Semestrálny projekt Programovanie (3)
dokumentácia

Erik Róbert Ján Jakubovský
jakubovsky2@uniba.sk

2.2.2020

Projekt implementuje asociatívne pole s otvoreným adresovaním a metódy na čítanie a zápis JSON súborov.

Na adresovanie používa uzavreté hashovanie, dvojice kľúč, hodnota reprezentuje dátovou štruktúrou Item, ktoré sú uchované v dátovej štruktúre DynamicArray. K týmto dvojiciam sa pristupuje pomocou číselného identifikátora generovaným štruktúrou std::hash. Asociatívne pole je možné načítať a zapísať do súboru typu JSON.

Pri čítaní a zapisovaní JSON súborov riešim problém rôznych foriem zápisov špeciálnych symbolov pre c++ a JSON, ako su úvodzovky v reťazcoch, alebo symboly nových riadkov. Taktiež sa snažím, čítať súbory s vnorenými slovníkmi. Samozrejme, treba pred volaním funkcie vytvoriť vhodnú formu viacnásobne vnoreného slovníka.

Triedy a metódy sú poväčšinou písané v header súboroch kvôli šablónovým typom.

Tento projekt využíva súbory:

- iostream
- sstream
- fstream
- string
- functional (štruktúra hash)

A mnou definované súbory:

- ArrayException.h
- DynamicArray.h
- Item.h
- AssociativeArray.h
- ReadJSON.h
- ReadJSON.cpp
- WriteJSON.h
- WriteJSON.cpp

A testy:

- testDynamicArray.cpp
- testItem.cpp
- testAssociativeArray.cpp
- testToJSON.cpp
- testReadJSON.cpp

Trieda Exception:

- Slúži na vyhadzovanie výnimiek a generovanie chybových hlásení.
- Atribút message slúži na uchovanie správy.

Metódy:

- Exception(const string &text):
  - je konštruktorom triedy, za text sa dosadzuje chybové hlásenie.
- Const string &getMessage() const:
  - vracia referenciu na chybové hlásenie.

Trieda Item:

- Reprezentuje dvojice kľúč, hodnota v dynamickom poli.
- Atribút identifier je kľúč ľubovoľného hashovateľného typu.
- Atribút element je hodnota ľubovoľného typu.
- Atribút type, určuje či je dvojica obsadená, voľná, alebo prázdna, pre účely obsadzovania dynamického poľa.

Metódy:

- Item():
    - Default konštruktor slúži na inicializáciu prázdnych polí.

- Item(const Key key, const Value value):
    - Konštruktor triedy Item.

- Key get_key():
    - Vráti hodnotu kľuča.

- Key& key():
    - Vráti referenciu na atribút identifier.

- Void set_key(const Key new_key):
    - Zmení hodnotu atribútu identifier.

- Value get_value():
    - Vráti hodnotu.

- Value& value():
    - Vráti referenciu na atribút element.

- Void set_value(const Value new_value):
    - Zmení hodnotu atribútu element.

- void set_type(const ItemType newType):
    - Zmení atribút type na newType.

- const ItemType get_type() const:
    - Vráti hodnotu atribútu type.

- bool is_empty() const:
    - Vráti true ak je type iný ako ItemType::FULL, teda prázdny.

- bool is_full() const:
    - Vráti true ak je type ItemType::FULL, teda plný.

Trieda DynamicArray:

- Je pole ktoré si vie dynamicky alokovať pamäť, je možné ho indexovať a škálovať.
- Atribút array je smerník na pole, size určuje jeho veľkosť nerátajúc poslednú položku.

Metódy:

- DynamicArray(int length=10):
  - Vytvorí pole dĺžky length + 1, kde práve posledný prvok slúži na reprezentovanie konca poľa, pri práci so smerníkmi.

- DynamicArray(DynamicArray<Type> &other):
  - Kopírovací konštruktor.

- DynamicArray(DynamicArray<Type> &&other):
  - Presúvací konštruktor.

- Type& operator[](int index):
  - Operátor slúži na indexovanie poľa, pri zadaní indexu z iného rozsahu ako 0 až veľkosť poľa – 1 vyhodí výnimku.

- DynamicArray<Type>& operator=(DynamicArray<Type> &other):
  - Umožňuje skopírovať iné pole do cieľového.

- void resize(int new_size):
  - Zmení veľkosť poľa na new_size + 1, pôvodný obsah si neuchová.

- void resize():
  - Zmení veľkosť poľa na dvojnásobok pôvodného + 1, pôvodný obsah si neuchová.

- int get_size() const:
  - Vráti veľkosť poľa, nerátajúc prvok na poslednej pozícii.

- Type* begin():
  - Vráti smerník na začiatok poľa, s ktorým možno robiť smerníkovú aritmetiku.

- Type* end():
  - Vráti smerník na posledný prvok poľa, tento prvok je neindexovateľný, slúži iba ako informácia ktorá adresa poľa je posledná.
  - Pomocou metód begin a end možno iterovať pole podobne ako štruktúry stl.

- ~DynamicArray():
  - Deštruktor, uvoľní pamäť obsadenú dynamickým poľom.

Trieda Dict:

- Implementácia asociatívneho poľa.
- Atribút numberOfItems určuje počet neprázdnych položiek v poli, teda počet hodnôt v poli, loadCapacity určuje pomer počtu položiek a veľkosti poľa, je to číslo z intervalu (0, 1), ktoré ak sa prekročí tak pole zväčší svoju veľkosť na dvojnásobnú pôvodného. Atribút array je dynamické pole v ktorom sa uchovávajú dvojice kľúč, hodnota.

