TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta strojní

Bakalářská práce

Kruhové seznamy a jejich použití

Liberec 2006 Martin Brejcha

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta strojní

Studijní obor : 2301R030 Výrobní systémy

Zaměření: Inženýrská informatika

Katedra aplikované kybernetiky

Kruhové seznamy a jejich použití

Circular lists and their use

Martin Brejcha

Vedoucí bakalářské práce : prof. Ing. Vladimír Věchet, CSc.

Konzultant diplomové práce: Ing. Jan Váša

Anotace

Technická univerzita v Liberci

Fakulta strojní

Katedra aplikované kybernetiky

Studijní obor:	2301R030 Výrobní systémy
Studijní zaměření:	Inženýrská informatika
Diplomant:	Martin Brejcha
Téma práce:	Kruhové seznamy a jejich použití
Theme of the work:	Circular lists and their use
Rok obhajoby BP:	2006
Vedoucí BP:	prof. Ing. Vladimír Věchet CSc.
Konzultant BP:	Ing. Jan Váša

Stručný výtah:

Cílem bakalářské práce je popsat datovou strukturu kruhového seznamu a možných operací s ním, součástí je popis těchto operací. Vybrané operace jsou realizované pomocí funkcí zapsaných v programovacím jazyce C. Tyto funkce jsou zapsány v modulu Regiony.c, který umožňuje jakémukoliv programátorovi snadné použití těchto funkcí.

Abstract:

The aim of the bachelor work is description data structure of circular list and possible operations with the list, including description of such operations. Choosen operations are realized by functions with usage of C programming language. These functions are written in Region.c module, which allow easily use of these functions for any programator.

Poděkování

Na tomto místě bych chtěl poděkovat prof. Ing. Vladimírovu Věchetovi, CSc. a Ing. Janu Vášovi za odborné vedení, cenné rady, poskytnuté informace a za pomoc při zpracování diplomové práce. Děkuji.

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., Zákon o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), zejména §60 (školské dílo).

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

V Liberci dne

….....

Martin Brejcha

Obsah

1 Úvod

Cílem bakalářské práce je popis datových struktur seznamů a práce s vybranými operacemi s kruhovým seznamem v jazyce C. Zadání proto bude zpracováno do konkrétního příkladu použití operací s kruhovým seznamem.

K požadovanému výsledku lze dojít různými způsoby a proto uvedený příklad je jedním z možných a tak je také nutné k výsledku přistupovat. Jeden problém jde řešit mnoha různými cestami. Programátor by si měl vybrat cestu takovou, aby výsledný program byl co nejefektivnější, ale zároveň co nejspolehlivější. Věřím, že jsem si tuto cestu vybral. Programový kód, který zajišťuje stabilitu programu, totiž často mnohonásobně zvětší výsledný program. Na druhou stranu obvykle platí, čím delší program, tím je pomalejší.

Seznamy je vhodné požívat tam, kde neznáme na začátku množství proměnných. U kruhových seznamů se jedná o cyklicky propojený datový typ. Obvykle jsou seznamy používané k řešení operací kombinatoriky, symbolické manipulaci s výrazy, algoritmů vnějšího třídění, uspořádání datové struktury.

Součástí této bakalářské práce je proto i informování čtenáře o základním použití seznamů v praxi.

2 Obvyklé použití seznamů

Seznamy je možné běžně používat různým způsobem. Obvyklé řešení nad dynamicky vytvořeným seznamem jsou operace kombinatoriky (permutace seznamu, generování variací a kombinací prvků ze seznamu), symbolická manipulace s výrazy (práce s mnohočleny, symbolická derivace), algoritmy vnějšího třídění (merge-sort na seznamech).

Datové struktury seznamu používá i Linux. V řadě příležitostí používá propojené nebo zřetězené datové struktury. Pokud každou datovou strukturu chápeme jako jednu instanci nebo výskyt něčeho, např. procesu nebo síťového zařízení, musí být jádro schopno nalézt všechny instance. V lineárním seznamu obsahuje kořenový ukazatel adresu první datové struktury (prvku), v seznamu a každá struktura obsahuje ukazatel na následující prvek seznamu. Ukazatel posledního prvku má hodnotu 0 nebo NULL, čímž se signalizuje konec seznamu. V obousměrně propojeném seznamu obsahuje každý prvek ukazatel jednak na následující prvek a jednak také ukazatel na předchozí prvek seznamu. Pomocí obousměrně propojených seznamů se usnadňuje přidávání a odstraňování prvků uprostřed seznamu, je k tomu ale zapotřebí více paměti. To je typické dilema každého operačního systému: rozhodování mezi zatížením paměti a procesoru.

Seznamy jsou užitečným nástrojem pro organizaci datové struktury, průchod takovýmto seznamem ale může být neefektivní. Pokud je třeba vyhledat určitý prvek, může se snadno stát, že bude nutné procházet celým seznamem. K obejití tohoto omezení je možné použít hashování, které používá i Linux (resp. používá hashovací tabulku). Hashovací tabulka je pole ukazatelů na datové struktury, přičemž její index se odvozuje od informací v těchto strukturách.

Pokud má datová struktura obsahující např. informace o obyvatelích nějaké vesnice, můžete jako index použít věk obyvatel. Při hledání dat o určité osobě použijete její věk jako index do hashovací tabulky obyvatel a pak už budete pouze

sledovat ukazatel na datovou strukturu obsahující informace o určité osobě. Bohužel je velmi pravděpodobné, že určitý věk bude mít více obyvatel v obci, takže ukazatel v hashovací tabulce je ukazatelem na řetězec či seznam datových struktur, které popisují obyvatele stejného věku. Nicméně průchod těmito kratšími seznamy je stále rychlejší než prohledávání všech datových struktur.

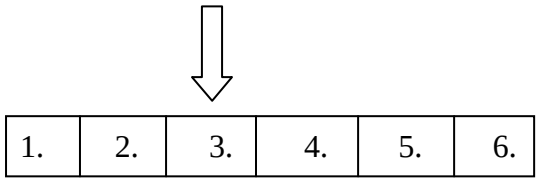
Hashovací tabulky urychlují přístup k často používaným datovým strukturám, Linux používá hashovací tabulky velmi často při implementaci vyrovnávacích pamětí. Datové struktury se ukládají a udržují ve vyrovnávacích pamětech, protože k nim jádro přistupuje velmi často.

Seznamy je možné také používat pro práci s zásobníky a frontami. Pomocí seznamu je možné obejít softwarové omezení pro práci s dlouhými přirozenými čísly. Tato dlouhá čísla můžeme rozdělit do seznamu po menších částech. Jedním ze způsobů, jak toto provést, je rozložení do každého prvku seznamu vložit samostatnou hodnotu každé pozice čísla. V tom případě je lepší, když číslo „překlopíme“, tedy na první pozici bude číslice s nejnižší váhou (mocninou deseti). Pozice (index) číslice pak odpovídá váze číslice.

3 Seznamy a operace s nimi

Seznam je datová struktura. Datová struktura seznamu je dána druhem dat, který je v rámci seznamu povolena a typickými operacemi, které je možné s daty provádět. Druhem dat máme na mysli množinu hodnot, kterých může datový typ (datové typy) v seznamu nabývat. Za datový typ můžeme prohlásit třídu datových struktur, které mají stejné druhy dat, operace a vlastnosti, tzn. datové struktury v rámci datového typu je možné vzájemně zaměnit z funkčního hlediska (mohou se lišit časovou a prostorovou náročností implementace).

Seznamem tedy označujeme posloupnost prvků dané datové struktury, na kterou ukazuje vnitřní pointer a tedy v závislosti na poloze pointeru můžeme vkládat nové údaje v libovolném místě, případně libovolný prvek ze seznamu můžeme odebrat.



Obr. 1

Při vkládání prvků je věcí dohody, zda se pole vkládá před ukazatele vnitřního pointeru, nebo za něj. Pomocí tohoto ukazatele určujeme na jaké místo bude prvek přidáván nebo z jakého místa bude prvek odebrán, kde budeme provádět aktualizaci vložených údajů nebo jen číst hodnoty.

