IZP 14

Dynamické datové struktury

Jitka Kreslíková, Aleš Smrčka

2022

Fakulta informačních technologií
Vysoké učení technické v Brně

IZP – Základy programování

IZP 14

Dynamické datové struktury

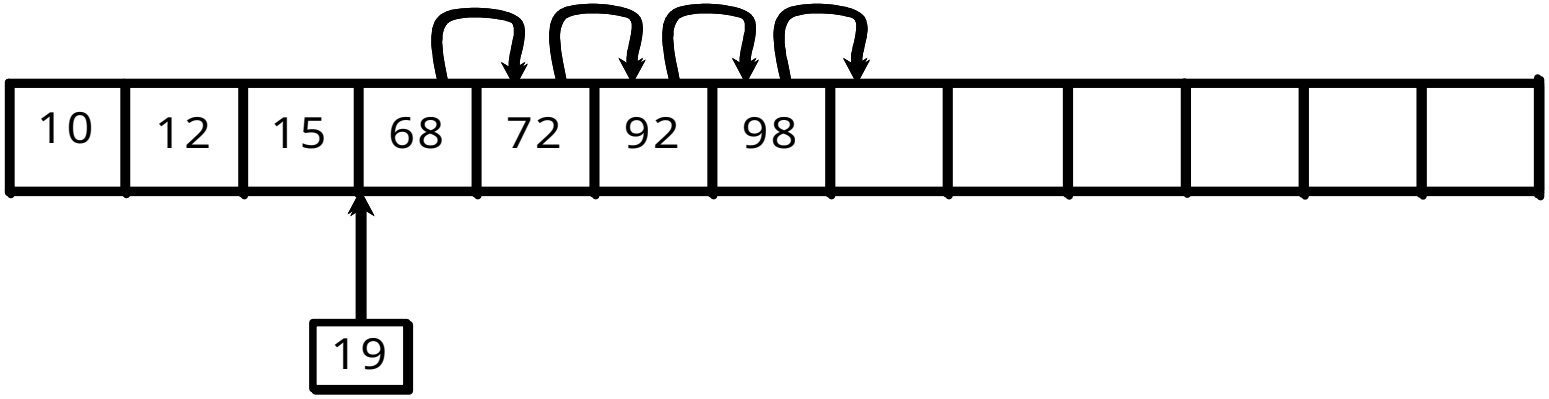
- Obecné vlastnosti dynamických datových struktur
- Abstraktní datové typy

Obecné vlastnosti dynamických datových struktur

- Základní nevýhoda statických datových struktur (pole, struktura) – pevná velikost.
- V praktickém programování často potřebujeme paměťové struktury s proměnnou velikostí.
- Typickou strukturou je seznam prvků určitého typu (např. záznamů o osobách). Takováto struktura reprezentována staticky polem generuje zásadní problémy:
  - pole musí obvykle obsahovat prázdné prvky jako rezervu pro přidávání,

Obecné vlastnosti dynamických datových struktur

- přidání prvku na konkrétní místo způsobí časově náročný odsun všech následujících prvků, obdobně rušení prvku.



