Viacúrovňový jednosmerný lineárny zoznam obsahujúci štruktúrované dáta v C++ programe

Igor Košťál[1]

Abstrakt
Jednosmerný lineárny zoznam je dynamická dátová štruktúra, ktorá sa používa na ukladanie dát v aplikáciách. Vyznačuje sa veľmi efektívnym využívaním pamäťového priestoru aplikácie. Najpoužívanejšou verziou jednosmerného lineárneho zoznamu je tzv. jednoúrovňový zoznam. Existujú však aj viacúrovňové jednosmerné lineárne zoznamy, ktoré sa komplikovanejšie vytvárajú, avšak vyhľadávanie požadovaných dátových prvkov v nich je efektívnejšie. Existujú implementácie viacúrovňových jednosmerných lineárnych zoznamov v C programoch s jednoduchými, neštruktúrovanými dátami v ich dátových prvkoch (Niemann, 1999) (Pugh, 1990). My sme vytvorili C++ program, v ktorom je implementovaný viacúrovňový jednosmerný lineárny zoznam, ktorý v dátových prvkoch obsahuje štruktúrované dáta. V článku sa zaoberáme princípom a fungovaním viacúrovňového jednosmerného lineárneho zoznamu, jeho implementáciou so štruktúrovanými dátami v jeho dátových prvkoch v C++ programe, výstupmi a fungovaním tohto programu.

Kľúčové slová
Jednoúrovňový jednosmerný lineárny zoznam, viacúrovňový jednosmerný lineárny zoznam, dynamická dátová štruktúra, hlavička zoznamu, dátové prvky zoznamu

Abstract
One-way linked list is a dynamic data structure that is used for storing data in applications. It is very efficient using the application's memory space. The most widely used version of the one-way linked list is the single level linked list. However, there are also multi-level linked lists that are more elaborate, but the search for the required data elements in them is more efficient. There are implementations of multi-level linked lists in C programs with simple, unstructured data in their data elements (Niemann, 1999) (Pugh, 1990).We created a C ++ program that implements a multi-level linked list. It contains structured data in data elements. In the paper we deal with the principle and operation of a multi-level linked list, its implementation with structured data in its data elements in a C ++ program, outputs and operation of this program.

Key words
Single level linked list, multi-level linked list, dynamic data structure, header of list, data elements of list

JEL classification
C88

1 Úvod

Ako sme uviedli vyššie, jednosmerný lineárny zoznam je dynamická dátová štruktúra používaná na ukladanie dát v aplikáciách, ktorá veľmi efektívne využíva pamäťový priestor aplikácie. Je to dané spôsobom vytvárania a fungovania jednosmerného lineárneho zoznamu.

[1] Ing. Igor Košťál, PhD., Ekonomická univerzita v Bratislave, Fakulta hospodárskej informatiky, Katedra aplikovanej informatiky, Dolnozemská cesta 1/b, 852 35 Bratislava, igor.kostal@euba.sk

Ihneď po vytvorení môže byť zoznam prázdny a podľa potreby sú do neho postupne pridávané dátové prvky. Podobne je možné tento zoznam aj mazať po jednom dátovom prvku od konca alebo od jeho začiatku. Pre tieto jeho vlastnosti je jednosmerný lineárny zoznam používanou dynamickou dátovou štruktúrou v rôznych druhoch aplikácií. Väčšinou sa používa ako jednoúrovňový jednosmerný lineárny zoznam, čo je jeho jednoduchšia implementácia. Avšak je možné vytvárať aj viacúrovňové verzie takéhoto zoznamu. Ako sme uviedli vyššie, existujú implementácie viacúrovňových jednosmerných lineárnych zoznamov v C programoch s jednoduchými, neštruktúrovanými dátami v ich dátových prvkoch (Niemann, 1999) (Pugh, 1990). V programátorskej praxi použiteľnejšia verzia takéhoto viacúrovňového zoznamu obsahuje v dátových prvkoch štruktúrované dáta, napr. dáta študentov, ako sú napr. ich mená, priezviská, dátumy ich narodenia, bydliská, body na ubytovanie, čísla ich ISIC kariet atď. My sme vytvorili C++ program, ktorý používa viacúrovňový jednosmerný lineárny zoznam pre ukladanie štruktúrovaných dát, dát študentov, v jeho dátových prvkoch. Program, okrem samotného vytvárania a vkladania nových dátových prvkov obsahujúcich štruktúrované dáta do viacúrovňového jednosmerného lineárneho zoznamu, dokáže v tomto zozname vyhľadávať požadované dátové prvky, čiže študentov, podľa rôznych kritérií, napr. podľa bodov na ubytovanie, priezviska, čísla ISIC karty atď.

V nasledujúcich kapitolách článku sa zaoberáme princípom a fungovaním viacúrovňového jednosmerného lineárneho zoznamu, jeho implementáciou so štruktúrovanými dátami v jeho dátových prvkoch v C++ programe, výstupmi a fungovaním tohto programu.

2 Viacúrovňový jednosmerný lineárny zoznam, jeho princíp a fungovanie

Obecný jednosmerný lineárny zoznam musí obsahovať nasledujúce prvky:

- ***informačný záznam*** alebo ***prvok*** - pomocou neho sa musí dať jednoznačne určiť začiatok a koniec zoznamu. V rôznych implementáciách jednosmerného lineárneho zoznamu má tento prvok rôzny obsah. Napr. môže obsahovať dva ukazovatele, na prvý a posledný dátový prvok zoznamu. Alebo môže obsahovať ukazovateľ na počiatočný a zároveň ukončovací dátový prvok, ktorý neobsahuje relevantné dáta, pretože plní len uvedenú úlohu a, v prípade viacúrovňového jednosmerného lineárneho zoznamu, aktuálny počet úrovní (náš prípad).
- ***dátový prvok*** - ten, okrem samotných dát, ktoré môžu byť jednoduché alebo štruktúrované, obsahuje väzobný člen, čo je ukazovateľ ukazujúci na nasledujúci dátový prvok. V prípade viacúrovňového jednosmerného lineárneho zoznamu obsahuje dátový prvok nie jeden ukazovateľ, ale pole ukazovateľov, ktoré ukazujú nielen na nasledujúci dátový prvok, ale aj na ďalšie dátové prvky v zozname.

