Abstraktní datové typy

Přednáška 5

Karty

- Úloha:
  - Programujeme karetní hru (Prší, Solitaire, Poker…)
  - Jako první krok potřebujeme kód pro „obsluhu karet“
    1. Připravit si strukturu pro karty
    2. Karty do struktury uložit
    3. Připravit si funkce pro:
       a) vybrání následující karty
       b) přidání karty (např. z jiného balíčku)
       c) zamíchání

Karty – Pole

1. Připravit si strukturu pro karty

```
struct TPackArray {
    size_t iCount;
    short iValues[CARD_MAX_COUNT];
 }
```

2. Karty do struktury uložit
3. Připravit si funkce pro:
   a) vybrání následující karty
   b) zamíchání
   c) přidání karty (např. z jiného balíčku)

Karty – Pole

```
struct TPackArray {
      size_t iCount;
      short iValues[CARD_MAX_COUNT];
}
```

a) vybrání následující karty

```
returnVal = struktura->iValues[0];
for (int i=0; i< struktura->iCount-1; i++)
   struktura->iValues[i] = struktura->iValues[i+1];
```

b) přidání karty (např. z jiného balíčku):

   a) na konec

```
struktura->iValues[struktura->iCount] = aVal;
struktura->iCount++;
```

   b) na začátek

```
for (int i=struktura->iCount; i>0; i--)
   struktura->iValues[i]= struktura->iValues[i-1];
struktura->iValues[0] = aVal;
struktura->iCount++;
```

c) zamíchání

Karty – Pole

```
struct TPackArray {
	size_t iCount;
	short iValues[CARD_MAX_COUNT];
}
```

c) Zamíchání
   1. Vytvořit si pomocné pole
   2. Překopírovat část karet z prostředku
   3. Na uvolněné místo nakopírovat karty z konce balíčku
   4. Za tyto karty nakopírovat karty z pomocného pole
   5. Opakujeme N-krát

Karty - Lineární seznam

```
struct TPackList {
      struct TPackListNode *iTop;
      struct TPackListNode *iLast;
}

struct TPackListNode {
      struct TPackListNode *iNext;
      short iValue;
}
```

a) vybrání následující karty

```
returnCard = struktura->iTop;
returnVal  = struktura->iTop->iValue;
struktura->iTop = returnCard->iNext;
```

Karty - Lineární seznam

```
struct TPackList {
        struct TPackListNode *iTop;
        struct TPackListNode *iLast;
}

struct TPackListNode {
        struct TPackListNode *iNext;
        short iValue;
}
```

b) přidání karty (např. z jiného balíčku):

a) na konec

```
novaKarta = (struct TPackListNode*)malloc( … );
novaKarta->iValue = aVal;
novaKarta->iNext = NULL;
struktura->iLast->iNext = novaKarta;
struktura->iLast = novaKarta;
```

b) na začátek

```
novaKarta = (struct TPackListNode*)malloc( … );
novaKarta->iValue = aVal;
novaKarta->iNext = struktura->iTop;
struktura->iTop = novaKarta;
```

Karty - Lineární seznam

```
struct TPackList {
	struct TPackListNode *iTop;
	struct TPackListNode *iLast;
}

struct TPackListNode {
	struct TPackListNode *iNext;
	short iValue;
}
```

c) Zamíchání

1. Vytvoříme si 2 ukazatele (`pA`, `pB`) na náhodných pozicích v seznamu:
   `iTop -> … -> pA -> … -> pB -> … -> iLast -> NULL`
2. Provedeme několik změn ve vazbách:
   ```
   iLast->iNext    = pA->iNext;
   pA->iNext       = pB->iNext;
   pB->iNext       = NULL;
   iLast           = pB;
   ```
3. Opakujeme N-krát

Karty – Pole indexů

```
struct TPackArray {
	size_t iCount;
	short iValues[CARD_MAX_COUNT];
	short iIndexes[CARD_MAX_COUNT];
}
```

- Přidali jsme pole indexů `iIndexes`
- `iIndexes` určuje pořadí vybírání karet z `iValues`
- Jak budou vypadat funkce pro:
  a) vybrání následující karty?
  b) zamíchání?
  c) přidání karty (např. z jiného balíčku)?

Iterátory

- Též označováno jako Kurzory
- Návrhový vzor pro procházení datové struktury
- Oddělení implementace od použití
- Vytváří jednotné rozhraní
- Spojeny s příkazy foreach/for/while pro procházení a vybírání prvků

Iterátory - for

```
Iterátor {
      kontejner;
      pozice;
}

for ( iterátor = iterátor_begin();
      iterátor_valid(iterátor);
      iterátor_next(iterátor);
      )
{
      vypiš ( iterátor_aktuální_hodnota ( iterátor ) );
}
```

Iterátory - for

```
iterátor = iterátor_begin ( )
iterátor_valid ( iterátor )
iterátor_next ( iterátor )
iterátor_aktuální_hodnota ( iterátor )
```

- Co budou dělat tyto funkce u zásobníku realizovaném:
  a) Statickým polem?
  b) Jednosměrně lineárně vázaným seznamem?