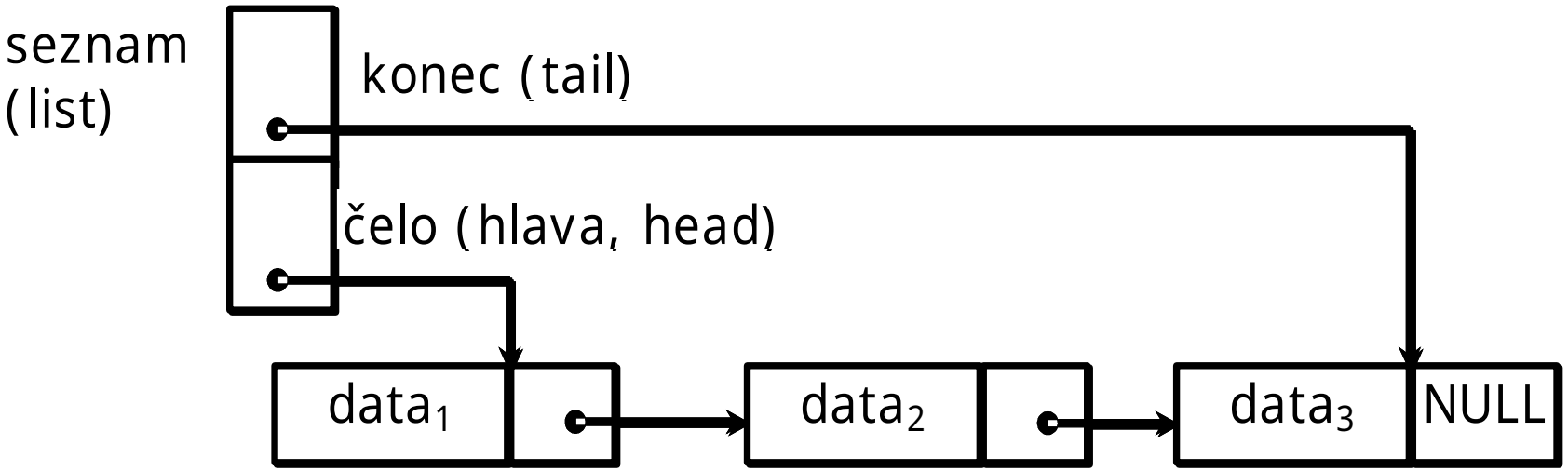
Výhodou této struktury je okamžitý přístup k libovolnému prvku přes index.

Obecné vlastnosti dynamických datových struktur

- Vhodnější řešení nabízejí dynamické datové struktury:
  - Skládají se z prvků (položek) svázaných ukazateli.
  - Základem bývají obvykle staticky deklarované ukazatele a typ záznamu, obsahující kromě užitečných dat ještě ukazatel(e) na stejný typ.
  - Dynamicky vytvořené proměnné uvedeného typu pak lze velmi snadno spojovat pomocí ukazatelů.

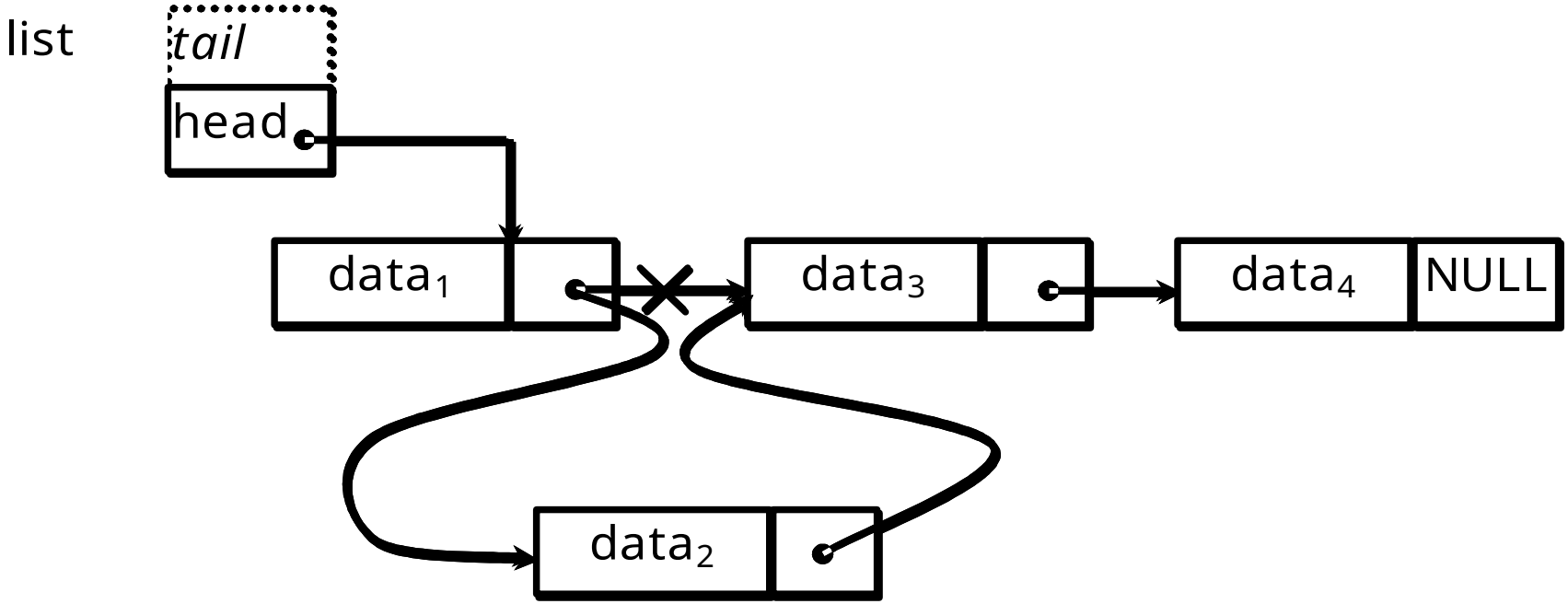
Obecné vlastnosti dynamických datových struktur