a musí mať nasledujúce vlastnosti, aby s ním dokázala aplikácia pracovať:

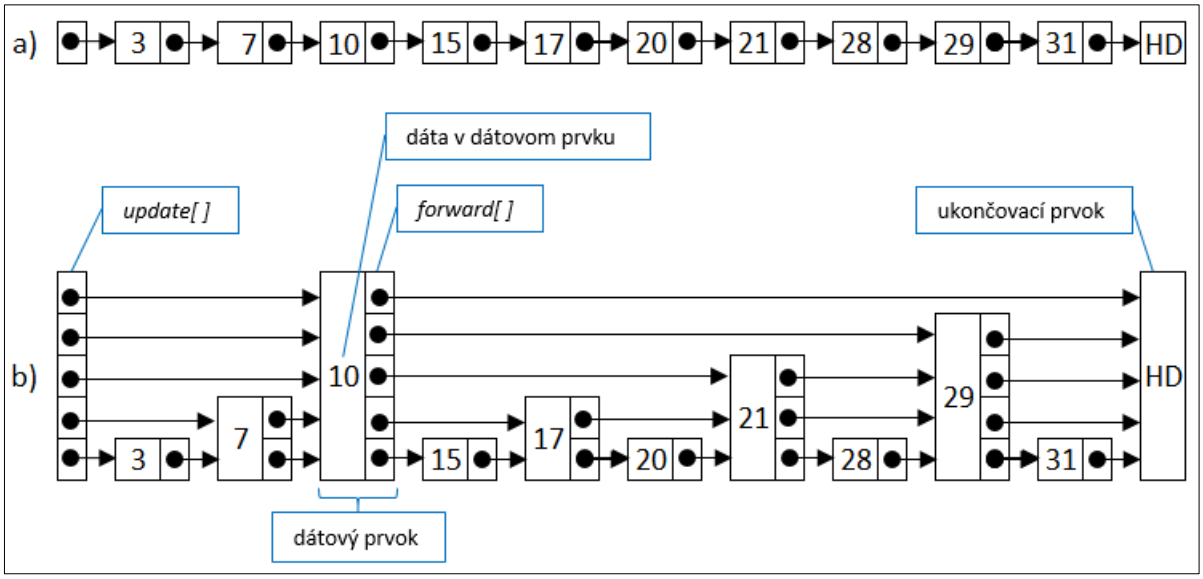
- vždy, aj v prípade keď je prázdny, musí obsahovať informačný záznam alebo prvok,
- ak zoznam obsahuje dátové prvky, tak väzobné členy prvého až predposledného dátového prvku musia obsahovať ukazovatele ukazujúce na nasledujúci dátový prvok.

Viacúrovňový jednosmerný lineárny zoznam, okrem uvedených prvkov a vlastností, obsahuje „globálne“ pole ukazovateľov, napr. *update[]*, ktoré obsahuje tzv. *úrovňové ukazovatele zoznamu*. Okrem toho jeho jednotlivé dátové prvky môžu mať aj viacero väzobných členov, čo sú takzvané *prvkové úrovňové ukazovatele* uložené v poli ukazovateľov, napr. *forward[]*.

Viacúrovňový (taktiež aj jednoúrovňový) jednosmerný lineárny zoznam môže byť počas fungovania aplikácie v nasledujúcich stavoch:

- zoznam je prázdny - obsahuje len informačný záznam alebo prvok a začiatočný a zároveň ukončovací prvok,
- zoznam obsahuje len jeden dátový prvok - vtedy zoznam obsahuje informačný záznam alebo prvok, jeden dátový prvok a začiatočný a zároveň ukončovací prvok. Väzobný člen dátového prvku ukazuje na začiatočný a zároveň ukončovací prvok a väzobný člen tohto prvku ukazuje na dátový prvok zoznamu.
- zoznam obsahuje dva a viac dátových prvkov- vtedy zoznam obsahuje informačný záznam alebo prvok, dátové prvky a začiatočný a zároveň ukončovací prvok. Väzobné členy prvého až predposledného dátového prvku musia obsahovať ukazovatele ukazujúce na nasledujúci dátový prvok. Väzobný člen posledného dátového prvku ukazuje na začiatočný a zároveň ukončovací prvok a väzobný člen tohto prvku ukazuje na prvý dátový prvok zoznamu.

Obr. 1: Zobrazenie 1-úrovňového (a)) a viacúrovňového (b)) jednosmerného lineárneho zoznamu s jednoduchými, neštruktúrovanými dátami v ich dátových prvkoch



Zdroj: Vlastné spracovanie inšpirované (Pugh, 1990)

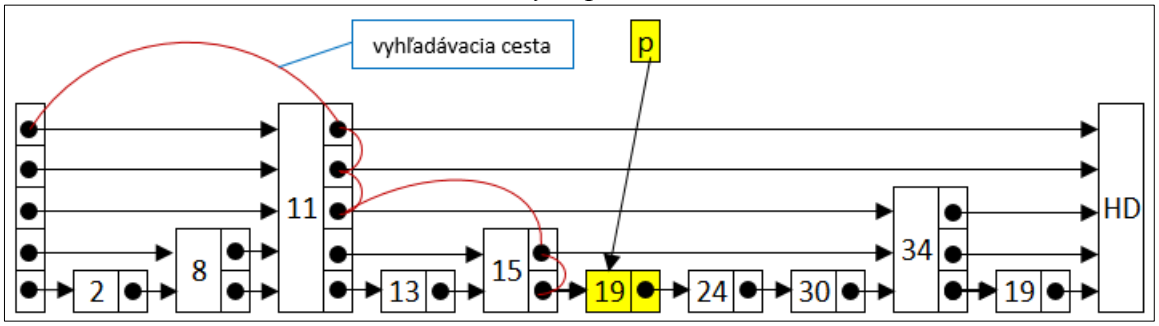
Po spustení aplikácie používajúcej jednosmerný lineárny zoznam, táto väčšinou najskôr načítava dáta z nejakého externého úložiska, napr. z diskového súboru, a následne ich ukladá do dátových prvkov zoznamu. Ak aplikácia ukladá tieto dáta do dátových prvkoch zoznamu usporiadane, vtedy je možné využiť na efektívne vyhľadávanie správneho miesta vloženia vkladaného dátového prvku viacúrovňový jednosmerný lineárny zoznam. Taktiež na vyhľadávanie požadovaných dát v usporiadanom zozname je možné využiť efektívny postup vyhľadávania požadovaného prvku vo viacúrovňovom jednosmernom lineárnom zozname.

