Pracovní list 5: Práce s dynamickými proměnnými

Co už máme znát

- datový typ ukazatel – definice;
- vytvoření (alokace) dynamické proměnné;
- zrušení (dealokace) dynamické proměnné;
- pole jako dynamická proměnná;
- znakové pole jako implementace řetězce;
- ukazatelová aritmetika;
- princip dynamického lineárního seznamu;
- způsob práce se seznamem jako s frontou nebo se zásobníkem;
- princip lineárního seznamu s aktivním prvkem;
- princip dvousměrného seznamu a kruhového seznamu.

Kontrolní otázky

5.1 Jak se zapisuje definice určitého ukazatele na zvolený typ?

5.2 Jak vytvoří dynamická proměnná v paměti?

5.3 Jak se zruší dynamická proměnná v paměti?

5.4 Co je reference a jak souvisí s ukazatelem?

5.5 Jak je implementováno pole v jazyce C++? Jakými dvěma způsoby lze s polem pracovat?

5.6 Co je ukazatelová aritmetika? Jak ji lze využít při práci s polem?

5.7 Jak se implementuje prvek lineárního dynamického jednosměrného seznamu?

5.8 Jaké operace implementuje zásobník, fronta, seznam s aktivním prvkem?

Příprava na cvičení

Ve cvičení budeme potřebovat překladač jazyka C++, editor pro přípravu zdrojových textů a vybavení příkazového řádku. Pro jednotlivé úlohy jsou k dispozici soubory s daty, případně výsledné soubory v adresáři `/home/rybicka/vyuka/progt/cecko/cviceni/cv05` na serveru akela. Konkrétní jména těchto souborů jsou uvedena u jednotlivých úloh.

Řešené příklady

Příklad 5.1 Deklarujte pole, do něhož je možné uložit až 100 tisíc řetězců o délce max. 180 znaků. Ze standardního vstupu do něj načtěte všechny řetězce a obsazené položky pole vypište do binárního souboru `retezce1.dat`. Analyzujte obsah výstupního souboru a zdůvodněte, proč má takovou délku.

Řešení: V definici řetězce použijeme znakové pole o 180 znacích, z těchto řetězců pak vytvoříme pole alokací příslušné paměti pro 100 tisíc položek. Celé pole v paměti zabírá 100 000 · 180 B plus ukazatele na řetězce, tj. plus 800 000 B. Po přečtení všech řetězců ze standardního vstupu se vypíší řetězce z naplněných položek do výstupního souboru. Řetězce se vypisují celé. Použijeme-li na standardním vstupu přesměrovaný soubor `data.txt`, obdržíme výsledný soubor o velikosti 1 698 300 B. Analýzou pomocí příkazu `od -ctx1 retezce1.dat` zjistíme, že každý řetězec ve výstupním souboru zabírá 180 B. Počet přečtených řetězců je 9 435, což vynásobeno 180 dává výslednou délku souboru.

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

int main(){
  const int MaxPocRet = 100000;
  const int MaxDelka = 180;
  typedef char TypRet[MaxDelka];

  TypRet *Pole = new TypRet[MaxPocRet];
  TypRet Cteni;
  unsigned int index=0;

  while (cin.getline(Cteni, MaxDelka)) {
    strcpy(Pole[index], Cteni);
    index++;
  }

  ofstream vyst (”x.dat”, ios::binary);
  for (int i=0; i<index; i++) vyst.write(Pole[i],MaxDelka);
  vyst.close();
  return 0;
}
```

Příklad 5.2 Na standardním vstupu se nachází posloupnost znakových řetězců (přesměrujte na standardní vstup soubor `data.txt`). Uložte tyto řetězce do dynamického jednosměrného lineárního seznamu. Na standardní výstup vypište z tohoto seznamu každý 100. řetězec počínaje od konce.

Řešení: Definice jednosměrného dynamického seznamu zahrnuje definici prvku – záznamu o dvou složkách: data a ukazatel. Pro řešení úlohy potřebujeme operaci vložení na konec seznamu – jde o implementaci seznamu v roli fronty. Při vkládání se proto vyplatí udržovat ukazatel na poslední prvek seznamu. Pro výpis řetězců pak využijeme cyklus `for`, v němž budeme hledat v seznamu řetězec s pořadovým číslem snižujícím se od celkového počtu řetězců postupně vždy o 100. Vkládání na konec, vyhledání *n*-tého řetězce a jeho výpis implementujeme jako podprogramy. Protože v jejich parametrech budeme potřebovat ukazatele na prvky, provedeme ještě před uvedením podprogramů potřebné typové definice.

V jednosměrném lineárním seznamu musíme vždy procházet od začátku. Je zřejmé, že tato úloha je jednosměrným seznamem implementována neefektivně, výhodnější by byl seznam dvousměrný. Na ten přijde řada v jiném příkladu.