- Lze vytvořit strukturu se snadným vkládáním a rušením s takovým počtem prvků, jaký je aktuálně potřebný.



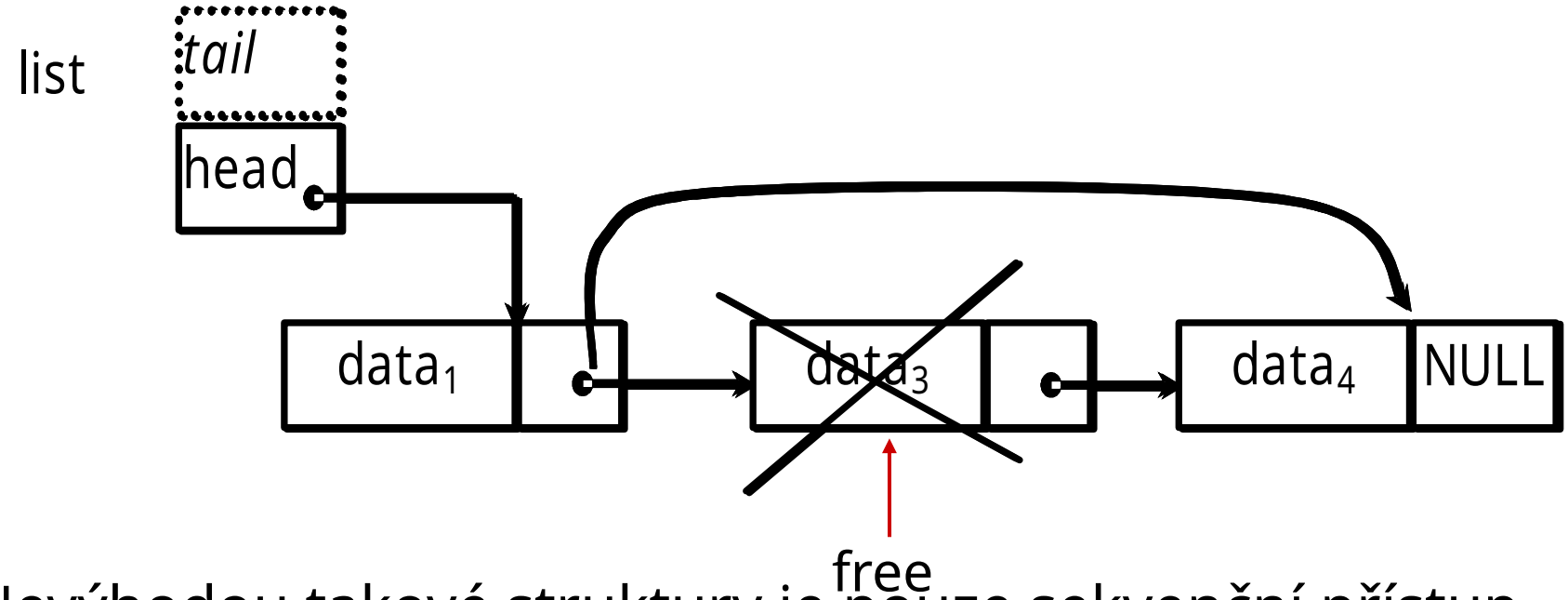
Obecné vlastnosti dynamických datových struktur

