Pracovní list 8: Dynamické struktury

Co už máme znát

- práce s polem záznamů;
- pojem ukazatele;
- práce s dynamickými proměnnými;
- princip fronty (FIFO) a zásobníku (LIFO).
- princip binárního stromu

Kontrolní otázky

8.1 Co je staticky alokovaná a dynamicky alokovaná paměť?

8.2 Jaké zásadní výhody má dynamicky alokovaná paměť?

8.3 Jaké nevýhody má práce s dynamicky alokovanou pamětí?

8.4 Jak se vytváří datový typ ukazatel?

8.5 Jak se přistoupí k proměnné, na niž ukazuje nějaký ukazatel?

8.6 Jaké operace se používají u fronty a jaké u zásobníku?

8.7 Co je binární strom a jak se implementuje každý jeho uzel?

Příprava na cvičení

Ve cvičení budeme potřebovat editor a překladač pro programy v jazyce C++. Dále si připravte textový soubor `seznam.txt` obsahující alespoň 15 jmen oddělených mezerou.

Řešené příklady

Příklad 8.1 Pomocí pole vytvořte abstraktní datový typ zásobník. Zásobník bude mít operace přidej, odeber, vypiš a vyprázdni. Zásobník uchovává jména osob. Do zásobníku je možno vložit maximálně 20 osob.

Řešení: Celý program je již vytvořen trochu komplexněji. Má i své menu (nabídku), aby si uživatel mohl sám vybrat, kterou operaci má program vykonat. Nabídka je realizována funkcí menu. Ta nutí uživatele vybrat jednu z pěti možností. V hlavním těle programu je menu vyvoláváno opakovaně, dokud uživatel nezadá číslo 5, čímž se běh programu ukončí. Pro každou z dalších operací je vytvořen vlastní podprogram. Pro práci se statickým zásobníkem potřebujete znát maximální počet prvků, které lze vložit do zásobníku, a vytvořit pole s těmito prvky. Prvkem může být obecně cokoliv, v tomto případě to jsou jména. Všechny podprogramy jsou velice jednoduché. Je však potřeba dávat pozor, kdy se parametry volají hodnotou a kdy odkazem. Při přidání na zásobník se vloží nový prvek na odpovídající pozici a zvýší se počitadlo určující počet prvků. Uvědomte si rozdíl mezi `pocet++` a `++pocet`. U odebrání je to stejně jednoduché. Tato funkce navíc vrací odebírané jméno. Vypisuje se i při vyprázdnění. Schválně se podívejte, v čem se liší výpis od vyprázdnění.

```
#include <iostream>
using namespace std;

typedef string typZasobnik[20];

void pridej(string jmeno, typZasobnik z, int &pocet){
  z[pocet++]=jmeno;
}

string odeber(typZasobnik z, int &pocet){
  return z[--pocet];
}

void vyprazdni(typZasobnik z, int &pocet){
  while(pocet>0)
    cout<<z[--pocet]<<" ";
  cout<<endl;
}

void vypis(typZasobnik z, int pocet){
  while(pocet>0)
    cout<<z[--pocet]<<" ";
  cout<<endl;
}

int menu(){
  int vyber;
  do {
      cout<<"1 ... přidej"<<endl
          <<"2 ... odeber"<<endl
          <<"3 ... vypiš"<<endl
          <<"4 ... vyprázdni"<<endl
          <<"5 ... konec"<<endl;
      cin>>vyber;
```

```
497   }while(vyber<1 or vyber>5);
498   return vyber;
499 }
500
501
502 int main () {
503   typZasobnik zasobnik;
504   string jmeno;
505   int pocet=0, menu;
506   do {
507     ukol=menu();
508     switch(ukol) {
509       case 1 :
510         cout<<"Zadejte jméno: ";
511         cin>>jmeno;
512         pridej(jmeno, zasobnik, pocet);
513         cout<<"Přidáno."<<endl;
514         break;
515       case 2 :
516         jmeno=odeber(zasobnik, pocet);
517         cout<<"Bylo odebráno jméno:" <<jmeno<<endl;
518         break;
519       case 3 :
520         vypis(zasobnik, pocet);
521         break;
522       case 4 :
523         vyprazdni(zasobnik, pocet);
524     }
525   }while (ukol!=5);
526   return 0;
527 }
```