Ukazatel na seznam je možné realizovat několika způsoby. Resp. v tomto ukazateli je možné uchovávat různé hodnoty, které se vztahují k seznamu na který ukazuje. Nejčastější případy jsou:

- Minimální varianta je **pouze ukazatel na aktuální polohu** v seznamu. Tento typ ukazatele nelze použít pro jednosměrný nekruhový seznam, jelikož v seznamu by pak nebyla možnost vrátit se na začátek.
- Ukazatel na seznam obsahuje **ukazatel na aktuální polohu v seznamu a počet prvků**, který je v seznamu. I pro tento ukazatel platí stejné omezení

jako pro předcházející typ ukazatel.

- Ukazatel na seznam obsahuje ukazatel na **aktuální polohu a na první prvek v seznamu**. Toto je minimální typ ukazatele na seznam pro jednosměrný nekruhový seznam.
- Ukazatel na seznam obsahuje ukazatel na **aktuální polohu a na první prvek v seznamu a počet prvků**, které jsou v seznamu.

Obecně je vhodné do ukazatele na seznam vytáhnout často využívané hodnoty (např. počet záznamů v seznamu, maximální nebo minimální hodnoty) nebo informace, které mohou zrychlit vyhledávání v seznamu (např. ukazatel na první nebo poslední prvek seznamu).

U kruhových seznamů si obvykle vystačíme s ukazatelem, který obsahuje ukazatel na seznam a počet prvků v kruhovém seznamu. Zde je možné pro zjednodušení operací prováděných se seznamem vložit do ukazatele i odkaz na první vložený prvek do seznamu. Toto můžeme označit za první prvek, i když v kruhu je to zcela relativní pojem.

Seznam je datový typ parametrizovatelný typem údajů, které se do seznamu vkládají. Základní operace, které jsou pro datový typ seznam charakteristické lze shrnou do těchto bodů:

- **init** – vytvoření nového seznamu
- **insert** – vloží nový prvek do seznamu před/ za polohou ukazatele seznamu dle výše uvedené dohody
- **delete** – zruší prvek v seznamu na který ukazuje ukazatel
- **first** – nastaví ukazatel na první záznam v seznamu
- **last** – nastaví ukazatel na poslední záznam v seznamu
- **next** – posunutí ukazatele na následující záznam v seznamu
- **prev** – posunutí ukazatele na předchozí záznam v seznamu
- **read** – čtení dat z aktuálního prvku na jehož polohu v seznamu ukazuje ukazatel
- **length** – délku seznamu

- **empty** – test na prázdnost seznamu
- **full** – test na plnost seznamu
- **atbeg** – test toho, že ukazatel ukazuje na začátek seznamu
- **atend** – test toho, že ukazatel ukazuje na konec seznamu

Tento seznam základních operací se může lišit dle typu seznamu. Např. u kruhového seznamu odpadají operace na kontrolu, zda ukazatel ukazuje na začátek či konec seznamu, jelikož v kruhu není začátek ani konec. Tyto operace, pokud jsou vyžadovány u kruhového seznamu, se nevztahují k začátku a konci, ale k definovanému významnému bodu seznamu. Za významný bod můžeme prohlásit např. první vložený prvek takovéhoto seznamu.

Seznam je dynamická datová struktura, která kromě datové části obsahuje také část vazební. Vazební část seznamu zajišťuje propojení jednotlivých prvků seznamu mezi sebou, což umožňuje procházení seznamem.

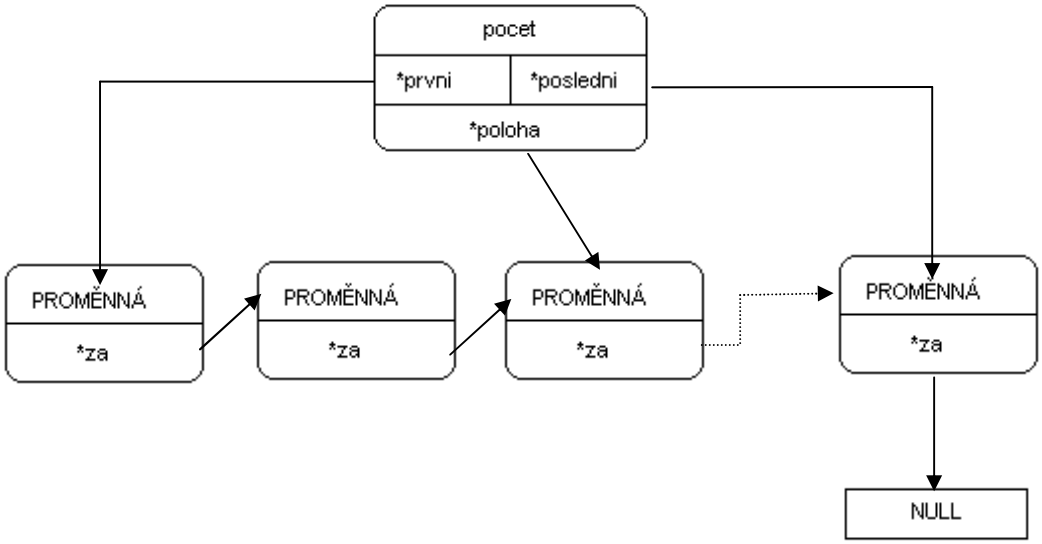
Seznam můžeme realizovat několika způsoby v závislosti na způsobu řetězení jednotlivých prvků seznamu. A to na:

- Jednosměrný seznam
- Jednosměrný kruhový seznam
- Obousměrný seznam
- Obousměrný kruhový seznam

3.1 Jednosměrný seznam

Vazební část seznamu se skládá pouze z jednoho ukazatele, který ukazuje na následující prvek seznamu, případně obsahuje NULL v případě, že se jedná o poslední prvek v seznamu. Nejmenší množství statických dat, které potřebujeme pro správu jednosměrného seznamu, je ukazatel na první prvek. A podpůrné funkce.

V takovémto seznamu můžeme postupně projít od začátku seznamu až na konec. Zpětný průchod seznamem není možný.



Obr 2. Jednosměrný seznam

definice struktury v programu:

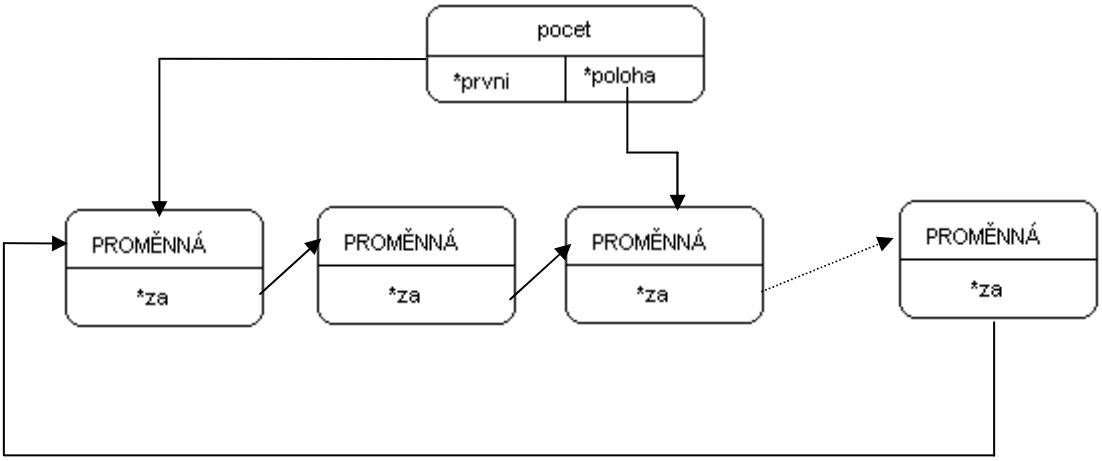
```
typedef struct item {
    PROMENNA hodn;
    struct item *za;
    } POLOZKA;
```

Obrázek 2 odpovídá datové struktuře realizace jednosměrného seznamu. Do položky PROMĚNNÁ se vkládají hodnoty jednotlivých prvků seznamu. Do pointeru *za je ukládána hodnota na následující prvek seznamu. Pomocí tohoto ukazatele je seznam zřetězen a je možné pomocí něj seznamem procházet. Všechny prvky v seznamu jsou dynamické. Poslední prvek seznamu má v ukazateli na následující prvek hodnotu NULL.