- vložení nové položky



Obecné vlastnosti dynamických datových struktur

- odstranění položky



Nevýhodou takové struktury je pouze sekvenční přístup.

py Doporučení *py*: Nedovedu-li si to představit, tak si to nakreslím!

IZP 14 Abstraktní datové typy tvořené dynamickými strukturami

- abstraktní datový typ je určen:
  - množinou svých hodnot
  - množinou operací, které jsou nad ním definovány.

Základní abstraktní datové typy

- lineární seznamy (jednosměrně i obousměrně vázané)
- cyklické seznamy (jednosměrně i obousměrně vázané)
- zásobník (LIFO – last in, first out)
- fronta (FIFO – first in, first out)

Základní abstraktní datové typy

- stromy (binární, n-nární, B-stromy)

[on line, cit. 2019-11-1]

- obecný graf
- nehomogenní struktury – s různými typy prvků

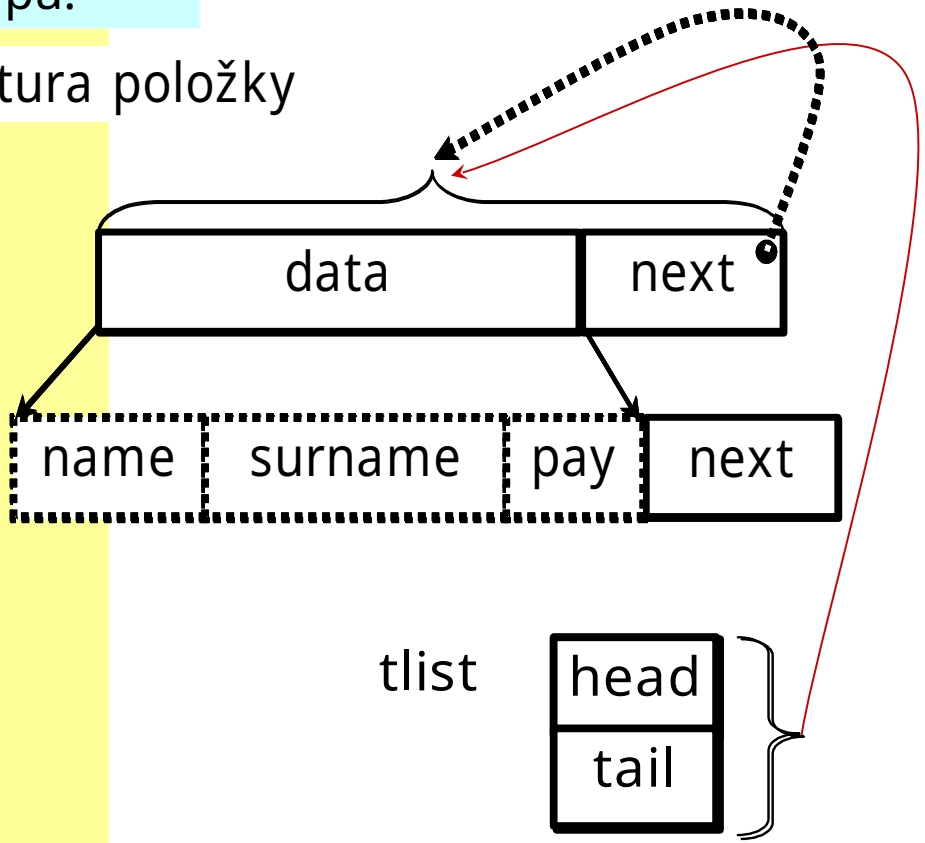
Lineární seznam (linked list)

- Základní operace se seznamem:
  - inicializace seznamu,
  - vložení prvku do seznamu,
  - zrušení prvku v seznamu,
  - průchod seznamem (provedení operace).