Usporiadané *vkladanie nového dátového prvku* do ***jednoúrovňového jednosmerného lineárneho zoznamu*** vykoná vkladacia funkcia aplikácie tak, že sa posúva pomocou ukazovateľov uložených vo väzobných členoch jednotlivých dátových prvkov od prvého po posledný dátový prvok a hľadá vhodné miesto pre jeho vloženie. Keď takéto miesto vkladacia funkcia nájde, tak do neho vloží nový dátový prvok a aktualizuje väzobné členy (ukazovatele) susedných prvkov nového dátového prvku a väzobný člen jeho samotného. Pri takomto jednoúrovňovom jednosmernom lineárnom zozname musí vkladacia funkcia prejsť všetkými dátovými prvkami v zozname až na požadované miesto vloženia. Rovnako postupuje pri *vyhľadávaní požadovaného dátového prvku* v jednoúrovňovom jednosmernom lineárnom

zozname aj vyhľadávacia funkcia. Tiež musí prejsť všetky dátové prvky zoznamu od prvého až po hľadaný alebo po posledný dátový prvok, ak sa požadovaný prvok v zozname nenachádza.

Vkladacia funkcia aplikácie používajúcej ***viacúrovňový jednosmerný lineárny zoznam*** vkladá nový dátový prvok do zoznamu efektívnejším postupom. Vhodné miesto pre jeho vloženie začne hľadať od najvyššej úrovne zoznamu. Ak ho pomocou tejto najvyššej úrovne zoznamu nenájde tak si vkladacia funkcia zníži úroveň zoznamu o jednu nižšie a hľadá vhodné miesto pre vloženie nového dátového prvku na tejto zníženej úrovni zoznamu. Ak sa vkladacej funkcii nepodarí nájsť toto vhodné miesto na tejto zníženej úrovni zoznamu, zníži aktuálnu úroveň zoznamu znovu o jednu nižšie a pokračuje v hľadaní vhodného miesta vloženia nového dátového prvku na tejto zníženej úrovni zoznamu. Takto pokračuje vkladacia funkcia v znižovaní (dekrementovaní) aktuálnej úrovne zoznamu a v hľadaní vhodného miesta vloženia nového dátového prvku dovtedy, kým vhodné miesto vloženia nenájde. Na toto nájdené miesto v zozname napokon vkladacia funkcia vloží nový dátový prvok a aktualizuje väzobné členy (ukazovatele) susedných prvkov nového dátového prvku a väzobný člen jeho samotného. Počas hľadania vhodného miesta vloženia nového dátového prvku vkladacia funkcia s vysokou pravdepodobnosťou ***vynechá niektoré dátové prvky***. Pri rozsiahlych viacúrovňových jednosmerných lineárnych zoznamoch s veľkými počtami dátových prvkov, napr. s desať tisícami prvkov, je možné predpokladať merateľnú úsporu exekučného času spotrebovaného vkladacou funkciou aplikácie na hľadanie vhodného miesta vloženia nového dátového prvku oproti vkladacej funkcii pracujúcej s jednoúrovňovým jednosmerným lineárnym zoznamom. Rovnako postupuje pri vyhľadávaní požadovaného dátového prvku vo viacúrovňovom jednosmernom lineárnom zozname aj vyhľadávacia funkcia. V prípade *vyhľadávania požadovaného dátového prvku* v rozsiahlom viacúrovňovom jednosmernom lineárnom zozname (napr. v 10000-prvkovom) touto vyhľadávacou funkciou je možné predpokladať merateľnú úsporu exekučného času spotrebovaného touto funkciou oproti vyhľadávacej funkcii pracujúcej s jednoúrovňovým jednosmerným lineárnym zoznamom.

Obr. 2: Zobrazenie postupu vyhľadávania dátového prvku s hodnotou 19 vo viacúrovňovom jednosmernom lineárnom zozname s jednoduchými, neštruktúrovanými dátami v jeho dátových prvkoch



Zdroj: Vlastné spracovanie inšpirované (Pugh, 1990)

3 C++ aplikácia používajúca viacúrovňový jednosmerný lineárny zoznam so štruktúrovanými dátami v jeho dátových prvkoch

Naša C++ aplikácia, používajúca viacúrovňový jednosmerný lineárny zoznam so štruktúrovanými dátami v jeho dátových prvkoch, bola vyvinutá v neriadenom C++ jazyku ako konzolová aplikácia vo vývojovom prostredí Microsoft Visual Studio 2013. Jej zdrojový kód je rozdelený do nasledujúcich zdrojových súborov:

- *data_structures.h* - hlavičkový súbor obsahujúci deklarácie vymenovaného typu, nových štruktúrových dátových typov *nodeDataType*, *SkipList*, ktoré program

použije na vytvorenie nových dátových prvkov a hlavičky zoznamu a deklarácie dátových štruktúr *struct datum* a *struct TStudent*. Tie program použije na vytvorenie štruktúrovaných častí dátových prvkov zoznamu.