3.2 Jednosměrný kruhový seznam

Vazební část seznamu, stejně jako v předchozím případě, se skládá z jednoho ukazatele jenž ukazuje na následující prvek seznamu. V seznamu neexistuje prvek bez odkazu na následující. Jednosměrný kruhový seznam je uzavřený jednosměrný seznam, kdy poslední vložený prvek ukazuje na první vložený a tím se uzavírá v kruh.

Toto kruhové propojení částečně řeší nemožnost zpětného průchodu, když seznam je možné procházet stále dokola. Toto řešení není z funkčního hlediska optimální. Ukazatel *první neukazuje na první prvek v seznamu, ale na první vložený prvek. Jednosměrný kruhový seznam je typický příklad optimalizace minimalizace datové velikosti programu a potenciální rychlosti provádění operací.



Obr 3. Jednosměrný kruhový seznam

definice struktury v programu:

```
typedef struct item {
    PROMENNA hodn;
    struct item *za;
    } POLOZKA;
```

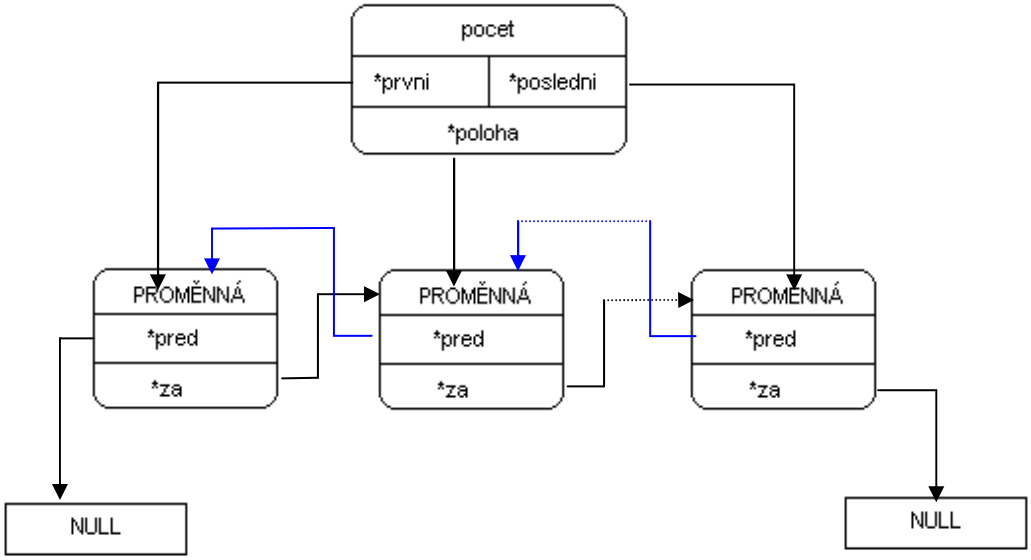
Obrázek 3 odpovídá datové struktuře vytvořené v programu, tato datová struktura je shodná s datovou strukturou jako v předchozím případě. Do položky PROMĚNNÁ se vkládají hodnoty jednotlivých prvků seznamu. Do pointeru *za je ukládána hodnota na následující prvek seznamu. Pomocí tohoto ukazatele je seznam zřetězen a je možné pomocí něj seznamem procházet. Všechny prvky v seznamu jsou dynamické. Poslední prvek seznamu, na rozdíl od jednosměrného seznamu, neobsahuje prázdnou hodnotu, ale ukazuje na první vložený prvek seznamu. Tím je seznam propojen do nekonečné smyčky (kruhu) a je možné jím cyklicky procházet.

Obecně nemají u kruhového seznamu smysl operace, které se vztahují k počátku či konci, protože kruh obecně nemá začátek ani konec. Z praktického hlediska však lze na kruhu vyznačit významný bod (v kruhovém seznamu významný prvek), ke kterému se vztahují operace závislé na začátek a konec.

3.3 Obousměrný seznam

Vazební část seznamu ukazuje na dva členy, jeden je následující prvek v seznamu a druhý je předcházející prvek seznamu. První prvek obsahuje v odkazu na předcházející prvek hodnotu NULL. V případě, že se jedná o poslední prvek v seznamu je hodnota NULL v ukazateli na následující prvek. Hodnota NULL odpovídá prázdné hodnotě ukazatele, jelikož v tomto směru již seznam nepokračuje. Podle těchto hodnot (NULL) je možné se v seznamu orientovat a při dosažení položky v seznamu, která má v některém z ukazatelů prázdnou hodnotu znamená to, že prvek je první, poslední nebo v seznamu došlo k chybě, resp. k přerušení seznamu. Nejmenší množství statických dat, které potřebujeme pro správu obousměrného seznamu, je ukazatel na první prvek. A podpůrné funkce.

V takovémto seznamu můžeme postupně procházet data v libovolném směru.



Obr 4. Obousměrný seznam

definice struktury v programu:

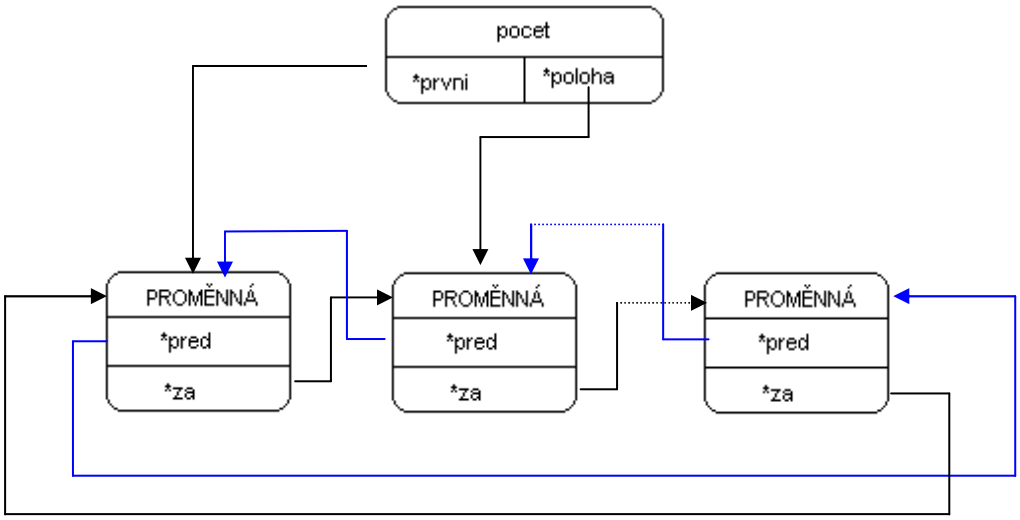
```
typedef struct item {
    PROMENNA hodn;
    struct item *za, *pred;
    } POLOZKA;
```

Obrázek 4 odpovídá datové struktuře obousměrného seznamu vytvořené v programu. Do položky PROMĚNNÁ se vkládají hodnoty jednotlivých prvků seznamu. Do pointeru *za je ukládána adresa na následující prvek seznamu, do pointeru *před je ukládána adresa na předchozí prvek seznamu. Pomocí těchto ukazatelů je seznam obousměrně zřetězen a je možné pomocí něj seznamem procházet. Všechny prvky v seznamu jsou dynamické. Poslední prvek seznamu má v ukazateli na následující prvek hodnotu NULL. První prvek má v ukazateli na předchozí prvek také hodnotu NULL.

3.4 Obousměrný kruhový seznam

Vazební část seznamu ukazuje na dva členy, jeden na následující a jeden předcházející prvek seznamu, první prvek obsahuje odkaz poslední prvek v seznamu s referencí jako „předchozí“ a obdobně poslední prvek seznamu ukazuje na první s referencí „následující“. Nejmenší množství statických dat, které potřebujeme pro správu obousměrného seznamu, je ukazatel na první prvek. A podpůrné funkce.