Lineární seznam

Příklad: operace se seznamem, definice typů.

```
typedef struct {
  char name[10];
  char surname[15];
  int pay;
} tdata;
typedef struct item titem;
struct item {
  tdata data;
  titem *next;
};
typedef struct {
  titem *head;
  titem *tail;
} tlist;
```

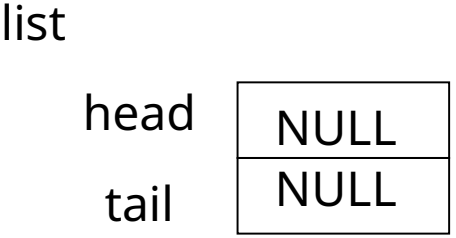


IZP 14

Lineární seznam

Příklad: inicializace seznamu.

```
void listInit(tlist *list)
{
  list->head = NULL;
  list->tail = NULL;
  return;
}
```



Poznámka: V dalším kódu budeme prozatím abstrahovat od přílišných kontrol vstupních hodnot předávaných funkcím.

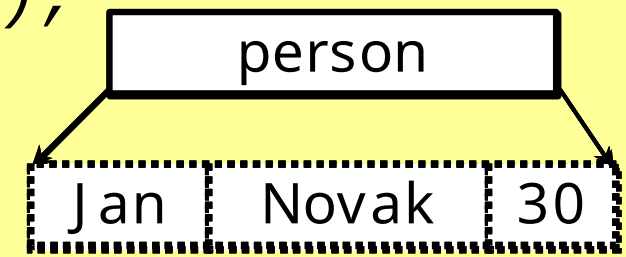
- načtení dat a vložení
  - čtení záznamu
  - vložení prvního prvku (položky) do seznamu

Poznámka: V dalším kódu budeme vytvářet seznam vkládáním prvku na začátek seznamu. Nebudeme používat ukazatel *tail*.

Lineární seznam

Příklad: čtení záznamu.

```
// Na konci souboru vrací počet načtených položek
// nebo EOF.
int read(tdata *person)
{
  int i = scanf("%9s%14s%d", person->name,
                person->surname, &person->pay);
  if (i != 3 && i != EOF)
  { // chybný vstup
    detectError("Chyba čtení\n");
  }
  return i;
}
```



Lineární seznam

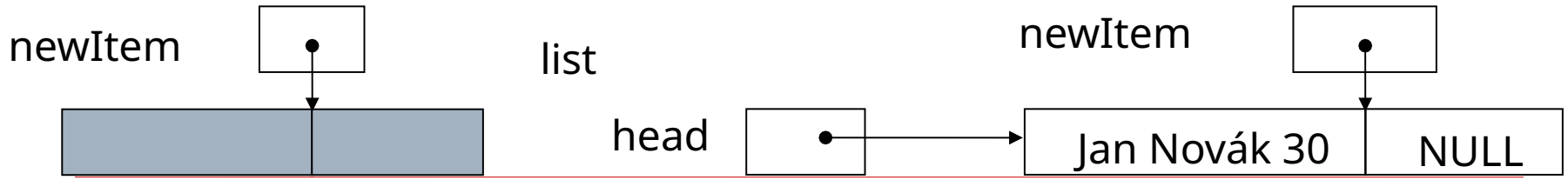
Příklad: vložení prvního prvku do seznamu


```
void insertFirst (tlist *list, tdata person)
{
  titem *newItem;
  if ((newItem = malloc(sizeof (titem)))== NULL)
    exit(EXIT_FAILURE);
  newItem->data = person;
  newItem->next = list->head;
  list->head = newItem;
  return;
}
```