- *functions.cpp* - zdrojový súbor obsahujúci definície funkcií *insert_1st, find_1st_pts, find_1st_y, find_1st_surname* a *find_1st_isic*, ktoré dokážu vykonať príslušné operácie s dátovými prvkami viacúrovňového jednosmerného lineárneho zoznamu, ako sú vytvorenie 1 nového dátového prvku s požadovanými dátami študenta a jeho usporiadané vloženie do zoznamu, hľadanie 1 dátového prvku s dátami takého študenta, ktorý má v príslušnej vnútornej premennej jeho štruktúrovej premennej požadovaný počet bodov na ubytovanie, alebo požadovaný rok narodenia, alebo požadované priezvisko, alebo požadované číslo ISIC karty. Všetky tieto funkcie volajú funkciu *compare_st_LT*, ktorá im porovná 2 študentov podľa ňou určených priorít a kritérií (obr. 9). Ďalej tento zdrojový súbor obsahuje definíciu funkcie *display_1st*, ktorá zobrazí dáta 1 študenta uložené v dátovom prvku zoznamu, definíciu funkcie *display_sts_l*, ktorá zobrazí dáta všetkých študentov uložených v dátových prvkoch zoznamu, tiež definíciu funkcie *deleteK_st*, ktorá zmaže 1 dátový prvok s takými dátami študenta, ktorý má v príslušnej vnútornej premennej jeho štruktúrovej premennej požadovaný počet bodov na ubytovanie a tiež definíciu funkcie *initList*, ktorá inicializuje samotný viacúrovňový jednosmerný lineárny zoznam tým, že inicializuje vnútorné premenné *hdr* (ukazovateľ na počiatočný a zároveň ukončovací dátový prvok zoznamu) a *listLevel* (počiatočná úroveň zoznamu) existujúcej štruktúrovej premennej reprezentujúcej hlavičku zoznamu.
- *main.cpp* - hlavný zdrojový súbor obsahujúci funkciu *main*. Tá v svojom tele podľa potreby volá jednotlivé funkcie definované v zdrojovom súbore *functions.cpp*.

Jednotlivé ***časti*** a ***prvky viacúrovňového jednosmerného lineárneho zoznamu*** *so štruktúrovanými dátami študentov v jeho dátových prvkoch*, sú v našom programe implementované nasledovne.

Nové dátové prvky zoznamu sú vytvárané ako štruktúrové premenné novo vytvoreného štruktúrového dátového typu *nodeDataType* (obr. 3) definovaného v hlavičkovom súbore *data_structures.h*. Hlavička zoznamu je vytvorená ako štruktúrová premenná novo vytvoreného štruktúrového dátového typu *SkipList* (obr. 3) definovaného v hlavičkovom súbore *data_structures.h*.

Štruktúrovanými dátami v každom dátovom prvku viacúrovňového jednosmerného lineárneho zoznamu aplikácie sú nasledujúce dáta študenta:

- meno, priezvisko, bydlisko, dátum narodenia,
- body na ubytovanie, vzdialenosť bydliska v km od sídla školy,
- číslo ISIC karty.

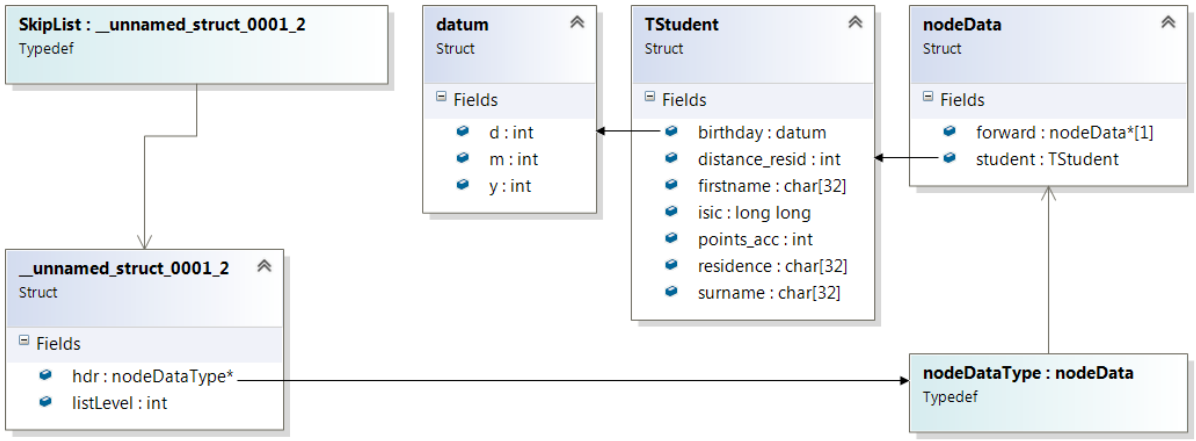
Všetky tieto dáta študenta sú uložené vo vnútorných štruktúrových premenných premennej *student*, ktorá je typu *TStudent* (obr. 3) a ktorá je vnútornou štruktúrovou premennou štruktúrovej premennej typu *nodeDataType* (obr. 3). Táto štruktúrová premenná reprezentuje 1 dátový prvok zoznamu v programe. Samotná štruktúra *TStudent* (obr. 3) a ňou vyžívaná štruktúra *datum* (obr. 3), sú definované v hlavičkovom súbore *data_structures.h*.

Obr. 3: Deklarácie vymenovaného typu, štruktúr a nových dátových typov v hlavičkovom súbore data_structures.h

```
//deklaracia vymenovaneho typu; konstanty tohto typu pouzivaju niektore funkcie, napr. 'insert_1st' a
//'find_1st_pts', ako navratove hodnoty
typedef enum {
  STATUS_OK,
  STATUS_MEM_EXHAUSTED,
  STATUS_KEY_NOT_FOUND
} statEnum;

//deklaracia 2 struktur; strukturove premenne takychto typov reprezentuju data studenta ulozeneho
//v datovom prvku zoznamu
struct datum {
  int d, m, y;
};

struct TStudent {
  char firstname[32], surname[32], residence[32];
  datum birthday;
  int points_acc, distance_resid;
  long long isic;
};

//deklaracia noveho datoveho typu 'nodeDataType'; strukturova premenna tohto typu reprezentuje
//1 datovy prvok zoznamu
typedef struct nodeData {
  TStudent student; //strukturova prem. 'student' obsahuje data studenta uloz. v datov. prvku zoznamu
  struct nodeData *forward[1]; //pole ukazovatelov reprezentuje vazobny clen datov. prvku zoznamu
} nodeDataType;

//deklaracia noveho datoveho typu 'SkipList'; strukturova premenna tohto typu reprezentuje hlavicku
//zoznamu
typedef struct {
  nodeDataType *hdr; //ukazovatel na pociatocny a zaroven ukoncovaci datovy prvok zoznamu
  int listLevel;          //aktualna uroven zoznamu
} SkipList;
```

