Abstraktní datový typ

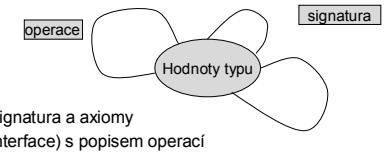
A0B36PR1-Programování 1
Fakulta elektrotechnická
České vysoké učení technické

Datové struktury

- Klasická kniha o programování prof. Wirtha má název
  Algorithms + Data structures = Programs
- Datová struktura (typ) = množina dat + operace s těmito daty
- Abstraktní datový typ - formálně definuje data a operace s nimi, např.
  - Fronta (Queue)
  - Zásobník (Stack)
  - Pole (Array)
  - Tabulka (Table)
  - Seznam (List)
  - Strom (Tree)
  - Množina (Set)

Abstraktní datový typ

- je množina *druhů dat* (hodnot) a příslušných *operací*, které jsou přesně specifikovány, a to ***nezávisle na konkrétní implementaci***.



- Definovat lze
  - Matematicky – signatura a axiomy
  - Jako rozhraní (interface) s popisem operací
    - Interface poskytuje
      - konstruktor, který vrací abstraktní odkaz a
      - operace, které akceptují odkaz jako argument a které mají přesně definovaný účinek na data

Příklad1: ADT - popis matematicky I

- Syntaxe popisuje, jak správně vytvořit logický výraz:
  1. true, false jsou logické výrazy,
  2. if x,y jsou logické výrazy, pak
     i. ! (x), negace
     ii. (x & y), logický součin, and
     iii. (x | y), logický součet, or
     iv. (x==y), (x!=y), relační operátory

     jsou logické výrazy. Pokud nechceme u každé operace psát závorky, musíme definovat priority operátorů.

Datový typ boolean

- Sémantika popisuje význam jednotlivých operací. Lze definovat pomocí axiomů:

  1)!(true) = false 2) !(false) = true
  3) x & false = false 4) x & true = x
  5) x & y = y & x 6) x | true = true
  7) x | false = x 8) x | y = y | x

Příklad1: ADT - popis matematicky II

Datový typ boolean

– Sémantika, pokračování

9) true == true = true 10) true == false = false
11) false == false = true 11) false == true = false

– nebo lépe(vhodnější pro úpravy):

9) true == x = x 10) false == x = !x
11) x==y = y==x

– … dále by následovala sémantika pro !=

Příklad2: ADT - programově

```
public interface Citac {
   public int getHodnota();
   public void zvetsi();
   public void zmensi();
   public void reset();
}
public class MujCitac implements Citac {
   private int hodnota = 0;
   public int getHodnota(){return hodnota;}
   public void zvetsi(){hodnota++;}
   public void zmensi(){hodnota--;}
   public void reset(){hodnota = 0;}
}
```

Abstraktní datový typ, pouze rozhraní

Implementace datového typu

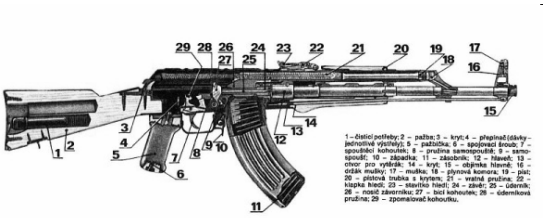
Pro uživatele je implementace skryta, používá jen veřejné metody objektu

(Abstraktní) datový typ - dělení

- Počet složek:
  - neměnný = statický datový typ, počet položek je konstantní, pole, řetězec, třída
  - proměnný = dynamický datový typ, počet složek je proměnný, mezi operace patří vložení, odebrání určitého prvku
- Typ položek, dat:
  - homogenní = všechny položky stejného typu
  - nehomogenní = různého typu
- Existence bezprostředního následníka
  - lineární = existuje [např. pole, fronta, seznam,…]
  - nelineární = neexistuje [strom, tabulka,…]

Zásobník (Stack)

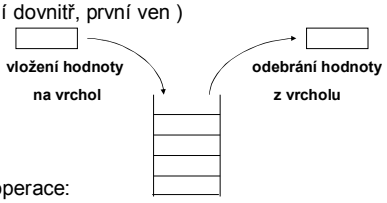
Je to dynamická datová struktura, umožňující vkládání a odebírání hodnot, přičemž naposledy vložená hodnota se odebere jako první



Zásobník (Stack)

Je to dynamická datová struktura, umožňující vkládání a odebírání hodnot, přičemž naposledy vložená hodnota se odebere jako první

Je to paměť typu LIFO (zkratka z angl. Last-In First-Out, poslední dovnitř, první ven)