V takovémto seznamu můžeme postupně procházet data v libovolném směru. A procházení dat můžeme libovolně cyklicky opakovat.



Obr 5. Obousměrně propojený kruhový seznam

```
typedef struct item {
    PROMENNA hodn;
    struct item *za, *pred;
    } POLOZKA;
```

Obrázek 5 odpovídá datové struktuře obousměrného seznamu, jako v předchozím případě, vytvořené v programu. Do položky PROMĚNNÁ se vkládají hodnoty jednotlivých prvků seznamu. Do pointeru *za je ukládána adresa na následující prvek

seznamu, do pointeru *před je ukládána adresa na předchozí prvek seznamu. Pomocí těchto ukazatelů je seznam obousměrně zřetězen a je možné pomocí něj seznamem procházet. Všechny prvky v seznamu jsou dynamické. Rozdíl oproti předchozímu případu je v zřetězení posledního a prvního prvku, kdy ukazatel *za posledního prvku seznamu ukazuje na první prvek a obdobně ukazatel *pred prvního prvku ukazuje na poslední prvek seznamu. Tím je kruhově seznam propojen a je možné jím periodicky procházet oběma směry.

Způsob práce při operacích, které se vztahují k začátku a konci seznamu, je stejný jako u práce s jednosměrným kruhovým seznamem, viz. výše.

4 Operace se seznamem

4.1 Obecná pravidla pro tvorbu seznamů

Při tvorbě seznamu lze využít buď pole nebo dynamické spojové struktury. Pro lepší vyhledávání dat v seznamu je vhodnější udržovat seznam setříděný. Při implementaci seznamu pomocí pole je možné jej reprezentovat pomocí:

- pole, obsahující vlastní prvky
- index na aktuální prvek seznamu (ukazatel)
- počet obsazených prvků v seznamu (délka seznamu)

tedy:

```
int pole [MAX];      /* pole prvků seznamu */
int ukaz;            /* ukazatel na prvek v seznamu */
int len;             /* délka seznamu */
```

Je-li předpoklad, že počty prvků v seznamu budou vysoké, nebo není jednoznačně definován počet prvků je pro implementaci vhodné použít spojové struktury. Ke každému prvku seznamu přidáme ukazatel na následující prvek seznamu. Pro snadnější práci se seznamem je v ukazateli uložena i délka seznamu a odkazy na první a poslední prvek. Při implementaci pomocí spojových struktur je možné ji reprezentovat pomocí:

- struktury obsahující vlastní prvky a ukazatele na okolní prvek/ prvky
- ukazatele na prvek v seznamu (realizovaný pomocí pointeru)
- počet prvků v seznamu (délka seznamu)

tedy:

```
            /* definice struktury jednosměrného seznamu */
typedef struct item {
    PROMENNA hodn;                  /* hodnota vložené proměnné */
    struct item *za;                /* ukazatel na následující prvek */
    } POLOZKA;

            /* definice struktury ukazatele na seznam s ukazateli na první,
poslední prvek seznamu, na aktuální polohu a obsahuje počet prvků v seznamu pro
snadnější práci se seznamem */

typedef struct {
    long pocet;                             /* počet prvků v seznamu */
    POLOZKA *prvni, *poloha, *posledni;     /* ukazatele na významné */
                                            /* body v seznamu */
    } UKAZATEL;
```

Seznamy jsou pro větší objemy dat neefektivní. Složitost mazání i vyhledávání je v průměrném i nejhorším případě lineární, při n záznamech v seznamu musíme projít n/2 v průměrném a n záznamů v nejhorším případě pro každou operaci. Ve skutečných aplikacích, které intenzivně pracují s větším množstvím dat, se proto používají jiné datové struktury.

Jednou z možností jsou binární stromy, kde má každý prvek hned dva následníky (místo jednoho ukazatele “další” v případě seznamu), levý a pravý, hlavě se v případě stromů říká kořen. Klíč v pravém následníku je vždy větší než v rodičovském záznamu. Vyhledávání, přidávání a mazání má potom jen logaritmickou časovou složitost, ovšem algoritmy jednotlivých operací jsou složitější než v případě seznamu.

Jednosměrné seznamy (protože mají jen jeden pointer) jsou úspornější co do obsazení paměti než obousměrné seznamy. Jejich použitím si však uzavíráme

možnost procházení seznamu ve zpětném směru. Jde opět o klasický kompromis mezi výkonností a nároky na paměť (hardware).

4.2 Operace se seznamem

Následující kapitoly obsahují některé vybrané operace pro práci s jednosměrným kruhovým seznamem.

Vycházíme z předpokladu definic. Vzhledem k tomu, že v příkladu nejsou uvedeny všechny operace, resp. některé by byly složitější, uvádím zde jak obecné, tak (pokud použito v příkladu) konkrétní použití základních operací s kruhovým seznamem.

Definice obecné položky kruhového seznamu

```
typedef struct polozka {
        TYPPRVKU hodnota;
        struct polozka *next;
    } POLOZKA;
```

Definice obecného ukazatele na položky kruhového seznamu

```
typedef struct {
    POLOZKA *prvni, *ukazatel;
    } UKAZ;
```

Definice položky kruhového seznamu z příkladu (souřadnic)

```
typedef struct item {
      PROMENNA x,y;
      struct item *dalsi;
      } POLOZKA;
```

Definice ukazatele na kruhový seznam z příkladu (záhlaví regionu)

```
typedef struct aaa {
    float minx, miny, maxx, maxy;
    int poradi;
    POLOZKA *reg;
    struct aaa *pred, *za;
    } SEZN;
```

V záhlaví regionu jsou uchovávané proměnné, které uchovávají hodnoty :

- maximální souřadnici x a y
- minimální souřadnici x a y
- ukazatel na předchozí a následující region
- ukazatel na kruhový seznam obsahující souřadnice

4.3 Init

Vytvoření nového záhlaví seznamu lze realizovat pomocí níže popsané funkce, kdy vytvoříme ukazatel, který neukazuje na žádný seznam souřadnic (neboť neexistuje = je prázdný) a do pořadí je vloženo číslo, které určuje pořadí vznikajícího regionu. Tato hodnota je předávána při volání funkce. V případě realizace příkladu je inicializace jednosměrného kruhového seznamu (obsahující souřadnice) totožná s vytvořením nového prvku v kruhovém seznamu.

```
SEZN *init(int REGION) {
   SEZN *pps;

    pps = (SEZN *) malloc (sizeof(SEZN));    // alokace mista

    pps->poradi = REGION;
    pps->maxx = 0; pps->maxy = 0;
    pps->minx = 0; pps->miny = 0;
    pps->reg = NULL;
```

```
    return(pps);
}
```

Obecná definice vypadá takto:

```
UKAZ *init (void) {
    UKAZ s=(UKAZ *) malloc (sizeof (UKAZ));
    s->prvni=NULL;
    s->ukazatel=NULL;
    return(s)
    }
```

4.4 Insert

Funkce insert vloží nový prvek do seznamu před/ za polohou ukazatele seznamu dle požadované funkce programu nebo dle volby programátora. Při vkládání prvku je nutné ošetřit stav, kdy je vkládán první prvek. Při vložení prvního prvku jsou nastaveny ukazatele tak, že prvek ukazuje sám na sebe, čímž je vytvořen kruh, přestože celý seznam obsahuje jen jeden prvek. Pojem kruhový seznam lépe vystihuje pojem cyklický seznam.