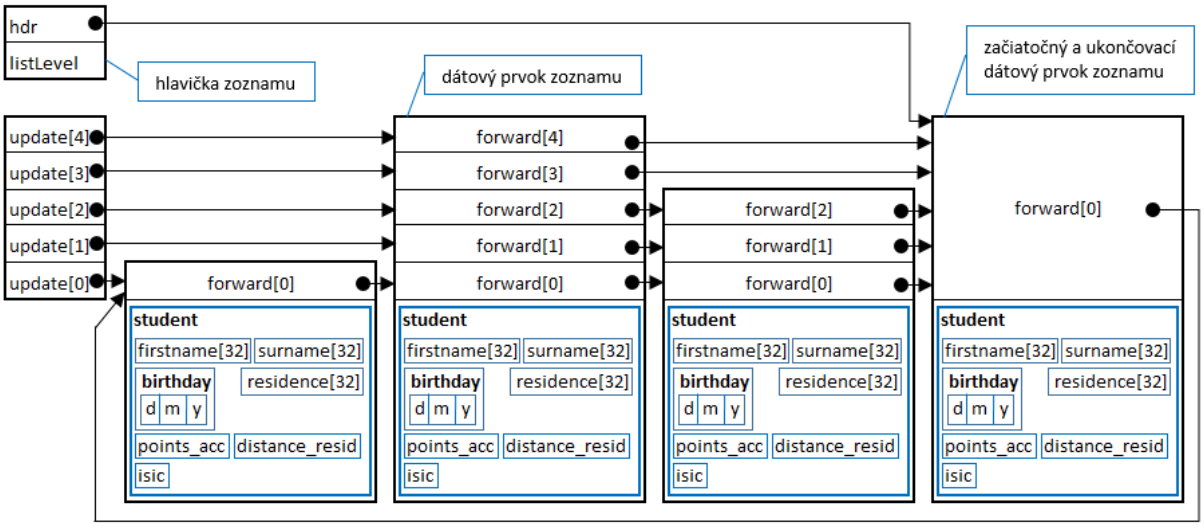
Zdroj: Vlastné spracovanie

Obr. 4: Vzájomné vzťahy štruktúr a nových dátových typov deklarovaných v hlavičkovom súbore data_structures.h



Zdroj: Vlastné spracovanie

Obr. 5: Viacúrovňový jednosmerný lineárny zoznam, ktorý používa náš program, s dátami študentov v jeho dátových prvkoch



Zdroj: Vlastné spracovanie

Požadované ***operácie s dátovými prvkami v zozname*** vykonáva náš program pomocou funkcií s nasledovnými deklaráciami, ktoré sú definované v zdrojovom súbore *functions.cpp*:

- *statEnum insert_1st(SkipList &list, TStudent st);* - funkcia vytvorí jeden nový dátový prvok s ukazovateľom *'nodeDataType *p'* s hodnotou vnútornej premennej *'p->student'* totožnej s parametrom funkcie *'st'* a vloží ho do zoznamu s referenciou *'&list'* usporiadane podľa priorít a kritérií určených funkciou *compare_st_LT* (obr. 9), ktorú zavolá funkcia *insert_1st* v svojom tele.
  Funkcia *insert_1st* je veľmi dôležitá pre usporiadané vkladanie nových dátových prvkov do zoznamu, čiže pre vytváranie samotného zoznamu, preto uvádzame aj jej celú definíciu.

Obr. 6: Definícia funkcie 'insert_1st' nachádzajúca sa v zdrojovom súbore functions.cpp

```
statEnum insert_1st(SkipList &list, TStudent st) {
  int i, newLevel;
  nodeDataType *update[MAXLEVEL + 1];
  nodeDataType *p; //deklaracia pomocneho ukazovatela

  /* hladanie miesta v zozname, kde bude novy datovy prvok vlozeny. Zacina sa hladat od najvyssej
     urovne zoznamu 'list.listLevel', napr. od urovne 2, postupne je uroven zozn. dekrementovana */
  p = list.hdr;
  for (i = list.listLevel; i >= 0; i--) {
      while (p->forward[i] != list.hdr && compare_st_LT(p->forward[i]->student, st))
          p = p->forward[i]; //ukazovatel 'p' zostane nastaveny na dat. prvok pred vkladanym dat. prvkom
      update[i] = p;
  }
  p = p->forward[0];

  srand((unsigned)time(NULL));

  //urcenie velkosti urovne zoznamu
  for (newLevel = 0; rand() < RAND_MAX / 2 && newLevel < MAXLEVEL; newLevel++);

  if (newLevel > list.listLevel) {
      for (i = list.listLevel + 1; i <= newLevel; i++)
          update[i] = list.hdr;
      list.listLevel = newLevel;
  }

  //alokovanie miesta pre novy datovy prvok zoznamu, cize jeho vytvorenie
  if ((p = (nodeDataType *)malloc(sizeof(nodeDataType) + newLevel*sizeof(nodeDataType *))) == 0)
      return STATUS_MEM_EXHAUSTED;
  p->student = st;

  //aktualizacia pola ukazovatelov (vazobneho clena) noveho datoveho prvku
  for (i = 0; i <= newLevel; i++) {
      p->forward[i] = update[i]->forward[i];
      update[i]->forward[i] = p;
  }
  return STATUS_OK;
}
```

Zdroj: Vlastné spracovanie inšpirované (Niemann, 1999)

- *statEnum find_1st_pts(SkipList list, TStudent st, int pts);* - funkcia hľadá v zozname *'list'* jeden dátový prvok s dátami takého študenta, ktorý má hodnotou vnútornej premennej *'p->student.points_acc'* totožnú s parametrom funkcie *'pts'*.
  Funkcia *find_1st_pts* je veľmi dôležitá pre vyhľadávanie dátového prvku zoznamu s dátami takého študenta, ktorý má v príslušnej vnútornej premennej jeho štruktúrovej premennej požadovaný počet bodov na ubytovanie, preto uvádzame aj jej celú definíciu.