Příklad 8.2 Podobně, jako tomu bylo v předchozím příkladu, realizujte abstraktní datový typ fronta. Tentokrát však nevyužívejte pole, ale potřebné místo v paměti alokujte dynamicky.

Řešení: I v tomto příkladu je za účelem jednoduššího ovládání vytvořeno menu, které nabízí operace přidej, odeber, vypiš, vyprázdni, počet jmen ve frontě a ukončení programu. Jednotlivé operace jsou pak realizovány podprogramy. U fronty se přidává na konec a bere se ze začátku. Při práci s dynamicky vytvářenými proměnnými je potřeba pracovat s ukazateli. Aby bylo možné vytvořit dynamickou frontu, je zapotřebí datový typ, který bude obsahovat potřebnou informaci a také ukazatel na další prvek. K tomu se nejlépe hodí záznam. Avšak nelze odkazovat na něco, co ještě neexistuje. Jak tento problém vyřešit, vidíte na začátku zdrojového kódu.

Fronta je obsluhována ze začátku i z konce, proto je typické, že k frontě se udržují dva ukazatele – jeden na začátek a jeden na konec. V hlavní funkci je to naznačeno proměnnými `lidezac` a `lidekon`.

Na začátku programu je fronta prázdná. Ukazatel na začátek i na konec tedy neukazuje nikam. To je specifikováno pomocí hodnoty `NULL`. Chce-li uživatel přidat do fronty další prvek, je zavolána procedura `pridejDoFronty`. Pokud je fronta prázdná, pomocí `f = new Osoba` se do proměnné `f` vloží adresa místa v paměti, alokovaného pro proměnnou typu `Osoba`. Procedura `nactiOsobu` spočívá v tom, že od uživatele načte jméno osoby a jako další nastaví `NULL`. To znamená, že za danou osobou již ve frontě nikdo není. Povšimněte si, jakým způsobem je předáván parametr. Zápis `*f` znamená, že se nepracuje s adresou, ale s paměťovým místem, na které adresa ukazuje. Pokud fronta není prázdná, pracuje se s ukazatelem na konec. Pro následníka posledního prvku fronty, který je nyní prázdný, se zavolá `new`. Tím dojde k vytvoření paměťového místa, se kterým bude pracovat procedura `nactiOsobu`, a zároveň k napojení nového prvku ke stávajícímu seznamu.

Při odebírání z fronty je odstraněn první prvek, tedy ten, na který ukazuje ukazatel v proměnné `fronta`. Odebírat je možné pouze, pokud se ve frontě nějaký prvek nachází. V případě, že ukazatel má hodnotu `NULL`, není co odebírat. V opačném případě se prvním prvkem fronty stane následník tohoto prvku, což zajistí příkaz `f=f->dalsi;`. Navíc funkce uvedená v kódu vrací odebíranou osobu. Toho je dosaženo tak, že ještě než dojde k přesunutí ukazatele na další prvek, je do pomocné proměnné `pom` vložen obsah paměti, na kterou ukazuje proměnná `f`, tedy začátek fronty. Obsah proměnné `pom` je pak návratovou hodnotou této funkce. Poslední, avšak velice důležitou částí je uvolnění paměti. Do proměnné `odstran` funkce vloží ukazatel na začátek fronty. Poté, co je začátek fronty přesunut na další prvek, je potřeba toto místo uvolnit. K tomu dochází příkazem `deleteodstran;`.

Procedura zajišťující výpis postupně projde všechny prvky a vypíše informace o osobě. Tím, že je parametr volán hodnotou, nedojde ke změně ukazatele a proměnná `fronta` bude ukazovat stále na začátek fronty.