V případě, že vkládáme prvek (v uvedeném příkladě souřadnice) za polohu ukazatele řešíme vložení takto:

```
void insert_behind(float sx, float sy){
  POLOZKA *psr;
if (!hlv->reg) {
            plz = (POLOZKA *) malloc (sizeof(POLOZKA));
            hlv->reg = plz;
            plz->x = sx;
            plz->y = sy;
            plz->dalsi = plz;
           }
```

```
            else {
             psr = (POLOZKA *) malloc (sizeof(POLOZKA));
             psr->x = sx;
             psr->y = sy;
             psr->dalsi = plz->dalsi;
             plz->dalsi = psr;
             plz = psr;
               }
}
```

V případě obecného řešení, by podle uvedené definice prvků funkce vypadaly takto:

V případě, že vkládáme prvek před polohu ukazatele:

```
void insertbefore (UKAZ *s, TYPPRVKU x) {
    POLOZKA *b;
        /* vytvoření nového prvku v seznamu*/
    POLOZKA *c = (POLOZKA *) malloc (sizeof(POLOZKA));
        /* vložení nového prvku seznamu na konec */
        c->hodnota = x;
    if (!s->prvni) {
                        /*ukazatel nikam neukazuje = v sezanamu není žádný prvek */
        c->next = c;
        s->prvni = c;
        s->ukazatel = c;
    }
    else {                  /* nový prvek je vkládán do neprázdného seznamu */
        b = s->prvni;
        if (b->next == b) {
                /* jedná se o seznam jen s jedním prvkem */
                        c->next = b;
                        b->next = c;
                }
```

```
                else
        while (b->next != s->ukazatel) b = b->next; /* dojdi na předchozí prvek */
                c->next = s->ukazatel->next;
                b->next = c;
    }
}
```

Pokud chceme vkládat prvek za polohu ukazatele:

```
void insertbehind (UKAZ *s, TYPPRVKU x) {
    POLOZKA *b;
        /* vytvoření nového prvku v seznamu*/
    POLOZKA *c = (POLOZKA *) malloc (sizeof(POLOZKA));
        /* vložení nového prvku seznamu na konec */
        c->hodnota = x;
    if (!s->prvni) {
                        /*ukazatel nikam neukazuje = v sezanamu není žádný prvek */
        c->next = c;
        s->prvni = c;
        s->ukazatel = c;
    }
    else {              /* nový prvek je vkládán do neprázdného seznamu */
        b = s->prvni;
        if (b->next == b) {
                /* jedná se o seznam jen s jedním prvkem */
                        b->next = c;
                        c->next = b;
                }
                else
        b = s->ukazatel;
        c-next = b->next;
        b->next = c;
```

```
    }
}
```

4.5 Delete

Zde popisuji pouze obecné řešení, nicméně v této obecné rovině není v příkladu použito. V příkladu nedochází k mazání jednotlivých prvků v seznamu, je v něm řešeno až uvolnění paměti na konci běhu programu.

Funkce delete zruší prvek v seznamu, na který ukazuje ukazatel. Je nutné provést kontrolu, zda seznam není prázdný a tedy je co mazat. A důležitá kontrola je i na mazání posledního prvku v seznamu.

```
void delete (UKAZ *s) {
    POLOZKA *a, *b;
    if (empty(s)) printf(„Ze seznamu není co smazat, je prazdny!\n“);
        else {
                if (length(s) == 1) {
    /* jedná se o poslední prvek v kruhovém seznamu, stačí smazat jen tento prvek */
                        a = s->ukazatel;
                        free(a);
                                }
                                else {
    /* v seznamu je více prvků je třeba zachovat propojení */
                        a = s->ukazatel;
                        prev(s);
                        b= s->ukazatel;
                        b->next = a->next;
                        free(a);
                                }
        }
    }
```

4.6 First

Nastaví ukazatel na první vložený záznam v seznamu. Zde je uveden jen obecné řešení. V řešeném příkladu nedochází ke změně ukazatele v záhlaví souřadnic (položce definující region). Pokud by bylo nutné v programu získat první vloženou souřadnici regionu, funkce by vracela hodnotu obsaženou v proměnné hlv->reg (kde je adresa ukazující na první vloženou souřadnici).

```
void first (UKAZ *s) {
   s->ukazatel = s->prvni;
   }
```

4.7 Last

V kruhovém seznamu neexistuje poslední prvek. Tato funkce nastaví ukazatel na prvek před první vložený prvek.

```
void last (UKAZ *s) {
   POLOZKA *a;
   a = s->prvni;
   while (a->next != s->ukazatel) a = a->next;
   s->ukazatel = a;
   }
```

4.8 Next

Posunutí ukazatele na následující záznam v seznamu. V příkladu jsou dva seznamy, pro každý tedy musí být samostatná funkce posunu na následující prvek. Jedná se o dvě funkce, 1. posouvá na následující prvek hlavního seznamu (návěští souřadnic regionu), 2. posouvá ukazatel na následující prvek v seznamu souřadnic.

```
void next_hlv(){
  hlv = hlv->za;
}

void next_plz(){
  plz = plz->dalsi;
}
```

V obecném řešení lze provést např. takto:

```
void next (UKAZ *s) {
    POLOZKA *a;
    a = s->ukazatel;
    s->ukazatel = a->next;
    }
```

4.9 Prev

Funkce prev zajišťuje posunutí ukazatele na předchozí záznam v seznamu. Obecné řešení je možné realizovat tímto způsobem:

```
void prev (UKAZ *s) {
    POLOZKA *a;
    a = s->ukazatel;
    while (a->next != s->ukazatel) a = a->next;
    s->ukazatel = a;
    }
```

4.10 Read

Funkce read zajišťuje čtení dat z aktuálního prvku, na jehož polohu v seznamu ukazuje ukazatel. Funkce načte prvek na adrese, na kterou ukazuje záhlaví a vrátí hodnotu v tomto prvku. Ukázka řešení je opět obecná z důvodu přehlednosti. V pří-

kladu je složitější datová struktura položek v kruhovém seznamu.

```
TYPPRVKU read (UKAZ *s){
    POLOZKA *a;
    a = s->ukazatel;
    return (a->hodnota);
}
```

4.11 Length

Pomocí této operace získáme délku seznamu. Je vhodné délku seznamu uchovávat v ukazateli na seznam, abychom nemuseli procházet jednotlivé prvky a ty počítat. Pokud tuto hodnotu máme v ukazateli uloženou, pak je funkce triviální. Pokud v ukazateli nemáme počet prvků, pak jimi musíme projít a spočítat je. Na začátku je nutná kontrola na prázdnost seznamu.

```
int length (UKAZ *s) {
    POLOZKA *a;
    int i = 0;
    if (s->ukazatel == NULL) return(0);
        else{
                a = s->ukazatel;
                do {a = a->next;i++;}
                while (a != s->ukazatel);
                return (i);
        }
    }
```

5 Popis řešení programu – práce s regiony

5.1 Základní předpoklady programu

Předkládaný program je vytvořen jako ukázka pro práci s různými dynamickými seznamy. V programu jsou použity dva. Jeden obousměrný kruhový seznam a jeden jednosměrný kruhový seznam.

Předpokladem pro korektní běh programu je, že data obsažená v datovém zdrojovém souboru jsou korektní a nepotřebují kontrolu. Kontrola vstupních dat není programem prováděna.

5.2 Popis načítání dat do seznamu

Tento program načítá data ze souboru, který je uložen ve formátu csv ve stejném adresáři jako je spouštěný program. V tomto csv souboru jsou informace o souřadnicích, které definují rozměr regionu na mapě. Jedná se dvourozměrný souřadnicový systém, kdy každá souřadnice může být zadána jako reálné číslo. Nový region ve zdrojovém csv souboru je označen slovem „Region“.