Základní operace:

vložení hodnoty na vrchol zásobníku
odebrání hodnoty z vrcholu zásobníku
test na prázdnost zásobníku

Zásobník

- Příklad třídy realizující zásobník znaků:

```
class ZasobnikZnaku {
  public ZasobnikZnaku() {...}
  public void vloz(char z) { ... }
  public char odeber() { ... }
  public boolean jePrazdny() { ... }
  ...
}
```

- Poznámky
  - v angličtině se operace nad zásobníkem obvykle jmenují *push*, *pop* a *isEmpty*
  - pro zásobník může být definována ještě operace čtení hodnoty z vrcholu zásobníku bez jejího odebrání (v angl. *top* či *peek*)
- Implementací zásobníku se prozatím nebudeme zabývat, ukážeme si nejprve jeho použití

Příklad použití zásobníku

- Program, který přečte výraz obsahující tři druhy závorek a zkontroluje jejich správné párování
  - příklad správného párování: { [ ] ( ) }
  - příklad nesprávného párování: { [( ]) }
- Nástin řešení:
  1. Postupně projdeme všechny znaky tvořící výraz. Pro každý znak mohou nastat tři případy:
     - znakem je otevírací závorka: uložíme ji do zásobníku
     - znakem je zavírací závorka: ze zásobníku (musí být neprázdný) odebereme naposledy uloženou otevírací závorku a zjistíme, zda odpovídá zavírací závorce
     - jiný znak: žádná operace
  2. Po zpracování všech znaků výrazu musí být zásobník prázdný (pokud není, pak chybí zavírací závorky)

Párování závorek

- **Pomocné funkce:**

```
static boolean jeOteviraci(char z) {
  return z=='(' || z=='[' || z=='{';
}

static boolean jeZaviraci(char z) {
  return z==')' || z==']' || z=='}';
}

static char zaviraciK(char z) {
  if (z=='(') return ')';
  if (z=='[') return ']';
  if (z=='{') return '}';
  return z;
}

static void chyba(String str) {
  Systém.out.println( " chyba: " +str);
  System.exit(0);
}
```

Párování závorek

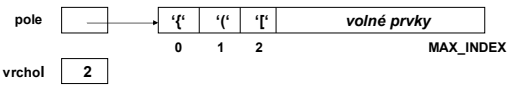
- **Hlavní metoda:**

```
public class Zavorky {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    ZasobnikZnaku zasobnik = new ZasobnikZnaku();
    Systém.out.println( "zadejte radek se zavorkami" );
    String str = sc.nextLine();
    int i;
    for (i=0; i<str.length(); i++) {
      char znak = str.charAt(i);
      System.out.print(znak);
      if (jeOteviraci(znak)) zasobnik.vloz(znak);
      else if (jeZaviraci(znak)) {
       if (zasobnik.jePrazdny())
        chyba( "k zavorce chybi otevíraci" );
       char ocekavany =zaviraciK(zasobnik.odeber());
           if (znak!=ocekavany)
              chyba( "ocekava se " +ocekavany);
      }
    }
```


Párování závorek

```
if ( !zasobnik.jePrazdny() ) {
     String chybi = "" ;
     do {
       chybi = chybi + zasobnik.odeber();
     } while ( !zasobnik.jePrazdny() );
     chyba( "chybi zaviraci zavorky k " +chybi );
   }
   Systém.out.println();
 }
```


Implementace zásobníku polem




```
class ZasobnikZnaku {
  static final int MAX_INDEX = 99;
  private char[] pole;
  private int vrchol;

  public ZasobnikZnaku() {
    pole = new char[MAX_INDEX+1];
    vrchol = -1;
  }
```

Implementace zásobníku polem

```
public void vloz(char z) {
   if ( vrchol==MAX_INDEX )
      throw new RuntimeException( "plný zásobník" );
       pole[++vrchol] = z;
    }
public char odeber() {
   if ( vrchol<0 )
    throw new RuntimeException( "prázdný zásobník" );
      return pole[vrchol--];
    }
public boolean jePrazdny() {
      return vrchol<0;
    }
  }
```

- Poznámka: *RuntimeException* je třída výjimek, jejichž šíření z metody netřeba deklarovat v hlavičce metody

Implementace zásobníku rozšiřitelným polem

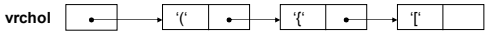
- Při vkládání do plného zásobníku lze vytvořit nové, větší pole a původní pole do něj zkopírovat

```
public class ZasobnikZnaku2 {
    private char[] pole;
    private int vrchol;
  public ZasobnikZnaku2() {
      pole = new char[2];
      vrchol = -1;
    }
  public void vloz(char z) {
      if (vrchol==pole.length-1) {
        char[] nove = new char[2*pole.length];
        System.arraycopy(pole,0,nove,0,pole.length);
        pole = nove;
      }
      pole[++vrchol] = z;
    } ...}
```

dvojnásobná velikost nového pole zaručuje průměrnou konstantní dobu vložení jednoho prvku

