne brojeve. Komponenta `SlijedBrojeva` treba biti oblikovana na način da je prilikom njezinog stvaranja moguće umetnuti odgovarajuć izvor brojeva. Rješenje također treba biti takvo da omogućava transparentno dodavanje novih izvora bez promjene koda komponente `SlijedBrojeva`. Razmislite o kojem se tu oblikovnom obrascu radi i implementirajte rješenje u skladu s njime. Komponenta `SlijedBrojeva` treba započeti preuzimanje brojeva od podešenog izvora kada se pozove metoda `kreni` koja potom svake sekunde od izvora pokuša preuzeti po jedan broj. Ako izvoru treba više vremena za generiranje broja, preuzimanje sljedećeg broja potrebno je obaviti jednu sekundu nakon završetka prethodnog čitanja, ma koliko ono trajalo. Programsko rješenje treba napisati na način da je prilikom svake promjene interne kolekcije komponente `SlijedBrojeva` moguće obaviti **jednu ili više** akcija. Akcije koje treba podržati su sljedeće:

1. u tekstovnu datoteku zapisati sve elemente koji se trenutno nalaze u kolekciji te datum i vrijeme zapisa;
2. temeljem elemenata koji se trenutno nalaze u kolekciji potrebno je na zaslon ispisati sumu svih elemenata;
3. temeljem elemenata koji se trenutno nalaze u kolekciji potrebno je na zaslon ispisati prosjek svih elemenata;
4. temeljem elemenata koji se trenutno nalaze u kolekciji potrebno je na zaslon ispisati medijan svih elemenata.

Rješenje treba biti takvo da omogućava konfiguriranje akcija koje treba poduzeti te transparentno dodavanje novih akcija bez potrebe za mijenjanjem komponente `SlijedBrojeva` (primjerice, stupčasti grafički prikaz i slično). Razmislite koji je oblikovni obrazac prikladan za rješavanje ovog problema i implementirajte rješenje u skladu s tim oblikovnim obrascem.

6. Tablični kalkulator (20% bodova, bilo koji jezik)

Tablični kalkulator sadrži tablicu polja koja mogu sadržavati ili konstantu ili matematički izraz. Matematički izrazi mogu referencirati vrijednosti drugih polja koja pak mogu ovisiti o vrijednostima trećih polja itd. Kad god se sadržaj nekog polja X promijeni potrebno je ponovo izračunati vrijednosti svih polja čiji izrazi neposredno ili posredno ovise o polju X.

Napišite programsko rješenje koje podržava zadavanje tablice s numeričkim konstantama i jednostavnim računskim izrazima (dovoljno je podržati zbrajanje s dva operanda), te ispisivanje njenog sadržaja. Rješenje mora omogućiti automatsko prosljeđivanje izmjena kroz proizvoljno dugačke lance ovisnosti. U slučaju kružnih ovisnosti, program treba baciti iznimku. Navedite kojem obrascu odgovara vaše rješenje te nacrtajte dijagram razreda.

Upute:

- Polja tablice modelirajte primjercima razreda `Cell`.
- Neka polje čuva svoj sadržaj u tekstnom podatkovnom članu `exp`. Primjerice, sadržaj može biti `"5"` ili `"A1+A2"`.
- Neka polje čuva cacheiranu vrijednost sadržaja u numeričkom podatkovnom članu `value`.
- Tablicu modelirajte razredom `Sheet` koji sadrži 2D polje objekata razreda `Cell`.
- Neka tablica ima metodu `cell(ref)` koja dohvaća referencu na polje zadano tekstnom adresom `ref`. Npr. `sheet.cell("A1")` vraća polje na koordinatama (0,0).
- Neka tablica ima metodu `set(ref, content)` koja sadržaj polja na adresi `ref` postavlja na tekst `content`.
- Neka tablica ima metodu `getrefs(cell)` koja vraća listu svih polja koja zadano polje referencira. Npr. ako vrijedi `cell.exp=="A3-B4"`, metoda treba vratiti polja na adresama `A3` i `B4`. Uputa: slobodno koristite neku biblioteku koja podržava regularne izraze (npr. standardni modul `re` Pythona).
- Neka tablica ima metodu `evaluate(cell)`, koja izračunava numeričku vrijednost zadanog polja. Uputa 1: radi jednostavnosti podržite samo zbrajanje. Uputa 2: slobodno koristite neku biblioteku koja podržava izračunavanje izraza. (npr. standardni modul `ast` Pythona).
- Propagiranje promjena provedite na razini ćelija, bez prozivanja tablice, osim za izračunavanje novih vrijednosti izraza. Kako biste omogućili pozivanje metode `evaluate`, svaka ćelija treba imati referencu na matičnu tablicu.

Ispravnost vašeg programa možete isprobati sljedećim ispitnim programom (Python):

```
if __name__=="__Main__":
  s=Sheet(5,5)
  print()

  s.set('A1','2')
  s.set('A2','5')
  s.set('A3','A1+A2')
  s.print()
  print()

  s.set('A1','4')
  s.set('A4','A1+A3')
  s.print()
  print()

  try:
    s.set('A1','A3')
  except RuntimeError as e:
    print("Caught exception:",e)
  s.print()
  print()
```

Na kraju navodimo kratki primjer za evaluiranje izraza modulom `ast`. Radi jednostavnosti, primjer radi samo za zbrajanje. **Ovdje** možete pogledati kako omogućiti i ostale operatore.

Prvo ćemo definirati funkciju za evaluiranje izraza s varijablama i operatorom `+`:

```
import ast

def eval_expression(exp, variables={}):
  def _eval(node):
    if isinstance(node, ast.Num):
      return node.n
    elif isinstance(node, ast.Name):
      return variables[node.id]
    elif isinstance(node, ast.BinOp):
      return _eval(node.left) + _eval(node.right)
    else:
      raise Exception('Unsupported type {}'.format(node))

  node = ast.parse(exp, mode='eval')
  return _eval(node.body)
```

Sada primjenom te funkcije možemo izračunavati izraze prema sljedećem primjeru:

```
# rječnik vrijednosti varijabli:
D={'a':5, 'b':3}
# definirajmo izraz s varijablama:
exp_var='a+b+a'
# izračunajmo vrijednost izraza:
rv = eval_expression(exp_var, D)
```

```
# 5+3+5=13
print(rv)
```

Izrađeno **vi**-jem i **gedit**om. Posljednja promjena: Friday, 12-Jan-2024 00:34:22 CET

Svi komentari su dobrodošli: sinisa.segvic@fer.hr

Povratak