V programu jsou vytvořeny dva seznamy. Obousměrný kruhový seznam funguje jako záhlaví na jednosměrný kruhový seznam, ve kterém jsou uloženy souřadnice. V obousměrném kruhovém seznamu jsou postupně načteny hodnoty:

- pořadí regionu
- ukazatel na souřadnice regionu
- maximální hodnoty souřadnic
- minimální hodnoty souřadnic regionu

Program nejprve otevře zdrojový csv soubor a řádek po řádku načítá hodnoty do seznamů. Řádek ve zdrojovém souboru může obsahovat:

- označení nového regionu
- souřadnice regionu

Pro načtení dat ze souboru je použita funkce :

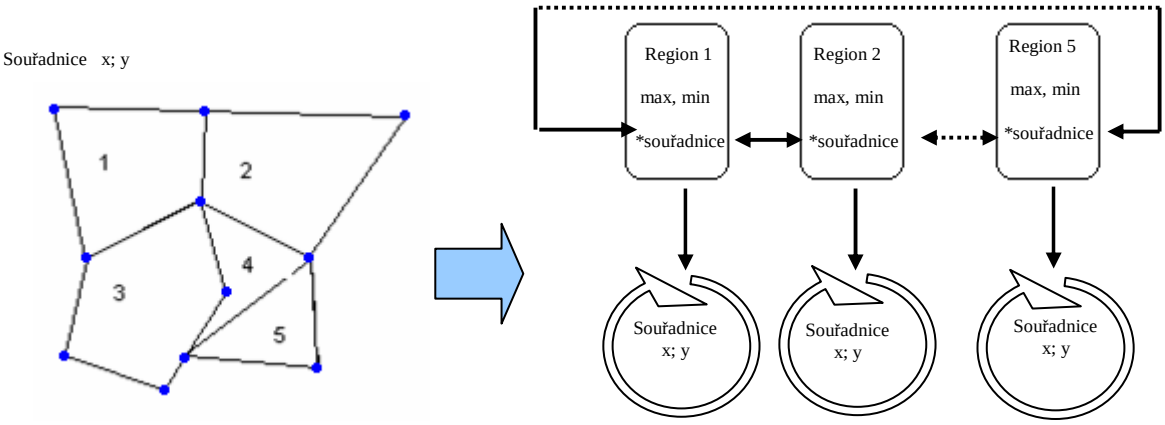
```
void nacti()
{
   FILE *f;
   char c[40];          // radka v souboru

   f = fopen ("zdroj.csv", "r");       // otevreni souboru pro cteni
   f == NULL ? printf("soubor se nepodarilo otevrit\n\n"):printf("soubor se podarilo otevrit\n\n");

   if (f) {                  // byl soubor otevren?
        while ((fgets(c, 39, f)) != NULL){
            if (strstr(c, "Region")) {
                    // novy region - vytvor novy clanek v seznamu regionu
                    REGION++;
                    pridej_region(REGION);
                            }
                    else {
                    // jedna se o souradnice, je treba je ziskat a prevest na float
                    // a pripojit k poslednimu zalozenemu regionu.
                    pridej_souradnice(c);
                        }
            }
        fclose(f);            // na konci prace uzavri soubor
        }
}
```

V této funkci jsou postupně načítány jednotlivé řádky. Následně je vyhodnocováno co řádek obsahuje. Pokud řádek obsahuje text „Region“, pak je založeno nové záhlaví pro souřadnice regionu připojením dalšího prvku do obousměrně propojeného seznamu. Vytvoření nového prvku je provedeno funkcí *pridej_region(REGION)*, kde hodnota REGION představuje pořadové číslo regionu, tak jak je načítán ze zdrojového souboru. Přidání souřadnic je řešeno funkcí *pridej_souradnice(c)*, kde hodnota c obsahuje řetězec se souřadnicemi, které jsou v uvedené funkci převedeny na typ float a vloženy do kruhového seznamu.

Ve funkci pro přidávání regionu je důležitá kontrola, zda již existuje seznam záhlaví souřadnic nebo ne. V případě, že seznam neexistuje, je vytvořen první prvek a navázán sám na sebe. Pokud již seznam existuje, je nový prvek přidán za aktuální polohu prvku. Do prvku seznamu je vloženo pořadí vytvořeného regionu, které je předáno při volání funkce.

Funkci přidávající souřadnice jsou postupně předávány řetězce, načtené ze souboru, a ty jsou pak převedeny na souřadnice x a y, které reprezentují hraniční body regionů. Je prováděna kontrola, zda se jedná o první souřadnici v novém regionu, nebo je to další souřadnice k již existujícím. Pokud se jedná o první souřadnice je třeba založit jednosměrný kruhový seznam a propojit se záhlavím. Souřadnice získané ze souboru jsou vloženy do jednosměrného kruhového seznamu. Pokud se jedná o další souřadnice mezi již existující, jsou tyto nová souřadnice vloženy za aktuální polohu ukazatele.

Souřadnice regionů zapsané ve zdrojovém souboru jsou převedeny do datové struktury dvou kruhových seznamů, jak je naznačeno na obrázku č.6. Kdy seznam regionů s maximálními a minimálními souřadnicemi reprezentuje záhlaví jednosměrného kruhového seznamu ve kterém jsou uloženy souřadnice definovaných hraničních bodů jednotlivých regionů.



Obr 6. Převedení souřadnic

5.3 Získání hraničních souřadnic

Po načtení posledního řádku zdrojových dat jsou do obousměrného kruhového seznamu načteny maximální souřadnice, pomocí nichž následně vyhodnocuji, zda regiony navzájem mohou spolu sousedit. Pokud se čtverce porovnávaných regionů alespoň částečně překrývají, pak platí, že regiony mohou navzájem sousedit. Získání souřadnic je řešeno pomocí funkce *max_sour()*.

```
void max_sour()
{
  for (i=1; i<=REGION; i++) {
      while (hlv->poradi != i) {next_hlv();}
      plz = hlv->reg;
         hlv->maxx = plz->x;       // inicializace hranicnich souradnic
         hlv->minx = plz->x;
         hlv->maxy = plz->y;
         hlv->miny = plz->y;
      do {
         (plz->x > hlv->maxx)? hlv->maxx = plz->x:hlv->maxx;
         (plz->x < hlv->minx)? hlv->minx = plz->x:hlv->minx;
```

```
            (plz->y > hlv->maxy)? hlv->maxy = plz->y:hlv->maxy;
            (plz->y < hlv->miny)? hlv->miny = plz->y:hlv->miny;
            next_plz();
        } while ( plz != hlv->reg);
        hlv = hlv->za;
    }
}
```

Ve funkci jsou postupně procházeny všechny regiony i jejich souřadnice. První souřadnice v regionu jsou vloženy do záhlaví a ty jsou pak přepsány většími v případě maximálních hodnot nebo menšími v případě minimálních hodnot.

5.4 Potenciálně sousedící regiony

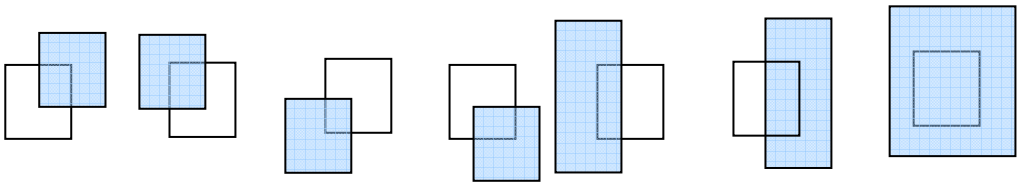
Po načtení všech dat ze souboru a vložení maximálních a minimálních souřadnic do záhlaví je možné provádět vyhodnocení, zda jednotlivé regiony spolu mohou sousedit. Vzhledem ke zjednodušené metodice vyhodnocování , pomocí obvodových obdélníků – minimum bounding rectangle, považujeme regiony za potenciálně sousedící v případě, že se čtverce dané maximálními a minimálními souřadnicemi alespoň částečně překrývají.

```
void sousedici() {
  // funkce vypise ktere regiony spolu potenciálně sousedi
  for (i=1; i<=REGION; i++) {
     printf("\nRegion c. %i potenciálně sousedi s regiony : ", i);
     sousedi(i);
  }
  printf("\n\n");}
```

Funkce *sousedici()* se rozpadá na drobnější části. Postupně jsou procházeny jednotlivé regiony, resp. jejich maximální souřadnice a ty jsou pak porovnávány s hraničními souřadnicemi ostatních regionů. Z funkce sousedici() je volána funkce

sousedi(), kde jsou oba porovnávané regiony načteny do pomocných proměnných a následně porovnány. V případě že regiony spolu mohou sousedit je toto vypsáno na obrazovku.

Minimální čtverce (regionů) se mohou překrývat různými způsoby. Kromě základních způsobů sousedství – jak je naznačeno na obrázku č. 7, jsou vzájemně sousedící regiony také, v případě, pokud mají alespoň jeden společný bod. Regiony mohou být také vzájemně vnořené, pak jsou také sousedící.