Iterátory – for – statické pole

```
struct Tstack {
      size_t iCount;
      TStackElement iValues[STACK_MAXCOUNT];
}

struct TStackIterator {
      const struct TStack *iStack;
      size_t iPos;
}

iterátor = iterátor_begin ( )        //vytvoří nový objekt

iterátor_valid ( iterátor )          //došli jsme na konec?

iterátor_next ( iterátor )           //přejdi na další

iterátor_aktuální_hodnota ( iterátor )
                                     //vrátí aktuální hodnotu
```

Iterátory – for – lineární vázaný seznam

```
struct Tstack {
      struct TStackNode *iTop;
}

struct TStackNode {
      struct TStackNode *iNext;
      TStackElement iValue;
}

struct TStackIterator {
      const struct TStack *iStack;
      struct TStackNode *iActual;
}

iterátor = iterátor_begin ( )        //vytvoří nový objekt

iterátor_valid ( iterátor )          //došli jsme na konec?

iterátor_next ( iterátor )           //přejdi na další

iterátor_aktuální_hodnota ( iterátor )
                                     //vrátí aktuální hodnotu
```

Iterátory - while

```
while ( není_prázdný ( seznam->další_prvek ) )
{
     vypiš ( získej_další_prvek ( seznam ) );
}
```

Iterátory - foreach

```
foreach ( aktuální_hodnota in seznam )
{
     vypiš (aktuální_hodnota );
}
```

Abstraktní datové typy

- Typ dat
  - Definované chování
  - Nezávislost na implementaci
  - Možnost uložit různé typy dat
  - Zapouzdření (často struct)
- Práce s ADT
  - Programátorská/autorská
  - Uživatelská

- Např. množina:
  - Realizace
    - Pole
    - Lineární seznam
    - Strom
  - Vždy obsahuje funkce pro:
    - Vložení prvku
    - Vybrání prvku
    - …a další

ADT – obecné funkce

- compare – porovnání zda jsou proměnné stejné
- print, show – výstup v „lidské“ formě
- create – vytvoření nové proměnné
- initialize – nastavení (nově vytvořené) proměnné do základního stavu
- copy – zkopírování dat z jedné proměnné do druhé
- clone – vytvoří novou proměnnou (create) a inicializuje ji daty z jiné (copy)
- free, destroy – korektní ukončení proměnné
- empty – zjistí, zda jsou data
- size – velikost, počet prvků

ADT – přehled

- zásobník
- fronta (prioritní, obousměrná)
- seznam
- množina / multimnožina
- asociativní pole
- strom
- hašovací tabulka
- řetězec

Zásobník (Stack)

- LIFO – Last In First Out
- push – vložení prvku
- pop – vybrání posledního vloženého prvku (nemusí být vrácen ale pouze zrušen)
- peek/top – zpřístupnění posledního vloženého prvku bez jeho vyjmutí
- přístupný pouze z jedné strany
- lze realizovat jednosměrně vázaným seznamem

Fronta (queue)

- FIFO – First In First Out
- enqueue – vložení prvku
- dequeue – vybrání prvku
- přístupný pouze z jedné strany
- lze realizovat
  - posuvným registrem
  - kruhovým bufferem

Fronta prioritní

- fronta, ke každé hodnotě navíc přiřazena priorita
- prvky s vyšší prioritou řazeny dopředu k přednostnímu zpracování
- insert (insert_with_priority) – pro vložení
- pull (pull_highest_priority) – vyjme prvek podle priority (a pořadí)
- peek/top – vrátí prvek, ale nezmění frontu
- přístupný pouze z jedné strany
- lze realizovat
  - posuvným registrem
  - kruhovým bufferem

Fronta obousměrná (DEQUE)

- Double Ended QUEue – DEQUE
- push_back; push_front – vložení prvku
- pop_back; pop_front – vybrání prvku
- back; front – zpřístupnění prvku
- přístupný z obou stran

- možná i varianta s prioritou

Seznam (list, sequence)

- konečný počet uspořádaných prvků
- možné opakování prvků
- operace:
  - vložení na začátek
  - vložení na konec
  - vyjmutí prvního prvku
  - vrácení prvku na dané pozici

Množina (set)

- seznam s podmínkami
- skládá se s unikátních prvků
- nezáleží na pořadí prvků
- operace
  - union (sjednocení) – všechny prvky
  - intersection (průnik) – společné prvky
  - difference – rozdílné prvky
  - subset – test zde je podmnožinou

Multimnožina (multiset)

- může obsahovat více stejných prvků
- při porovnávání může nebo nemusí záležet na pořadí a počtu – podle filozofie návrhu
- realizace
  - samostatné hodnoty
  - hodnota + počet
- operace
  - union (sjednocení) – všechny prvky
  - intersection (průnik) – společné prvky
  - difference – rozdílné prvky
  - subset – test zde je podmnožinou

Asociativní pole (associative array, dictionary, map)

- skládá se z dvojce klíč – hodnota
- operace
  - add / insert – přidání prvku
  - remove / delete – odstranění prvku

Strom (tree)

- stromová struktura prvků s propojenými uzly
- kořen – výchozí uzel
- uzel může obsahovat
  - hodnotu
  - podmínku
- list – nemá potomka
- cesta – posloupnost uzlů vedoucí k danému listu
- délka cesty
- hloubka uzlu
- šířka stromu
- procházení stromu
  - do hloubky
  - do šířky

Hašovací tabulka (Hash table)

- vyhledávací datová struktura
- asociuje hašovací klíče s odpovídajícími hodnotami
  ```
  pole[hash_fce(hodnota)] = záznam()
  pole[hash_fce("jmeno_prijmeni")]
  ```
- pro rychlé vyhledávání v poli
- kolize klíčů

Řetězec (string)

- pro uložení konečné posloupnosti znaků
- implementace
  - délka je definována ukončovacím znakem `\0`
    - v C definované jako char[]
  - délka uložena v pomocné struktuře
- operace
  - Tisk řetězce
  - Získání znaku z pozice
  - Délka řetězce
  - Rozdělení/spojování řetězců
  - Porovnání obsahu
  - Prohledávání obsahu (znaku, podřetězce)