Při vyprazdňování fronty dochází k volání funkce `odeberZfronty`, dokud fronta není prázdná. S návratovou hodnotou funkce se nijak dál nepracuje.

Počet jmen ve frontě se spočítá podobně jako při výpisu. Pouze místo výpisu konkrétní osoby se zvýší počitadlo, které je na konci vráceno.

Na konci programu je fronta vyprázdněna, a je tak i uvolněno místo v paměti.

```
528 #include <iostream>
529 using namespace std;
530 typedef struct Osoba Osoba;
531 struct Osoba{
532   string jmeno;
533   Osoba* dalsi;
534 };
535
536 void nactiOsobu(Osoba &clovek){
537   cout<< "jméno: ";
538   cin>>clovek.jmeno;
539   clovek.dalsi=NULL;
540 }
541
542 void vypisOsobu(Osoba clovek){
```

```
543   cout<< ”jméno: ”<<clovek.jmeno<<endl;
544 }
545
546 void pridejDoFronty(Osoba* &f, Osoba* &k){
547   if (f == NULL){
548     f = new Osoba;
549     nactiOsobu(*f);
550     k=f;  //na první prvek ukazuje i ukazatel na konec
551   }
552   else{
553     k->dalsi = new Osoba;
554     k=k->dalsi;
555     nactiOsobu(*k);
556   }
557 }
558
559 Osoba odeberZfronty(Osoba* &f, Osoba* &k){
560   Osoba pom = *f;
561   Osoba* odstran=f;
562   if(f!=NULL){
563     f=f->dalsi;
564     delete odstran;
565     if (f==NULL) k=NULL;
566        //je-li fronta prázdná, ukazatel na konec má být NULL
567   }
568   return pom;
569 }
570
571 void vypisFrontu(Osoba* f){
572   while(f!=NULL){
573     vypisOsobu(*f);
574     f=f->dalsi;
575   }
576 }
577
578 int pocetLidi(Osoba* f){
579   int poc =0 ;
580   while(f!=NULL){
581     poc++;
582     f=f->dalsi;
583   }
584   return poc;
585 }
586
587 int menu(){
```

```
588   int vyber;
589   do {
590      cout<<”1 ... přidej”<<endl
591          <<”2 ... odeber”<<endl
592          <<”3 ... vypiš”<<endl
593          <<”4 ... vyprázdni”<<endl
594          <<”5 ... počet jmen”<<endl
595          <<”6 ... konec”<<endl;
596      cin>>vyber;
597   } while(vyber<1 or vyber>6);
598   return vyber;
599 }
600
601 int main () {
602   Osoba* lidezac = NULL, lidekon = NULL;
603   Osoba clovek;
604   int pocet=0, ukol;
605   do {
606     ukol=menu();
607     switch(ukol) {
608       case 1 :
609         pridejDoFronty(lidezac, lidekon);
610         cout<<”Přidáno.”<<endl;
611         break;
612       case 2 :
613         clovek=odeberZfronty(lidezac, lidekon);
614         cout<<”Bylo odebráno ”;
615         vypisOsobu(clovek);
616         break;
617       case 3 :
618         vypisFrontu(lidezac);
619         break;
620       case 4 :
621         while (lidezac != NULL)
622            clovek=odeberZfronty(lidezac, lidekon);
623         cout<<”Vyprázdněno.”<<endl;
624         break;
625       case 5 :
626         cout<<pocetLidi(lidezac)<<endl;
627     }
628   }while (ukol!=6);
629   while (lidezac != NULL) //zrušení celé fronty na závěr
630      clovek=odeberZfronty(lidezac, lidekon);
631   return 0;
632 }
```

Příklad 8.3 Z textového souboru seznam.txt, který jste si připravili, vytvořte dynamický seznam, který následně vypište na výstup.
a) Jména načtěte do seznamu tak, jak jsou ve vstupním souboru.
b) Jména načtěte tak, aby byla v seznamu řazena abecedně (podle anglické abecedy).

