Vežbe 1 - Uvod i C++ STL

Veljko Petrović

Uvod

Uvod

Osnovne informacije

Organizacija vežbe

- Vežbe se održavaju kombinovano kao mešavina vođenog i samostalnog rada
- Kada je rad vođen, preporučuje se da pratite šta vam asistent priča i da slušate instrukcije koje dobijete tokom tog perioda
- Kada je rad nezavistan od vas se očekuje da radite samostalno na nekom unapred datom problemu
- Ako ne učestvujete u nezavisnom radu, nećete puno naučiti na ovim vežbama.

Bodovi

- Na vežbama je u opticaju 70 bodova ukupno
- 40 nosi T1234 (ne pitajte za ime)
- 30 nosi SOV
- Nema drugog izvora bodova

Provere

- Provere će se raditi na času
- Termini još nisu sigurni ali očekujte T1234 negde oko nedelje 7 a SOV blizu kraja
- Trenutno, pravila dozvoljavaju da ponovimo jednu proveru van nastave što vam omogućava da popravite premali broj bodova da izađete na ispit
- Ovo ponavljanje nikako ne sme da vam bude plan ‘A’

Prepisivanje

- Prepisivanje ovde znači da predate u bilo kom trenutku bilo koji kod koji niste vi lično pisali
- Ako prepišete, nećete dobiti bodove i protiv vas se može povesti disciplinski postupak
- Verujte, ne isplati se.

Nedoumice

- Za bilo kakve nedoumice se obratite ili asistentu ili predmetnom nastavniku

Tehnički detalji

Rad kod kuće

- Vežba kod kuće je nešto što se apsolutno od vas očekuje
- Da bi to moglo treba vam okruženje kod kuće gde možete da raspolažete sa Linux-om, i odgovarajućim C++ kompajlerom.
- Treba vam, minimalno, `g++` verzija `9.4.0` (to je u laboratorijama).
- Kasnije verzije su OK.

Podešavanje okruženja kod kuće

- Pogledajte sledeća dva video zapisa koja pokazuju dva različita načina da, bezbolno, napravite sebi Linux okruženje u okviru Windows operativnog sistema

Tehnički detalji o radu

- Mi ovde koristimo efektivno samo dva alata: editor i kompajler
- Kompajler je C++ kompajler za GCC kolekciju poznat i kao g++
- Editor je štagod vi želite: Visual Studio Code koji bi trebao da vam je na raspolaganju je verovatno šta želite da koristite ali ste slobodni da upotrebljavate šta god želite.

Kompajliranje

- Većinu koda propuštamo kroz sledeću komandu:
  `g++ -pthread -o kod kod.cpp`
- Ovo poziva kompajler (`g++`) i kaže mu da će kod koristiti niti (`-pthread`) i to kroz mehanizam POSIX niti (na predavanjima ćete čuti više o ovome) što je za većinu naših programa istina (mada nije potrebno u ovim prvim časovima)
- Zatim specificira da će izlazni, izvršni, fajl biti imenovan ‘kod’
- Konačno specificramo da kompajliramo fajl izvornog koda `kod.cpp`

Kompajliranje

- Ovo kompajlira samo jedan fajl (većina našeg koda je takva da staje u jedan fajl)
- Naravno možemo kompajlirati odjednom i veliki broj fajlova koristeći isti ovaj mehanizam, mada je najbolje to uraditi preko posebnih mehanizama za automatizaciju komilacije softvera kao što je, npr. Make.
- Za sada, obično pozivanje g++ je dovoljno za naše potrebe.

Test C++ sposobnosti

Zadatak 1

- Napisati mali alat koji uzme ime fajla sa komandne linije i pita korisnika za PIN.
- PIN je dugačak tačno 1 bajt i može biti od 1 do 255.
- Program treba da generiše nov fajl koji ima isto ime kao uneseno sa dodatom ekstenzijom `.enc`
- Taj fajl ima u sebi svaki bajt ulaznog fajla XOR-ovan sa PIN vrednošću.