Obr 7. některé případy sousedících regionů

5.5 Konec programu – uvolnění paměti

Po vypsání všech potenciálně sousedících regionů je program ukončen. Před jeho úplným ukončením je potřeba uvolnit použitou paměť. K tomu slouží funkce *uvolni(hlv)*, skládající se ze dvou částí :

1. volání funkce *uvolni_souradnice(plz)*, která má na starost uvolnění paměti použité pro souřadnice aktuálního regionu
2. ve druhé části je uvolněna paměť návěští regionu.

6 Závěr

Cílem bakalářské práce bylo popsat datovou strukturu seznamu a funkce v jazyce C pro vybrané operace se seznamem. Popsané způsoby jsem pak aplikoval na příkladu v programu, který názorně ukazuje práci s obousměrně propojeným seznamem a kruhovým jednosměrným seznamem. Zdrojový kód je v příloze č.1 a 2, v příloze č.3 je uveden zdrojový datový soubor.

K zápisu základních operací byl použit jazyk C. Tyto operace jsou popsány v kapitole 4. Pomocí těchto operací lze se seznamy poměrně snadně a přehledně pracovat. Následně v kapitole 5 je popsán vytvořený program na kterém je názorně předvede funkčnost operací s kruhovými seznamy. Operace se pak v drobných obměnách mohou lišit. Jednotlivé práce se seznamem lze rozšiřovat na základě definice v jakém programu a k jakému účelu má být seznam použit.

7 Použitá literatura

[1] VĚCHET, V. : Algoritmy a datové struktury. Zásobníky, fronty a seznamy. Liberec, TU 2004.

[2] HEROUT, P. : Učebnice jazyka C. České Budějovice, Kopp 1992.

[3] RICHTA, K., ŠALOUN, P. : Programovací jazyk C, ČVUT Praha 2001

[4] http://casopis.programator.cz/

[5] http://www.programujte.com/

[6] http://www.karamba.cz/

[7] http://www.root.cz

8 Přílohy

Příloha 1 – Zdrojový text souboru *Region.c*

Příloha 2 – Zdrojový text hlavičkového souboru *Region.h*

Příloha 3 – *zdroj.csv* soubor obsahující zdrojová data

Příloha 4 – CD se zdrojovými soubory

Příloha 1 – Zdrojový text souboru *Regiony.c*

```
  /*****************************************************************************/
  /*  Name:        Prace se seznamy                                            */
  /*  Author:      Martin Brejcha                                              */
  /*  Soubor:      Regiony.c                                                   */
  /*  Description:  Nacteni dat ze souboru do seznamu a prace s oblastmi       */
  /*****************************************************************************/

  /*****************************************************************/
  /*                                                               */
  /*                    DEFINICE FUNKCI SE SEZNAMY                 */
  /*                                                               */
  /*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Regiony.h"

/* Funkce pro vyhledavani pozice znaku v retezci */
/* Pokud znak neni v retezci, vraci se 0         */
int najdi_znak(char b[40],char q) {
   i=0;
while (b[i] != q) {
     i++;
     if (b[i] == '\0') {
            i=0;
            return i;
            }
     }
i++;     // zvysit vystup o jednu, protoze pole zacina od 0
return i;
}

// inicializace záhlaví souřadnic regionu
SEZN *init(int REGION) {
    SEZN *pps;

 pps = (SEZN *) malloc (sizeof(SEZN));    // alokace mista v pameti
 pps->poradi = REGION;
 pps->maxx = 0;                    // inicializace promennych
 pps->maxy = 0;
 pps->minx = 0;
 pps->miny = 0;
 pps->reg = NULL;
 return(pps);
}
```

```
void next_hlv(){
  hlv = hlv->za;
}

void next_plz(){
  plz = plz->dalsi;
}


// pridani pole do seznamu (novy region)
void pridej_region(int REGION)        // pridani noveho prvku do seznamu
{
    SEZN *ps;

if (REGION == 1) {                    // zadny region jeste neexistuje
             hlv = init(REGION);

             hlv->pred = hlv;
             hlv->za = hlv;
             hlv->reg = NULL;
            }
            else {                 // je to dalsi nacitany region ze zdrojoveho souboru
             ps = init(REGION);
             ps->pred = hlv;       // zarazeni do obousmerneho seznamu za aktualni pozici
             ps->za = hlv->za;
             hlv->za = ps;
             hlv = ps->za;
             hlv->pred = ps;
             hlv = ps;
               }
}

void insert_behind(float sx, float sy){
   POLOZKA *psr;
if (!hlv->reg) {
  // vytvor a napln parametry polozky v seznamu.
             plz = (POLOZKA *) malloc (sizeof(POLOZKA));       // alokace prvnich souradnic
             hlv->reg = plz;                                   // propojeni na hlavni seznam
             plz->x = sx;                                      // naplneni souradnicemi
             plz->y = sy;
             plz->dalsi = plz;
            }
            else {
             psr = (POLOZKA *) malloc (sizeof(POLOZKA));      // jedna se o dalsi ze souradnic
             psr->x = sx;                                      // naplneni hodnotami
             psr->y = sy;
             psr->dalsi = plz->dalsi;          // zapojeni do kruhoveho seznamu za aktualni polozku
             plz->dalsi = psr;
             plz = psr;
               }
```

```
}

// pridani souradnic regionu
void pridej_souradnice(char d[40])     // pridani novych souradnic regionu
{
    float sx=0, sy=0;               // souradnice regionu
    int bb;
    char tempo[40] = " ";

    bb = najdi_znak(d,';');          // najdi kde je strednik ve stringu d

    // pred strednikem je souradnice x (prevest na double)
    strncpy(tempo,d,bb-1);           // do promenne tempo nacist retezec (bez stredniku)
    sx = atof(tempo);                // pro korektni prevod na float musi byt desetinna carka v
souboru teckou

    // za strednikem je souradnice y (prevest na double)
    strcpy(tempo," ");               // vynulovat promennou tempo
    for (i=0; i <= strlen(d)-bb; i++) tempo[i] = d[bb+i];       // naplnit tempo druhou polovinou
cislice
    sy = atof(tempo);                // prevest string na double

    insert_behind(sx,sy);
}


/*          Funkce pro nalezeni nejmensich a nejvetsich souradnic             */
/* max. a min. souradnice nasledne zapise do zahlavi souradnic daneho regionu */

void max_sour()
{
  for (i=1; i<=REGION; i++) {
       while (hlv->poradi != i) {next_hlv();}
       plz = hlv->reg;
          hlv->maxx = plz->x;       // inicializace hranicnich souradnic
          hlv->minx = plz->x;
          hlv->maxy = plz->y;
          hlv->miny = plz->y;
       do {
          (plz->x > hlv->maxx)? hlv->maxx = plz->x:hlv->maxx;
          (plz->x < hlv->minx)? hlv->minx = plz->x:hlv->minx;
          (plz->y > hlv->maxy)? hlv->maxy = plz->y:hlv->maxy;
          (plz->y < hlv->miny)? hlv->miny = plz->y:hlv->miny;
          next_plz();
       } while ( plz != hlv->reg);
       hlv = hlv->za;
    }
}

void uvolni_souradnice(POLOZKA *psz) {       // uvolni souradnice v regionu
```

```
    POLOZKA *sz;

    // prerusit kruh a postupne rusit jednotlive prvky
    sz = psz->dalsi;
    psz->dalsi = NULL;
    while (sz->dalsi != NULL) {
         psz = sz;
         sz = sz->dalsi;
         free(psz);
        }
    free(sz);
}

/* Funkce, ktera uvolnuje vseskerou pouzitou pamet */

void uvolni(SEZN *pps)
{
    SEZN *ps;

    ps = pps;

    while (REGION) {                    // Postupne uvolnuje pamet pouzitou regiony a jejich
souradnicemi
           pps = ps->za;
           uvolni_souradnice(ps->reg);  // uvolneni pameti pouzite souradnicemi
           free(ps);
           REGION--;
           ps = pps;
           }
}

/*  Funkce vypise vsechny nactene hodnoty souradnic v seznamu a zahlavi regionu */
void vypiss(){
    for (i=1; i<=REGION; i++) {
       while (hlv->poradi != i) {hlv = hlv->za;}
       printf("%i:%i - %f - %f - %f - %f - %d\n", i, hlv->poradi, hlv->maxx, hlv->maxy, hlv-
>minx, hlv->miny, hlv->reg);
       printf("    ||  %d  <- %d   -> %d\n",hlv,hlv->pred, hlv->za);
       plz = hlv->reg;
       do {
          printf("   souradnice:    %f - %f\n", plz->x, plz->y);
          plz = plz->dalsi;
       } while ( plz != hlv->reg);
       printf("\n\n");
       hlv = hlv->za;
    }
}

void nacti()
{
```

```
   FILE *f;
   char c[40];          // definice nacitane radky v souboru

   f = fopen ("zdroj.csv", "r");       // otevreni souboru pro cteni
   f == NULL ? printf("soubor se nepodarilo otevrit\n\n"):printf("soubor se podarilo
otevrit\n\n");

   if (f) {                  // Povedlo se soubor otevrit?
        while ((fgets(c, 39, f)) != NULL){
            if (strstr(c, "Region")) {
                    // novy region - vytvor novy clanek v seznamu regionu
                    REGION++;
                    pridej_region(REGION);
                        }
                    else {
                    // jedna se o souradnice, je treba je ziskat a prevest na float
                    // a pripojit k poslednimu zalozenemu regionu.
                    pridej_souradnice(c);
                        }
            }
        fclose(f);            // Konec prace se soubore. Uzavreni souboru.
        }
}

SEZN *vrat(int cr) {
    int j;
    for (j=1; j<=cr; j++) {
            while (hlv->poradi != j) {hlv = hlv->za;}
            }
    return (hlv);
}

void sousedi(int pr) {
    int k,pm;
    SEZN *p1, *p2;
    for (k=1; k<=REGION; k++){
       pm = 0;
       if (k != pr) {
            // do pomocnych promennych nacist zahlavi regionu u ktereho zjistujeme potencialni
sousedici
            // a nasledne postupne vsechny ostatni regiony do druhe pomocne promenne
            p1 = vrat(pr); p2 = vrat(k);

            // zjisteni prekryvu ctvercu
       if (
       /* regiony potencialne sousedici "rohem" */
        ((p1->maxx >= p2->minx) && (p1->maxy >= p2->miny) && (p1->minx <= p2-
>maxx) && (p1->miny <= p2->maxy)) ||
        ((p1->maxx >= p2->minx) && (p1->miny <= p2->maxy) && (p1->minx <= p2-
>maxx) && (p1->miny >= p2->maxy)) ||
```

```
        ((p1->minx <= p2->maxx) && (p1->maxy >= p2->miny) && (p1->minx >= p2-
>maxx) && (p1->miny <= p2->maxy)) ||
        ((p1->minx <= p2->maxx) && (p1->miny <= p2->maxy) && (p1->minx >= p2-
>maxx) && (p1->miny >= p2->maxy)) ||

        /* regiony potencialne sousedici "pasem" - svisle */
        ((p1->minx <= p2->minx) && (p1->miny >= p2->miny) && (p1->maxx >= p2->minx)
&& (p1->maxy <= p2->maxy)) ||
        ((p1->minx >= p2->minx) && (p1->miny >= p2->miny) && (p1->minx <= p2->maxx)
&& (p1->maxy <= p2->maxy)) ||

        /* regiony sousedici "pasem" - vodorovne */
        ((p1->minx >= p2->minx) && (p1->miny <= p2->maxy) && (p1->maxx <= p2-
>maxx) && (p1->miny >= p2->maxy)) ||
        ((p1->minx >= p2->minx) && (p1->miny <= p2->maxy) && (p1->maxx <= p2-
>maxx) && (p1->miny >= p2->miny)) ||

        /* vnorene regiony */
        ((p1->minx >= p2->minx) && (p1->miny >= p2->miny) && (p1->maxx <= p2-
>maxx) && (p1->maxy <= p2->maxy)) ||
        ((p1->minx <= p2->minx) && (p1->miny <= p2->miny) && (p1->maxx >= p2-
>maxx) && (p1->maxy >= p2->maxy))

        ) printf(" %i,",k);

          }
    }
}

void sousedici() {
  // funkce vypise ktere regiony spolu potencialne sousedi
  for (i=1; i<=REGION; i++) {
     printf("\nRegion c. %i potencialne sousedi s regiony : ", i);
     sousedi(i);
  }
  printf("\n\n");
}
```