Řešení: Pracovat se soubory již umíte. Neměl by tak být pro vás problém načíst jednotlivá jména. Přidání na konec seznamu již umíte z předchozího příkladu (fronta). Výpis byl již taky vyřešen v předchozím příkladu. Jediná novinka je zde tedy abacední řazení. Princip řazení byste již také měli znát. Při abecedním načítání je potřeba najít místo, kam prvek patří, a tam jej vložit. Jsou v podstatě tři možnosti.

a) Seznam je prázdný. Vytvoří se tedy nový prvek, na který bude ukazovat předávaný parametr `s`. Do nově vytvořeného místa se vloží jméno načtené ze souboru a jako následník se nastaví `NULL`.

b) Seznam není prázdný, ale přidávaný prvek patří na první místo. To, že prvek patří na první místo, se určí pomocí logického výrazu `s->jmeno>jmeno`. Pokud je jméno, na které ukazuje proměnná `s`, větší (v abecedě později) než načtené přidávané jméno, patří nový prvek na začátek. Do proměnné `pom` se vloží to, co je v proměnné `s`. Pro `s` se vytvoří nový prostor. Vloží se do něj přidávané jméno a jako následník se nastaví prvek, na který ukazuje proměnná `pom`, tedy původní první prvek.

c) Seznam není prázdný a prvek nepatří na začátek seznamu. V tomto případě se musí vyhledat místo, kam prvek patří. Pomocná proměnná `pom` bude procházet seznam, dokud nenajde správné místo, nebo nedorazí na konec seznamu. Logický výraz `pom->dalsi!=NULL` by vám měl být již jasný. Pokud se následovník nerovná `NULL`, je možné pokračovat. Matoucí může být logický výraz `pom->dalsi->jmeno<jmeno`. Algoritmus se potřebuje dívat, jaká hodnota je v následníkovi, protože prvek bude zařazen před něj. Je-li jméno dalšího prvku větší, cyklus se zastaví a prvek se vloží mezi prvek, na který ukazuje `pom`, a jeho následovníka. K tomu je potřeba další proměnná, která je v následujícím kódu pojmenována `novy`. Do místa, kam tato proměnná ukazuje, se vloží přidávané jméno a jako následník se nastaví následník `pom`. Následníkem prvku, na který ukazuje `pom`, se nově stane prvek, na který ukazuje `novy`. Pokud není nalezeno žádné jméno, které by bylo v abecedě později, je přidáno na konec. Přidávat na konec již umíte, takže není třeba to tu detailně popisovat.

```
633 #include <iostream>
634 #include <fstream>
635 using namespace std;
636 typedef struct Osoba Osoba;
637 struct Osoba{
638   string jmeno;
639   Osoba* dalsi;
640 };
641
642 void nacti_postupne (Osoba* &s, string nazev) {
643   Osoba* pom;
644   ifstream soubor(nazev);
645   string jmeno;
646   if (soubor.is_open()){
647     if (s == NULL){
648       soubor>>jmeno;
```

```
649       s = new Osoba;
650       s->jmeno = jmeno;
651       s->dalsi = NULL;
652     }
653     pom=s;
654     while (pom->dalsi!=NULL)
655        pom=pom->dalsi;
656     while (soubor>>jmeno){
657        pom->dalsi = new Osoba;
658        pom=pom->dalsi;
659        pom->jmeno=jmeno;
660        pom->dalsi=NULL;
661     }
662     cout<<”Soubor ”<< nazev <<” byl načten.”<<endl;
663     soubor.close();
664   }
665   else{
666     cerr<<”Soubor ”<< nazev <<” se nepodařilo otevřít.”<<endl;
667   }
668 }
669
670 void nacti_abecedne (Osoba* &s, string nazev) {
671   Osoba* pom;
672   Osoba* novy;
673   ifstream soubor(nazev);
674   string jmeno;
675   if (soubor.is_open()){
676     if (s == NULL){
677       soubor>>jmeno;
678       s = new Osoba;
679       s->jmeno = jmeno;
680       s->dalsi = NULL;
681     }
682     while (soubor>>jmeno){
683       pom=s;
684       if (s->jmeno>jmeno){
685         s = new Osoba;
686         s->jmeno=jmeno;
687         s->dalsi=pom;
688       }
689       else{
690         while(pom->dalsi!=NULL and pom->dalsi->jmeno<jmeno)
691             pom=pom->dalsi;
692         if(pom->dalsi == NULL){
693           pom->dalsi = new Osoba;
```

```
694             pom=pom->dalsi;
695             pom->jmeno=jmeno;
696             pom->dalsi=NULL;
697           }
698           else{
699             novy = new Osoba;
700             novy->jmeno=jmeno;
701             novy->dalsi=pom->dalsi;
702             pom->dalsi=novy;
703           }
704         }
705       }
706       cout<<”Soubor ”<< nazev <<” byl načten.”<<endl;
707       soubor.close();
708     }
709     else{
710       cerr<<”Soubor ”<< nazev <<” se nepodařilo otevřít.”<<endl;
711     }
712   }
713
714   void vypis(Osoba* s){
715     while(s!=NULL){
716       cout << s->jmeno<< ”, ”;
717       s=s->dalsi;
718     }
719   }
720
721   int main(){
722     Osoba* seznam = NULL;
723     nacti_postupne (seznam, ”seznam.txt”);
724     vypis(seznam);
725     cout<<endl;
726     seznam = NULL;
727     nacti_abecedne (seznam, ”seznam.txt”);
728     vypis(seznam);
729     return 0;
730   }
```