Implementace zásobníku spojovým seznamem

- Spojový seznam je datová struktura, jejíž prvky tvoří posloupnost a každý prvek obsahuje odkaz na další prvek seznamu




```
class Prvek {
  char hodn;
    Prvek dalsi;

  public Prvek(char h, Prvek p) {
      hodn = h; dalsi = p;
    }
  }
class ZasobnikZnaku {
    private Prvek vrchol;

  public ZasobnikZnaku() {
      vrchol = null;
    }
}
```

Implementace zásobníku spojovým seznamem

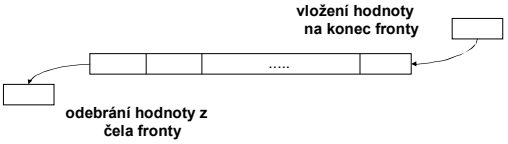
```
    public void vloz(char z) {
      vrchol = new Prvek(z, vrchol);
    }

    public char odeber() {
      if (vrchol==null)
        throw new RuntimeException( "prazdny zasobnik"
    );
      char vysl = vrchol.hodn;
      vrchol = vrchol.dalsi;
      return vysl;
    }

    public boolean jePrazdny() {
      return vrchol==null;
    }
  }
```


Fronta (Queue)

- Fronta je datová struktura v níž se hodnoty odebírají v tom pořadí, v jakém byly vloženy

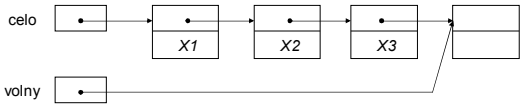
Fronta (Queue)

- Fronta je datová struktura v níž se hodnoty odebírají v tom pořadí, v jakém byly vloženy
- Je to paměť typu FIFO (zkratka z angl. First-In First-Out, první dovnitř, první ven)



- Implementace fronty:
  - pomocí pole
  - pomocí spojového seznamu

Implementace fronty spojovým seznamem



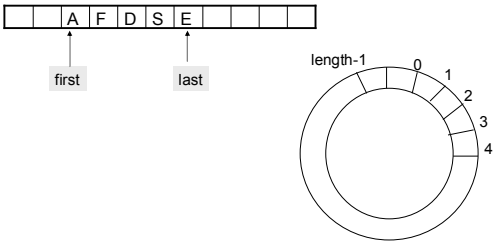

```
class PrvekFronty {
  PrvekFronty dalsi;
  int hodn;
}

public class FrontaCisel {
  private PrvekFronty celo;
  private PrvekFronty volny;
```


Implementace fronty spojovým seznamem

```
public FrontaCisel() {
   celo = new PrvekFronty();
   volny = celo;
 }
 public void vloz(int x) {
   volny.hodn = x;
   volny.dalsi = new PrvekFronty();
   volny = volny.dalsi;
 }
 public int odeber() {
   if (jePrazdna())
     throw new RuntimeException( " prazdna fronta"
);
   int vysl = celo.hodn;
   celo = celo.dalsi;
   return vysl;
 }
 public boolean jePrazdna() {
   return celo == volny;
 }
}
```


Implementace fronty - kruhový buffer

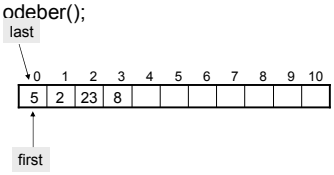
- implementace pomocí pole
- dva ukazatele
  - first, ukazatel na první prvek fronty, bude první čten
  - last, ukazatel na poslední prvek, sem se přidává



Implementace fronty - kruhový buffer, příklad

- vloz(5);
- vloz(2);
- vloz(23);
- další = odeber();
- vloz(8);
- další = odeber();
- další = odeber();

–fronta je prázdná, pokud last == first
–po vložení (výběru) prvku se nová hodnota ukazatele vypočítá: last = (last+1)%pole.length
–jak se pozná, že je fronta plná?



ADT množina

Operace

- **add**, vložení prvku (bez opakování)
- **remove**, smazání prvku
- **contains**, test, zda je prvek v množině obsažen
- **isEmpty**, test na prázdnost
- **size**, zjištění velikosti
- **equals**, test na shodu
- **containsAll(množina)**, zjištění, zda je daná množina podmnožinou

x.add(45) … bez efektu
x.add(15)
x.remove(8)

x.contains(3) …. true
x.isEmpty … false
x.size() … 7