Proširenja na Z1 - Za vežbu

- Proširiti tako da korisnik može da unese 32-bitnu numeričku vrednost koja se primenjuje na svakih 4 bajta ulaznog fajla tako da se fajl neadekvatne veličine proširuje bajtovima nulte vrednosti dok veličina nije umnožak broja 4
- Proširiti tako da korisnik može da unese bilo koji string kao lozinku tako da se koriste individualni bajtovi lozinke na individualnim bajtovima ulaza tako da nema potrebe da se fajl proširi bilo čim.
- Proširiti tako da program detektuje da se koristi na šifrovanom fajlu i dešifruje ga

Mera uspeha

- Ako ne možete da rešite zadatak 1 u datom vremenu, potrebno je da osvežite vaše poznavanje C++ jezika inače će vam ovaj predmet biti previše težak.
- Ako možete da ga rešite, verovatno neće biti problema da pratite gradivo.
- Ako možete da odradite sva proširenja: svaka čast!

STL

Dokumentacija

- Dok se bavite upotrebom standardne biblioteke, obavezno se služite dokumentacijom
- cppreference je nešto što može biti od koristi
- cplusplus je takođe dobar izvor dokumentacije

Šta je STL?

- Standard Template Library - standardna biblioteka šablona
- Deo standardne biblioteke C++ koji se odnosi na strukture podataka i najčešće algoritme
- Ono što naročito karakteriše STL je uvođenje kontejnera kao koncepta i *iteratora*.

Opšta ideja iza iteratora

- Iterator je apstrakcija na svim tipovima koji služe da pokazuju unutar struktura podataka
- To obuhvata indekse, ključeve, i ključno **pokazivače.**
- Omogućava da se, recimo, implementiraju algoritmi koji rade sa bilo kojom strukturom podataka

Opšta ideja iza iteratora

- To je zato što umesto da prosledite celu strukturu, vi pošaljete iterator koji pokazuje na početak i iterator koji pokazuje na kraj
- Iterator sadrži informaciju o tipu sadržaja i interfejs koji nam dozvoljava da dobavimo tekuću vrednost, pređemo na sledeću, i proverimo da li su iteratori ekvivalentni i par sličnih jednostavnih operacija.
- Sa tim može da se napravi algoritam koji se nesmetano kreće kroz strukturu, a da ne zna baš ništa o tome kako je ona dizajnirana.

Kontejneri STL-a

- Kontejneri STL-a se dele na:
  - Sekvencijalne
  - Asocijativne
  - Neuređene asocijativne
  - Adaptere

Sekvencijalni kontejneri

- Sekvencijalni kontejneri su oni čiji je sadržaj poređan u nekakav definitivan redosled
- Razlikuju se po tome kakve garancije nude i kako im je sadržaj raspoređen u memoriji što ima uticaj na performanse

Sekvencijalni kontejneri

- `array` - niz statičke veličine kontinualan u memoriji
- `vector` - niz dinamičke veličine kontinualan u memoriji
- `deque` - ne-kontinualna struktura koja je takva da je dodavanje na njen kraj i početak brzo i nikada ne invalidira iteratore koji pokazuju u unutrašnjost
- `forward_list` - jednostruko spregnuta lista
- `list` - dvostruko spregnuta lista

Primer: vektor

```
void vektor_primer() {
  vector<int> v = {3, 1, 4, 1, 5, 9, 2};
  for (auto it = v.begin(); it != v.end(); it++) {
    cout << *it << endl;
  }
  v.push_back(99);
  v.push_back(98);
  v.insert(v.begin() + 2, 77);
  v.pop_back();
  for (auto it = v.begin(); it != v.end(); it++) {
    cout << *it << endl;
  }
}
```

Primer: vektor

- Primetite da se STL kontejneri mogu statički inicijalizovati
- Takođe primetite da imamo potpunu kontrolu nad sadržajem vektora: jedino je problem što je ubacivanje kao ovo koje smo uradili *sporo.*
- Prikazan je idiomatski način na koji se radi sa iteratorima: obratite pažnju ‘auto’ time dajemo kompajleru priliku da sam zaključi koji je tip: ovo se zove ‘type inference’ i predstavlja jako korisnu osobinu budući da je stvarni tip

  `vector<int>::iterator`

Asocijativni kontejneri

- Asocijativni kontejneri su kontejneri gde umesto da element ima mesto koje je definisano redosledom, element se adresira kroz nekaku proizvoljnu vrednost koja se zove *ključ*.
- Za razliku od svakog drugog jezika sa kojim imate iskustva, u C++-u asocijativni kontejneri *nisu* bazirani na heš strukturama no se jednostavno čuvaju *sortirani* po ključu što garantuje $n \cdot log(n)$ vreme pretrage.