Příklady

Příklad 8.4 Pomocí pole vytvořte abstraktní datový typ fronta. Fronta bude mít operace přidej, odeber, vypiš a vyprázdni. Fronta uchovává jména osob. Do fronty je možno vložit maximálně 20 osob.

Příklad 8.5 Realizujte abstraktní datový typ zásobník. Tentokrát však nevyužívejte pole, ale potřebné místo v paměti alokujte dynamicky.

Příklad 8.6 Upravte třetí příklad tak, abyste byli schopni ze souboru https://akela.mendelu.cz/~xturcin0/algo/zamestnanci.txt načíst do dynamického seznamu informace o všech osobách. Při načítání je řaďte abecedně podle příjmení (platu, roku narození).

Příklad 8.7 Implementujte binární uspořádaný strom (veškeré informace jsou v příslušné přednášce) a použijte jej na seřazení jmen připravených na standardním vstupu.

Příklad 8.8 Implementujte operaci nad binárním stromem, která zjistí skutečný počet hladin daného stromu. Při použití stromu k řazení v předchozí úloze přidejte k seřazenému výstupu ještě informaci, kolik hladin měl strom použitý k seřazení. Porovnejte experimentálně získanou hodnotu s teoretickým minimálním počtem hladin při známém počtu vstupních dat.

Co máme po cvičení umět

- vytvořit záznam se složkou ukazující na tento záznam;
- pracovat se seznamem dynamických záznamů;
- vytvořit strukturu typu zásobník;
- vytvořit strukturu typu fronta;
- vytvořit a použít strukturu typu binární uspořádaný strom.

Kontrolní otázky

8.8 V čem se liší implementace zásobníku pomocí pole a pomocí dynamické struktury?

8.9 Jak lze vyřešit efektivní práci s oběma konci fronty?

8.10 Jaká data lze umístit do složek dynamického seznamu?

8.11 Co se stane, neuvolníme-li při odebírání prvků z dynamického seznamu paměť?

8.12 Jak se používá binární uspořádaný strom k řazení prvků?

8.13 Na čem závisí skutečný počet hladin binárního stromu? Liší se tento počet od teoretického množství hladin pro daný počet vstupních dat?