Obr. 7: Definícia funkcie 'find_1st_pts' nachádzajúca sa v zdrojovom súbore functions.cpp

```
statEnum find_1st_pts(SkipList list, TStudent st, int pts) {
  int i;
  nodeDataType *p = list.hdr; //deklaracia a definicia pomocneho ukazovatela 'p'

  for (i = list.listLevel; i >= 0; i--) {
      while (p->forward[i] != list.hdr && compare_st_LT(p->forward[i]->student, st))
          p = p->forward[i]; //ukazovatel 'p' zostane nastaveny na dat. prvok pred hladanym prvkom
  }
  p = p->forward[0]; //do 'p' je vlozeny ukazovatel na hladany datovy prvok

  if (p != list.hdr && (p->student.points_acc == pts)) {
      display_1st(p); //zobrazenie dat najdeneho dat. prvku na konzolu zavolanou funkciou 'display_1st'
      return STATUS_OK;
  }

  return STATUS_KEY_NOT_FOUND;
}
```

Zdroj: Vlastné spracovanie inšpirované (Niemann, 1999)

V nasledujúcich odrážkach krátko opísané funkcie *find_1st_y*, *find_1st_surname* a *find_1st_isic*, ktoré hľadajú v zozname tiež jeden dátový prvok s dátami takého študenta, ktorý má v príslušnej vnútornej premennej jeho štruktúrovej premennej požadovaný rok narodenia, alebo požadované priezvisko, alebo požadované číslo ISIC karty, majú veľmi podobné definície ako funkcia *find_1st_pts*.

- *statEnum find_1st_y(SkipList list, TStudent st, int year);* - funkcia hľadá v zozname *'list'* jeden dátový prvok s dátami takého študenta, ktorý má hodnotou vnútornej premennej *'p->student.birthday.y'* totožnú s parametrom funkcie *'year'*.
- *statEnum find_1st_surname(SkipList list, TStudent st, char *srnm);* - funkcia hľadá v zozname *'list'* jeden dátový prvok s dátami takého študenta, ktorý má hodnotou vnútornej premennej *'p->student.surname'* totožnú s parametrom funkcie *'*srnm'*.
- *statEnum find_1st_isic(SkipList list, TStudent st, long long int isic_f);* - funkcia hľadá v zozname *'list'* jeden dátový prvok s dátami takého študenta, ktorý má hodnotou vnútornej premennej *'p->student.isic'* totožnú s parametrom funkcie *'isic_f'*.
- *void display_1st(nodeDataType *px);* - funkcia zobrazí dáta jedného študenta uložene v dátovom prvku, na ktorý ukazuje ukazovateľ *'nodeDataType *px'*.
- *statEnum display_sts_l(SkipList list, int s);* - funkcia zobrazí vnútorné premenné vnútorných premenných *'p->student'* všetkých dátových prvkov zoznamu *'list'* s veľkosťou *'s'* (počet dátových prvkov zoznamu *'list'*).
- *statEnum deleteK_st(SkipList &list, TStudent st);* - funkcia zmaže zo zoznamu s referenciou *'&list'* jeden dátový prvok, ktorý má hodnotou vnútornej premennej *'p->student.points_acc'* totožnú s vnútornou premennou parametra funkcie *'st.points_acc'*.

Kľúčovou časťou funkcií *insert_1st*, *find_1st_pts*, *find_1st_y*, *find_1st_surname*, *find_1st_isic* a *deleteK_st* je zdrojový kód, ktorý musí nájsť správne miesto vloženia nového dátového prvku do zoznamu, alebo miesto, kde sa nachádza hľadaný alebo mazaný prvok zoznamu. Na túto činnosť používajú uvedené funkcie veľmi podobný nasledujúci zdrojový kód, ktorý je z funkcie *find_1st_pts* (jeho činnosť je zrejmá z komentárov vložených v ňom).

Obr. 8: Vyhľadávací zdrojový kód z funkcie 'find_1st_pts' nachádzajúcej sa v zdrojovom súbore functions.cpp

```
for (i = list.listLevel; i >= 0; i--) {
      while (p->forward[i] != list.hdr && compare_st_LT(p->forward[i]->student, st))
          p = p->forward[i]; //ukazovatel 'p' zostane nastaveny na datovy prvok pred hladanym prvkom
  }
  p = p->forward[0]; //do 'p' je vlozeny ukazovatel na hladany datovy prvok
```

Zdroj: Vlastné spracovanie

Cyklus *while* v zloženom logickom výraze volá funkciu *compare_st_LT* (obr. 9), ktorá porovná dvoch študentov, ktorých dáta sú uložené v štruktúrových premenných *'p->forward[i]->student'* a *'st'* typu *TStudent*, ktoré táto funkcia dostala ako skutočné parametre. Porovnanie týchto dvoch študentov sa vykonáva podľa nasledujúcich priorít a kritérií:

- najskôr sú študenti porovnaní podľa bodov na ubytovanie,
- potom podľa vzdialenosti bydliska od školy,
- potom podľa dátumu narodenia,
- priezviska a
- nakoniec podľa mena.

Uvedené porovnanie dvoch študentov pomocou funkcie *compare_st_LT* (obr. 9), jedného existujúceho v zozname s dátami uloženými vo vnútornej premennej štruktúrovej premennej, na ktorú ukazuje ukazovateľ *'p->forward[i]->student'* a druhého hľadaného, ktorý má hodnotu vnútornej premennej *'p->student.points_acc'* totožnú s parametrom funkcie *'pts'*, využije napr. funkcia *find_1st_pts* pri hľadaní takéhoto študenta v zozname *'list'*.