Příloha 2 – Zdrojový text hlavičkového souboru *Regiony.h*

```
/*************************************************************/
/*  Name:        Prace se seznamy                            */
/*  Author:      Martin Brejcha                              */
/*  Soubor:      Regiony.h                                   */
/*  Description:  Nacteni dat ze souboru do seznamu a prace s oblastmi   */
/*************************************************************/

/*************************************************************/
/*                                                           */
/* DEFINICE STRUKTUR SEZNAMU A DEKLARACE FUNKCI              */
/*                                                           */
/*************************************************************/

#ifndef JIZNACTENO              /* pomocna promenna, aby soubor */
#define JIZNACTENO              /* nebyl nacteny vickrat        */

typedef float PROMENNA;         /* definice typu PROMENNA*/

typedef struct item {           // struktura definujic souradnice regionu
    PROMENNA x,y;               // hodnoty souradnic
    struct item *dalsi;
    } POLOZKA;

typedef struct aaa {            // struktura definujici zahlavi regionu
    float minx, miny, maxx, maxy;
    int poradi;                 // poradove cislo regionu
    POLOZKA *reg;               // odkaz na souradnice regionu
    struct aaa *pred, *za;
    } SEZN;

extern int REGION;              // poradove cislo regionu
extern SEZN *hlv;               // hlavni seznam ukazujici na souradnice
extern POLOZKA *plz;            // polozky seznamu
extern int i;                   // pomocna promenna

extern SEZN *init(int REGION);  // inicializace seznamu
void insert_behind(float sx, float sy); // funkce pro vlozeni souradnic do kruhoveho seznamu
void next_hlv();                // funkce pro posun na nasledujici region
void next_plz();                // funkce pro posun na nasledujici souradnice
void pridej_region(int REGION); // pridani noveho prvku do seznamu
void nacti(void);               // funkce pro nacteni dat ze souboru
extern int najdi_znak(char b[40],char q);     // funkce pro nalezeni pozice v retezci hledaneho
znaku
void pridej_souradnice(char d[40]);           // pridani novych souradnic regionu
void max_sour(void);            // funkce pro nalezeni maximalnich a minimalnich
souradnic v regionu
void uvolni_souradnice(POLOZKA *psz);  // funkce uvolnujici pamet vyuzitou souradnicemi
```

```
void uvolni(SEZN *pps);          // funkce uvolnujici pamet vyuzitou zahlavimi regionu
void vypiss(void);               // funkce vypisujici nactena data na obrazovku
extern SEZN *vrat(int cr);       // funkce vrati ukazatel na region, jehoz poradove cilo je
predano
void sousedi(int pr);            // funkce vypisuje potencalne sousedici regiony
void sousedici(void);            // funkce zjistuje zda regiony mohou vzajemne sousedit

#endif

/*    konec definice souuboru */
```

Příloha 3 – *zdroj.csv* soubor obsahující zdrojová data

"Region";
5;0
16;10.5
29.8;15
30;0
33.4;-9.8
27;-13
12;-14
"Region";
16;10.5
8;30.8
11;47
28;48
33;28
29.8;15
"Region";
28;48
32;60
51;58
60.3;41
59.8;27
44;31.6
"Region";
33;28
44;31.6
59.8;27
62;12
60;0
52;-12.6
42;-14.2
33.4;-9.8
30;0
29.8;15
"Region";
12;-14
27;-13
33.4;-9.8
42;-14.2
52;-12.6
48;-32
43;-42
28;-43
17;-36
"Region";
52;-12.6
60;0
70.6;-12.6

```
73;-24
62;-28
48;-32
"Region";
5;56
4.7;69.4
10;79
27;84
34;78
35;69
32;60
28;48
11;47
```