Asocijativni kontejneri

- `set` - Kolekcija jedinstvenih ključeva i ničeg više
- `map` - Kolekcija parova tipa ključ-vrednost, ključevi su jedinstveni
- `multiset` - Kao `set` ali ključevi nisu jedinstveni.
- `multimap` - Kao `map` ali ključevi nisu jedinstveni.

Primer: Mapa

```
void mapa_primer() {
  map<string, int> m = {{"abc", 1}, {"def", 2},
  {"xyz", 907}};
  for (const auto &n : m) {
    cout << "m[" << n.first << "] = "
    << n.second << endl;
  }
  m["abc"] = 5;
  m["qux"] = 9;

  for (const auto &n : m) {
    cout << "m[" << n.first << "] = "
    << n.second << endl;
  }
}
```

Nova for petlja

- Umesto da direktno koristimo iteratore (budući da je prolaženje kroz sve elemente toliko često) možemo da koristimo for-each verziju for petlje kao ovde.
- Primetite da dobijamo i ključ (to je ovde `.first`) i vrednost (to je ovde `.second`)
- Ovo može, ako vam je verzija kompajlera dovoljno sveža (probajte!) i elegantnije kroz *destruktuiranje*

For sa destruktuiranjem

```
for (const auto& [k, v] : m){
   cout << "m[" << k << "] = "
   << v << endl;
}
```

Neuređeni asocijativni kontejneri

- Ovo su iste strukture kao i već pomenute samo što su `unordered_set`, `unordered_map`, `unordered_multiset` i `unordered_multimap` takve da čuvaju ključeve preko heš vrednosti.
- Ovo daje jako dobro prosečno vreme (Efektivno $O(1)$) sa rizikom da će biti $O(n)$ u najgorem mogućem slučaju.

Adapteri

- Adapteri služe da sekvencijalnim kontejnerima pruže drugačiji interfejs takav da odgovara ograničenjima nekih dobro poznatih struktura podataka
- Primeri su `stack`, `queue`, i `priority_queue`

Primer: Stek

```
void stek_primer() {
  stack<int> s;
  s.push(3);
  s.push(1);
  s.push(4);
  s.push(1);
  s.push(5);
  s.push(9);
  s.push(2);
  s.push(6);
  while (!s.empty()) {
    cout << s.top() << endl;
    s.pop();
  }
}
```

Algoritmi

Svrha biblioteke algoritama

- Vrlo je čest slučaj da rešenje programerskog problema može da se sastavi iz gradivnih elemenata (elementarnih algoritama) koji se stalno ponavljaju
- Svrha ove biblioteke jeste da se ti maksimalno generički algoritmi izdvoje na jedno mesto, implementiraju jako kvalitetno i koriste u rešenjima
- Nemoguće je proći sve algoritme: zato služi dokumentacija. Umesto toga možemo da pogledamo primer.

Primer: Suma svih brojeva deljivih sa 3 do 1024

```
bool div_three(int i) { return i % 3 == 0; }
void algo_primer() {
  vector<int> v(1024);
  vector<int> vv(1024);
  iota(v.begin(), v.end(), 1);
  copy_if(v.begin(), v.end(), vv.begin(), div_three);
  int r = accumulate(vv.begin(), vv.end(), 0);
  cout << r << endl;
}
```

Primer

- Ovde se koristi `iota` algoritam da napuni neku strukturu sa (redom) elementima od neke početne vrednosti, ovde to su vrednosti počevši od 1. Znamo da će ići do 1024 zato što smo napravili da bude tolika struktura.
- `copy_if` kopira u novu strukturu sve one elemente koje zadovoljavaju neki predikat - ovde to je eksplicitna funkcija: u praksi mnogo verovatnije bi se koristila *lambda* funkcija.
- `accumulate` je automatski algoritam za sumiranje strukture koji može da se adaptira da radi sa bilo kojom binarnom operacijom