Вежбе 2 - C++ STL

Радован Туровић

Тест Ц++ способности

Задатак 1

- Написати мали алат који узме име датотека са командне линије и пита корисника за ПИН.
- ПИН је дугачак тачно 1 бајт и може бити број од 1 до 255.
- Програм треба да генерише нову датотеку која има исто име као унешено али са додатом екстензијом `.enc`
- Та датотека има у себи сваки бајт улазне датотеке XOR-ован ПИН вредношћу.

Проширења на 31 - За вежбу

- Проширити тако да корисник може да унесе 32-битну нумеричку вредност која се примењује на сваких 4 бајта улазне датотеке тако да се датотека неадекватне величине проширује бајтовима нулте вредности док величина није умножак броја 4
- Проширити тако да корисник може да унесе било који стринг као лозинку тако да се користе индивидуални бајтови лозинке на индивидуалним бајтовима улаза тако да нема потребе да се датотека прошири било чим.
- Проширити тако да програм детектује да се користи на шифрованом датотеком и дешифрује је

Мера успеха

- Ако не можете да решите задатак 1 у датом времену, потребно је да освежите ваше познавање Ц++ језика иначе ће вам овај предмет бити превише тежак.
- Ако можете да га решите, вероватно неће бити проблема да пратите градиво.
- Ако можете да одрадите сва проширења: свака част!

STL (Стандардна библиотека шаблона)

Документација

- Док се бавите употребом стандардне библиотеке, обавезно се служите документацијом
- cppreference је нешто што може бити од користи
- cplusplus је такође добар извор документације

Шта је STL?

- *Standard Template Library* - стандардна библиотека шаблона
- Део стандардне библиотеке C++ који се односи на структуре података и најчешће алгоритме
- Оно што нарочито карактерише STL је увођење контејнера као концепта и *итератора.*

Општа идеја иза итератора

- Итератор је апстракција на свим типовима који служе да показују унутар структура података
- То обухвата индексе, кључеве, и кључно **показиваче.**
- омогућава да се, рецимо, имплементирају алгоритми који раде са било којом структуром података

Општа идеја иза итератора

- То је зато што уместо да проследите целу структуру, ви пошаљете итератор који показује на почетак и итератор који показује на крај
- Итератор садржи информацију о типу садржаја и интерфејс који нам дозвољава да добавимо текућу вредност, пређемо на следећу, и проверимо да ли су итератори еквивалентни и пар сличних једноставних операција.
- Са тим може да се направи алгоритам који се несметано креће кроз структуру, а да не зна баш ништа о томе како је она дизајнирана.

Контејнери STL-а

- Контејнери STL-а се деле на:
  - Секвенцијалне
  - Асоцијативне
  - Неуређене асоцијативне
  - Адаптере

Секвенцијални контејнери

- Секвенцијални контејнери су они чији је садржај поређан у некакав дефинитиван редослед
- Разликују се по томе какве гаранције нуде и како им је садржај распоређен у меморији што има утицај на перформансе

Секвенцијални контејнери

- `array` - низ статичке величине континуалан у меморији
- `vector` - низ динамичке величине континуалан у меморији
- `deque` - не-континуална структура динамичке величине
- `forward_list` - једноструко спрегнута листа
- `list` - двоструко спрегнута листа

Пример: вектор

```
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;
}
```

Пример: вектор

- Приметите да се STL контејнери могу статички иницијализовати
- Такође приметите да имамо потпуну контролу над садржајем вектора: једино је проблем што је убацивање као ово које смо урадили *споро.*
- Приказан је идиоматски начин на који се ради са итераторима: обратите пажњу на ‘auto’; тиме дајемо компајлеру прилику да сам закључи који је тип: ово се зове ‘type inference’ (откривање типа) и представља јако корисну особину будући да је стварни тип `vector<int>::iterator`

Асоцијативни контејнери

- Асоцијативни контејнери су контејнери где уместо да елемент има место које је дефинисано редоследом, елемент се адресира кроз некакву произвољну вредност која се зове *кључ*
- За разлику од сваког другог језика са којим имате искуства, у C++-у асоцијативни контејнери *нису* базирани на хеш структурама но се једноставно чувају *сортирани* по кључу што гарантује $log(n)$ време претраге

Асоцијативни контејнери

- `set` - Колекција јединствених кључева и ничег више
- `map` - Колекција парова типа кључ-вредност, кључеви су јединствени
- `multiset` - Као `set` али кључеви нису јединствени
- `multimap` - Као `map` али кључеви нису јединствени

Пример: мапа

```
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;
}
```

Нова for петља

- Уместо да директно користимо итераторе (будући да је пролажење кроз све елементе толико често) можемо да користимо `for-each` верзију for петље као овде
- Приметите да добијамо и кључ (то је овде `.first`) и вредност (то је овде `.second`)
- Ово може, ако вам је верзија компајлера довољно свежа (пробајте!) и елегантније кроз *деструктуирање*

For са деструктуирањем

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

Неуређени асоцијативни контејнери

- Ово су исте структуре као и већ поменуте само што су `unordered_set`, `unordered_map`, `unordered_multiset` и `unordered_multimap` такве да чувају кључеве преко хеш вредности.
- Ово даје јако добро просечно време (ефективно $O(1)$) са ризиком да ће бити $O(n)$ у најгорем могућем случају.

Адаптери

- Адаптери служе да секвенцијалним контејнерима пруже другачији интерфејс такав да одговара ограничењима неких добро познатих структура података
- Примери су `stack`, `queue`, и `priority_queue`

Primer: Stek

```
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();
}
```

Алгоритми

Сврха библиотеке алгоритама

- Врло је чест случај да решење програмерског проблема може да се састави из градивних елемената (елементарних алгоритама) који се стално понављају
- Сврха ове библиотеке јесте да се ти максимално генерички алгоритми издвоје на једно место, имплементирају јако квалитетно и користе у решењима
- Немогуће је проћи све алгоритме: зато служи документација; уместо тога можемо да погледамо пример

Пример: Сума свих бројева дељивих са 3 до 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;
}
```

Пример

- Овде се користи `iota` алгоритам да напуни неку структуру са (редом) елементима од неке почетне вредности, овде то су вредности почевши од 1; знамо да ће ићи до 1024 зато што смо направили да буде толика структура
- `copy_if` копира у нову структуру све оне елементе који задовољавају неки предикат - овде то је експлицитна функција: у пракси много вероватније би се користила *lambda* функција.
- `accumulate` је аутоматски алгоритам за сумирање структуре који може да се адаптира да ради са било којом бинарном операцијом