Obr. 9: Definícia funkcie 'compare_st_LT' nachádzajúca sa v zdrojovom súbore functions.cpp

```
bool compare_st_LT(TStudent a, TStudent b) {
  int porovnanie;

  //porovnanie studentov podla bodov na ubytovanie
  if (a.points_acc > b.points_acc) return true;
  if (a.points_acc < b.points_acc) return false;

  //porovnanie studentov podla vzdialenosti ich bydliska od skoly
  if (a.distance_resid > b.distance_resid) return true;
  if (a.distance_resid < b.distance_resid) return false;

  long da, db; //porovnanie studentov podla datumu ich narodenia
  da = a.birthday.d + a.birthday.m * 100 + a.birthday.y * 10000;
  db = b.birthday.d + b.birthday.m * 100 + b.birthday.y * 10000;
  if (da<db) return false;
  if (da>db) return true;

  //porovnanie studentov podla priezviska
  porovnanie = strcmp(_strlwr(a.surname), _strlwr(b.surname));
  if (porovnanie < 0) return true;
  else return false;

  //porovnanie studentov podla mena
  porovnanie = strcmp(_strlwr(a.firstname), _strlwr(b.firstname));
  if (porovnanie < 0) return true;
  else return false;

  return false; }
```

Zdroj: Vlastné spracovanie

Taktiež funkcie *find_1st_y, find_1st_surname, find_1st_isic* používajú uvedené porovnanie dvoch študentov pre nájdenie dátového prvku vo viacúrovňovom jednosmernom lineárnom zozname *'list'* s dátami študenta, ktoré obsahujú požadovaný rok narodenia *'year'*, alebo požadované priezvisko *'*srnm'* alebo požadované číslo ISIC karty *'isic_f'* hľadaného študenta. Funkcia *insert_1st* používa taktiež uvedené porovnanie dvoch študentov pre nájdenie vhodného miesta v tomto zozname pri usporiadanom vkladaní nového dátového prvku s dátami vkladaného študenta do neho. Funkcia *deleteK_st* používa toto uvedené porovnanie dvoch študentov pre nájdenie dátového prvku na zmazanie z tohto zoznamu s dátami študenta, ktoré obsahujú číslo ISIC karty zadané používateľom programu.

Po ***spustení*** používateľom vykoná naša C++ aplikácia nasledujúce činnosti:

- nájde na pevnom disku súbor *studenti.txt*.

Obr. 10: Obsah diskového súboru studenti.txt s programom nespracovanými dátami študentov

```
Jan Jankovic 36130757022430100 4 6 1995 75 Piestany 65
Juraj Jankovic 36130397022430135 8 12 1997 77 Nove-Mesto-nad-Vahom 78
Ferdinand Prvy 36035723022430329 4 12 1996 89 Banska-Bystrica 139
Viliam Tell 36100427025140126 1 1 1997 93 Sturovo 112
Adriana Balajova 36097207739845768 22 6 1998 41 Nove-Zamky 98
Frantisek Sykora 36104278107495531 6 12 1999 77 Dubnica-nad-Vahom 137
Jan Sinaj 36015378227465325 12 4 1996 61 Dunajska-Streda 101
Andrej Sykora 36072128769834583 5 12 1999 77 Dubnica-nad-Vahom 137
Fabien Arbet 36108258148060224 11 2 1999 55 Bratislava 5
Adrian Diko 36099605837843210 13 4 1996 61 Dunajska-Streda 101
Peter Novak 36104758538294289 9 3 1997 69 Humenne 454
```

Zdroj: Vlastné spracovanie

Ak sa jej to podarí, tak v pamäťovom priestore aplikácie vytvorí dátový prúd a dáta z diskového súboru *studenti.txt* do neho okopíruje. Potom dáta z dátového prúdu povkladá do príslušných vnútorných premenných štruktúrových premenných uložených v poli štruktúrových premenných *'s'*. Uvedené činnosti vykoná nasledujúci zdrojový kód umiestnený vo funkcii *main*.

Obr. 11: Časť kódu z funkcie main vkladajúca dáta z dát. prúdu s ukazovateľ. 'in' do poľa 's'

```
ifstream in; //vytvorenie objektu 'in' kniznicnej triedy 'ifstream'
  int j = 0;
  in.open("studenti.txt"); //vytvorenie datoveho prudu s ukazovatelom 'in'
  if (!in) {
      cout << "Subor sa nepodarilo otvorit";
      exit(1);
  }

  TStudent s[100]; //deklaracia statickeho pola strukturovych premennych typu 'TStudent'

  //postupne citanie dat z datoveho prudu s ukazovatelom 'in' a ich zapisovanie do prislusnych
  //vnutornych premennych jednotlivych strukturovyh premennych ulozenych v poli 's'
  while (!in.eof()){
      in >> s[j].firstname >> s[j].surname >> s[j].isic;
      in >> s[j].birthday.d >> s[j].birthday.m >> s[j].birthday.y;
      in >> s[j].points_acc;
      in >> s[j].residence >> s[j].distance_resid;
      j++; //v 'j' je velkost zoznamu dana poctom jeho datovych prvkov
  }
```

Zdroj: Vlastné spracovanie

- Ďalej program sám, pomocou opakovaného volania funkcie *insert_1st*, usporiadane povkladá po jednom nové dátové prvky, vždy 1 nový dátový prvok s dátami 1 študenta uloženými v 1 prvku poľa štruktúrových premenných *'s'*, do zoznamu. Túto činnosť vykoná nasledujúci zdrojový kód umiestnený vo funkcii *main*.

Obr. 12: Časť kódu z funkcie main vkladajúca dáta pomocou funkcie insert_1st z poľa 's' do zoznamu 'list_st' usporiadane

```
for (i = 0; i < j; i++) {
  //funkcia 'insert_1st' vytvori z dat ulozenych v prvku pola 'TStudent s[i]' jeden novy datovy prvok
  //a vlozi ho do zoznamu 'list_st' usporiadane
  status = insert_1st(list_st, s[i]);
}
```

Zdroj: Vlastné spracovanie

- Potom program vypíše na konzolu usporiadaný zoznam študentov a vyzve používateľa na zadanie parametra dát študentov, podľa ktorých chce používateľ vyhľadávať študentov v zozname. Po vybratí tohto parametra požívateľom si program vypýta hodnotu tohto parametra u hľadaných študentov. Po jej správnom vložení program vo funkcii *main* opakovane zavolá príslušnú vyhľadávaciu funkciu, ktorá zo svojho tela, po úspešnom vyhľadaní požadovaného študenta, zavolá zobrazovaciu funkciu *display_1st*, ktorá vypíše dáta nájdeného študenta na konzolu. Opakovaným volaním príslušnej vyhľadávacej funkcie funkciou *main* sú vyhľadané, a postupne zobrazené, všetky výsledky vyhľadávania na konzolu. Potom sa program používateľa opýta, či chce pokračovať vo vyhľadávaní ďalších študentov. Ak používateľ odpovie *a* (áno), tak sa zopakuje v predchádzajúcej vete opísaná procedúra. Ak používateľ odpovie *n* (nie), tak program pomocou opakovaného volania funkcie *deleteK_st* postupne po jednom zmaže dátové prvky viacúrovňového jednosmerného lineárneho zoznamu so štruktúrovanými dátami študentov v jeho dátových prvkoch a skončí. Výstup programu s opísaným správaním môže vyzerať nasledovne.