Metódy:

- Void _resize(const int new_size):
    - Metóda zväčší veľkosť poľa array pri prekročení loadCapacity, pôvodné hodnoty vloží späť do slovníka. Metóda je privátna pretože pri nesprávnom zaobchádzaní môže „poškodiť“ slovník.

- Item<KeyType, ValueType>& _find(const KeyType key):
    - Vráti referenciu na Item, ktorý hľadá z kľúča. Metóda je privátna a pri nesprávnom zaobchádzaní môže „poškodiť“ pole. Metóda vyráta hash, ktorým indexuje pole array, ak sa kľúč položky nezhoduje s tým hľadaným posunie sa k ďalšej položke, takto pokračuje kým nenájde položku so zhodným kľúčom alebo prvú neobsadenú položku. Ak metóda nenašla kľúč vráti prednostne prvú uvoľnenú položku, inak prvú neobsadenú položku.

- Dict(const double capacity=0.6):
    - Konštruktor prázdneho slovníka.

- int size() const:
    - Vráti počet obsadených položiek v slovníku, teda počet dvojíc kľúč, hodnota.

- void insert(const KeyType key, ValueType value):
    - Vloží položku do slovníka, podľa kľúča. Ak kľuč neexistuje vyhodí výnimku.
    - Ak položka s kľúčom key existuje, tak aktualizuje len jej value;

- ValueType& get_value(const KeyType key):
    - Vráti referenciu hodnoty, podľa kľúča. Ak kľuč neexistuje vyhodí výnimku.

- void remove(const KeyType key):
    - Uvoľní zo slovníka hodnotu, podľa kľúča, táto pozícia bude pri vkladaní nového kľúča uprednostnená oproti prázdnemu miestu. Ak kľuč neexistuje vyhodí výnimku.

- bool contains(const KeyType key):
    - Vráti true ak v slovníku existuje položka s kľúčom key.

- ValueType& operator[](const KeyType key):
    - Metóda kombinuje funkčnosť metódy insert a get_value. Ak položka s kľúčom key neexistuje tak ju priradí a vráti referenciu na jej hodnotu, inak vráti referenciu na hodnotu už existujúcej položky.

- DynamicArray<KeyType>& keys():
    - Metóda vráti inštanciu triedy DynamicArray obsahujúcej všetky kľúče, slovníka.

Funkcie na čítanie JSON súborov:

- string convertToSpecialCharacters(string text):
    - Funkcia slúži na konvertovanie JSON notácie špeciálnych znakov do štandardnej notácie c++.

- string trim (string text, char character):
    - Funkcia odstráni z konca a začiatku vstupného reťazca znak character.

- template <typename ValueType>
  istringstream& operator>> (istringstream& pairStream, Dict<string, ValueType> &dict):
    - Funkcia zo vstupného prúdu v tvare: ““<kluc>“ <hodnota>“ vytvorí string kľúč, hodnotu a zapíše túto dvojicu do slovníka.
    - V prípade, že je hodnota typu string, odstrihne z nej úvodzovky.

- template <typename ValueType>
  bool insertPair(Dict<string, ValueType> &dict, string pair):
    - Funkcia vyrobí zo vstupného reťazca v tvare ““<kluc>“: <hodnota>“, vstupný prúd v tvare ““<kluc>“ <hodnota>“, a cez funkciu >> vloží túto dvojicu do slovníka.

- template <typename ValueType>
  void read_JSON(Dict<string, ValueType> &dict, const char* fileName):
    - Funkcia otvorí súbor na čítanie, text zo súboru posúva ďalej na formátovanie.
    - Súbor načíta do jedného reťazca.
    - Výsledkom jej volania je naplnenie slovníka zo súboru JSON.

- template <typename ValueType>
  void insertPairs(string listOfPairs, Dict<string, ValueType> &dict):
    - Funkcia formátuje reťazec v tvare:

      ““{<kluc1>“: <hodnota1>, <kluc2>“: <hodnota2>, ...}“

    - Rozdeľuje ju na dvojice, ktoré ďalej formátuje, za účelom vloženia do slovníka:

- template <typename ValueType>
  void deconstructToPairs(string listOfValues, Dict<string,ValueType> &dict):
    - Funkcia rekurzívne volá samú seba a vnára sa do slovníka na základe kľúču. Spracováva reťazce listOfValues, ktoré reprezentujú vnorené slovníky, až kým nie sú vo formáte:

      ““{<kluc1>“: <hodnota1>, <kluc2>“: <hodnota2>, ...}“

    - Tie následne vkladá do slovníka.

Funkcie na zapisovanie JSON súborov:

- template <typename Type>
  Type convertFromSpecialCharacters(Type value):
  - Ak je typ hodnoty iný ako reťazec, tak funkcia túto hodnotu vráti.
  - Inak zamení špeciálne znaky jazyka c++ do špeciálnych znakov jazyka JSON. (\\ na \\\\, \n na \\n, ...)

- template <typename KeyType, typename ValueType>
  ostream& operator<< (ostream &os, Dict<KeyType, ValueType> &dict):
  - Funkcia vytvorí výstupný prúd z obsahu slovníka v tvare:
  - ““<kluc>“: <hodnota>“
    - Ak je ValueType string potom je hodnota v tvare “<hodnota>“

- template <typename KeyType, typename ValueType>
  bool to_JSON(Dict<KeyType, ValueType> &dict, const char* fileName)
  - Funkcia otvorí súbor a zapíše do neho obsah slovníka v JSON formáte.