```
206 #include <iostream>
207 using namespace std;
208
209 const int MaxDelkaRet = 180;
210 typedef char TypRet[MaxDelkaRet];
211 typedef struct Prvek {
212   TypRet data;
213   Prvek *dalsi;
214 } Prvek;  //definice prvku seznamu
215
216 typedef Prvek *UkPrvek;  //ukazatel na prvek
217 typedef struct Fronta {
218    UkPrvek Zac, Kon;     //fronta: 2 ukazatele
219 } Fronta;
220
221 void Pridej(Fronta &F, TypRet R){
222   if (F.Zac==NULL) {       //fronta prázdná
223     F.Zac = new Prvek;
224     F.Kon = F.Zac;
225   } else {                 //fronta neprázdná
226     F.Kon->dalsi = new Prvek;
227     F.Kon = F.Kon->dalsi;
228   }
229   strcpy(F.Kon->data, R);   //vložení dat
230   F.Kon->dalsi=NULL;        //prvek je nyní poslední
231 }
232
233 void VypisN(Fronta F, int N){
234   int Pozice=1;
235   UkPrvek Pom=F.Zac;       //pomocný ukazatel pro průchod
236   while (Pom!=NULL and Pozice<N) {
237      Pozice++;
238      Pom=Pom->dalsi;       //přechod na další prvek
```

```
239   }
240   if (Pom!=NULL) cout << Pom->data << endl;
241                              //výpis, nejsme-li za koncem
242 }
243
244 int main(){
245   Fronta F;
246   int Kolik=0;
247   TypRet Cteny;
248   while (cin.getline(Cteny, MaxDelkaRet)) { //čtení
249     Kolik++;   //počítání řetězců
250     Pridej(F, Cteny); //přidávání do fronty
251   }
252   for (int i=Kolik-100; i>0; i-=100) //indexy výpisů
253     VypisN(F, i);
254   return 0;
255 }
```

Příklady

Příklad 5.3 Přečtěte ze standardního vstupu posloupnost znaků a uložte ji do znakového pole o deklarované velikosti 1000 prvků. Při čtení vynechávejte bílé znaky. Na standardní výstup vypište každý desátý znak počínaje od konce. K přístupu do znakového pole využívejte ukazatelovou aritmetiku.

Příklad 5.4 Optimalizujte uložení řetězců z příkladu 5.1 tak, aby zabíraly v paměti jen tolik místa, kolik je nezbytně potřeba. Velikost základního pole ponechte stejnou. Pro kontrolu správnosti uložení a přístupů k jednotlivým řetězcům vypište na standardní výstup každý 100. řetězec počínaje od konce. Pro ladění využijte souboru `data.txt` přesměrovaného na standardní vstup.

Příklad 5.5 Proveďte další optimalizaci uložení řetězců – velikost základního pole. Vytvořte pole 1000 ukazatelů na stoprvková pole řetězců. Stoprvkové segmenty alokujte až v případě potřeby. Kontrolu správnosti uložení a přístupu k jednotlivým řetězcům proveďte stejným způsobem jako v příkladu 5.4. Po zpracování všech dat uvolněte veškerou paměť využívanou na ukládání řetězců.

Příklad 5.6 Posloupnost řetězců ze standardního vstupu uložte do lineárního jednosměrného dynamického seznamu fungujícího jako zásobník. Na standardní výstup vypište vstupní řetězce v obráceném pořadí.

Příklad 5.7 Posloupnost řetězců ze standardního vstupu uložte do dvousměrného dynamického lineárního seznamu a na výstup vypište každý 100. řetězec počínaje od konce.

Příklad 5.8 Implementujte jednosměrný dynamický lineární seznam s aktivním prvkem s operacemi: inicializace, vložení prvku za aktivní prvek, odstranění prvku za aktivním prvkem, výpis obsa-

hu aktivního prvku, nastavení aktivity na začátek, posun aktivity na další prvek, test aktivity. Operace implementujte jako podprogramy. Využijte tento seznam pro uložení posloupnosti řetězců ze standardního vstupu, v seznamu pak odstraňte každý 10. řetězec a výsledný seznam vypište na standardní výstup. Jako testovací vstup můžete použít zdrojový text tohoto programu.

Příklad 5.9 Modifikujte implementaci seznamu s aktivním prvkem z příkladu 5.8 tak, aby byl dvousměrný. Přidejte operaci nastavení aktivity na konec a přesunu aktivity na předchozí prvek. Využijte tento seznam pro uložení řetězců ze standardního vstupu, v němž pak vymažete každý desátý řetězec a zbytek vypíšete na standardní výstup v obráceném pořadí.

Příklad 5.10 Implementujte lineární jednosměrný kruhový seznam a uložte do něj prvních 10 řetězců ze standardního vstupu. Následně čtěte další řetězce ze vstupu až do konce a na standardní výstup vypisujte dvojice řádků: jeden bude obsahovat právě přečtený řetězec ze vstupu, druhý pak řetězec z kruhového seznamu. Po výpisu dvojice se provede přesun v kruhovém seznamu na další prvek.

Co máme po cvičení umět

- Práce s polem jako s dynamickou strukturou, ukazatelová aritmetika.
- Způsoby uložení dat v poli, optimalizace spotřeby paměti.
- Princip implementace jednosměrného dynamického seznamu.
- Princip implementace dvousměrného a kruhového seznamu.

Kontrolní otázky

5.9 Jakou zásadní výhodu má ukládání dat v poli proti lineárnímu seznamu?

5.10 Jak pracuje ukazatelová aritmetika?

5.11 Jak lze optimalizovat velikost pole pro ukládání variabilního většího množství dat?

5.12 Které operace je potřebné implementovat pro ovládání aktivity v seznamu s aktivním prvkem?

5.13 V čem spočívá výhoda lineárního seznamu fungujícího jako zásobník?

5.14 Jaké výhody má lineární dvousměrný seznam?

5.15 Jaké odlišnosti v jednotlivých operacích má lineární kruhový seznam od obyčejného lineárního seznamu?