Obr. 13: Výstupy C++ programu pri rôznych vstupoch používateľa (tučným písmom)

```
11 studentov vo viacurovnovom linearnom jednosmernom zozname usporiadanych podla bodov na
ubytovanie, vzdialenosti bydliska, datumu narodenia (od mladsich po starsich), priezviska a mena:

[          ISIC,          priezvisko, meno, body, bydlisko,                vzdialenost(km),datum narod.]
36100427025140126 Tell Viliam       (93) Sturovo                 (112) 1. 1. 1997
36035723022430329 Prvy Ferdinand    (89) Banska-Bystrica         (139) 4. 12. 1996
36104278107495531 Sykora Frantisek  (77) Dubnica-nad-Vahom       (137) 6. 12. 1999
36072128769834583 Sykora Andrej     (77) Dubnica-nad-Vahom       (137) 5. 12. 1999
36130397022430135 Jankovic Juraj    (77) Nove-Mesto-nad-Vahom    (78) 8. 12. 1997
36130757022430100 Jankovic Jan      (75) Piestany                (65) 4. 6. 1995
36104758538294289 Novak Peter       (69) Humenne                 (454) 9. 3. 1997
36099605837843210 Diko Adrian       (61) Dunajska-Streda         (101) 13. 4. 1996
36015378227465325 Sinaj Jan         (61) Dunajska-Streda         (101) 12. 4. 1996
36108258148060224 Arbet Fabien      (55) Bratislava              (5) 11. 2. 1999
36097207739845768 Balajova Adriana  (41) Nove-Zamky              (98) 22. 6. 1998

Podla coho chces vyhladavat studentov?
(podla bodov na ubytovanie(0) / roku narodenia(1) / priezviska(2) / ISIC(3)): 0
Vloz pocet bodov na ubytovanie, ktory maju mat hladani studenti: 77

36130397022430135 Jankovic Juraj    (77) Nove-Mesto-nad-Vahom    (78) 8. 12. 1997
36104278107495531 Sykora Frantisek  (77) Dubnica-nad-Vahom       (137) 6. 12. 1999
36072128769834583 Sykora Andrej     (77) Dubnica-nad-Vahom       (137) 5. 12. 1999
  3 najdeni studenti

Chces hladat dalsich studentov? (a/n): a

Podla coho chces vyhladavat studentov?
(podla bodov na ubytovanie(0) / roku narodenia(1) / priezviska(2) / ISIC(3)): 1
Vloz rok narodenia hladanych studentov: 1999

36104278107495531 Sykora Frantisek  (77) Dubnica-nad-Vahom       (137) 6. 12. 1999
36072128769834583 Sykora Andrej     (77) Dubnica-nad-Vahom       (137) 5. 12. 1999
36108258148060224 Arbet Fabien      (55) Bratislava              (5) 11. 2. 1999
  3 najdeni studenti

Chces hladat dalsich studentov? (a/n): a

Podla coho chces vyhladavat studentov?
(podla bodov na ubytovanie(0) / roku narodenia(1) / priezviska(2) / ISIC(3)): 2
Vloz priezvisko hladanych studentov: sykora

36104278107495531 Sykora Frantisek  (77) Dubnica-nad-Vahom       (137) 6. 12. 1999
36072128769834583 Sykora Andrej     (77) Dubnica-nad-Vahom       (137) 5. 12. 1999
  2 najdeni studenti

Chces hladat dalsich studentov? (a/n): a

Podla coho chces vyhladavat studentov?
(podla bodov na ubytovanie(0) / roku narodenia(1) / priezviska(2) / ISIC(3)): 3
Vloz cislo ISIC karty hladaneho studenta: 36015378227465325

36015378227465325 Sinaj Jan         (61) Dunajska-Streda         (101) 12. 4. 1996
  1 najdeny student

Chces hladat dalsich studentov? (a/n): n
```

Zdroj: Vlastné spracovanie

4 Záver

Viacúrovňový jednosmerný lineárny zoznam sa z pohľadu efektívneho využitia pamäťového priestoru aplikácie javí ako veľmi efektívna dynamická dátová štruktúra, dokonca pri vkladaní, vyhľadávaní a mazaní jeho dátových prvkov, ešte efektívnejšie fungujúca ako jednoúrovňový jednosmerný lineárny zoznam. V článku sme ukázali, že je možné tento viacúrovňový jednosmerný lineárny zoznam, aj so štruktúrovanými dátami v jeho dátových prvkoch, implementovať v C++ programe. Náš program poskytuje používateľovi vyhľadávacie funkcie, ktoré dokážu efektívne vyhľadať dátové prvky zoznamu s rôznymi požadovanými dátami študentov. Toto všetko demonštruje použiteľnosť viacúrovňového jednosmerného lineárneho zoznamu v C++ programe nielen s jednoduchými neštruktúrovanými dátami (Niemann, 1999) (Pugh, 1990) v jeho dátových prvkoch, ale aj so štruktúrovanými dátami v nich.

Literatúra

[1] Niemann, T. (1999). *Sorting and Searching Algorithms*. epaperpress.com.

[2] Pugh, W. (1990). *Skip Lists: A Probabilistic Alternative to Balanced Trees*. Communications of the ACM, 33(6).