Lineární seznam

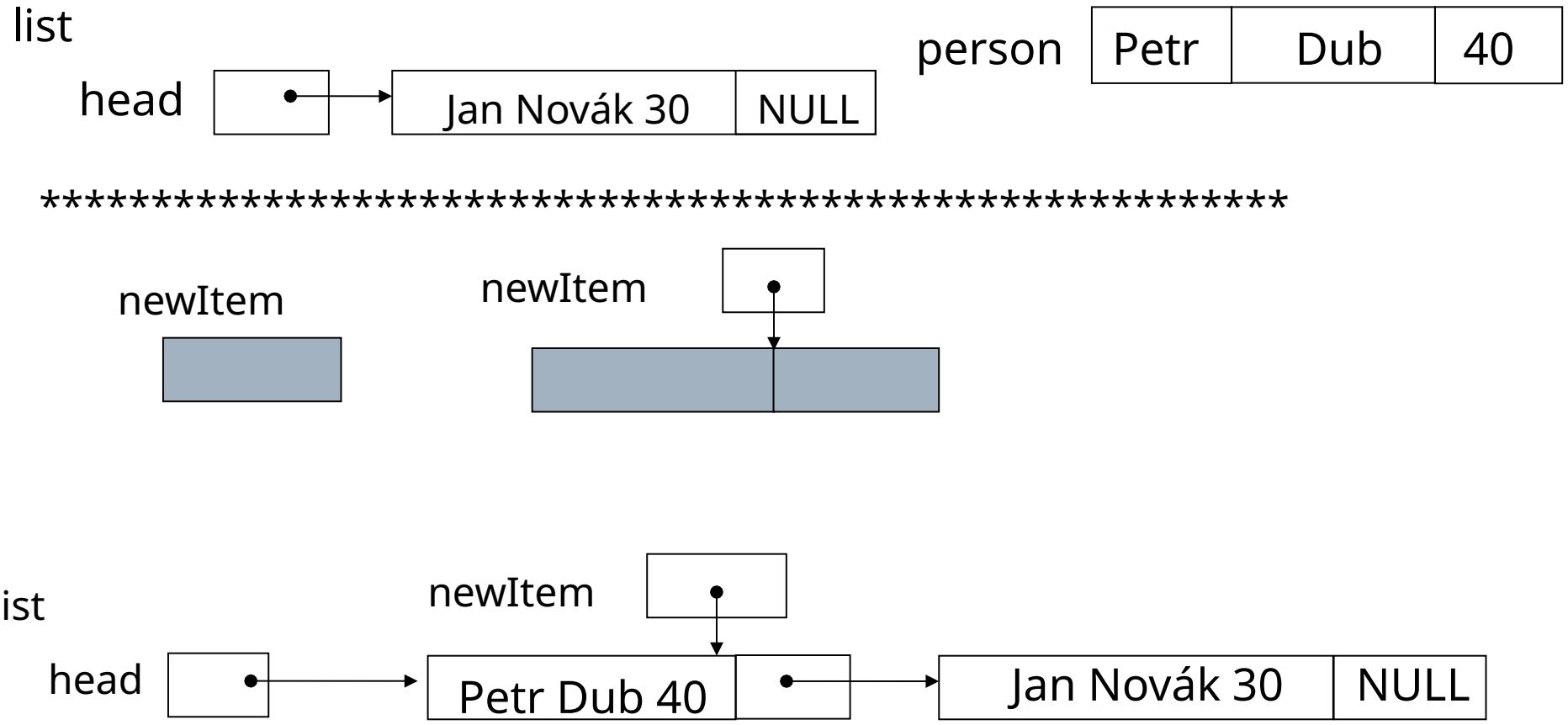
Příklad: vložení prvního prvku do seznamu - vysvětlení

```
    .
    .
if ((newItem = malloc (sizeof (titem))) == NULL)
     exit (EXIT_FAILURE);
  newItem->data = person;
  newItem->next = list->head;
  list->head = newItem;
  return;
}
```



Lineární seznam

Příklad: vložení dalšího prvku do seznamu (na první pozici)



Lineární seznam

Příklad: načtení dat do seznamu.

```
void readList (tlist *list)
{
  tdata tmp;
  while (read (&tmp) != EOF)
    insertFirst (list, tmp);
  return;
}
```

Lineární seznam

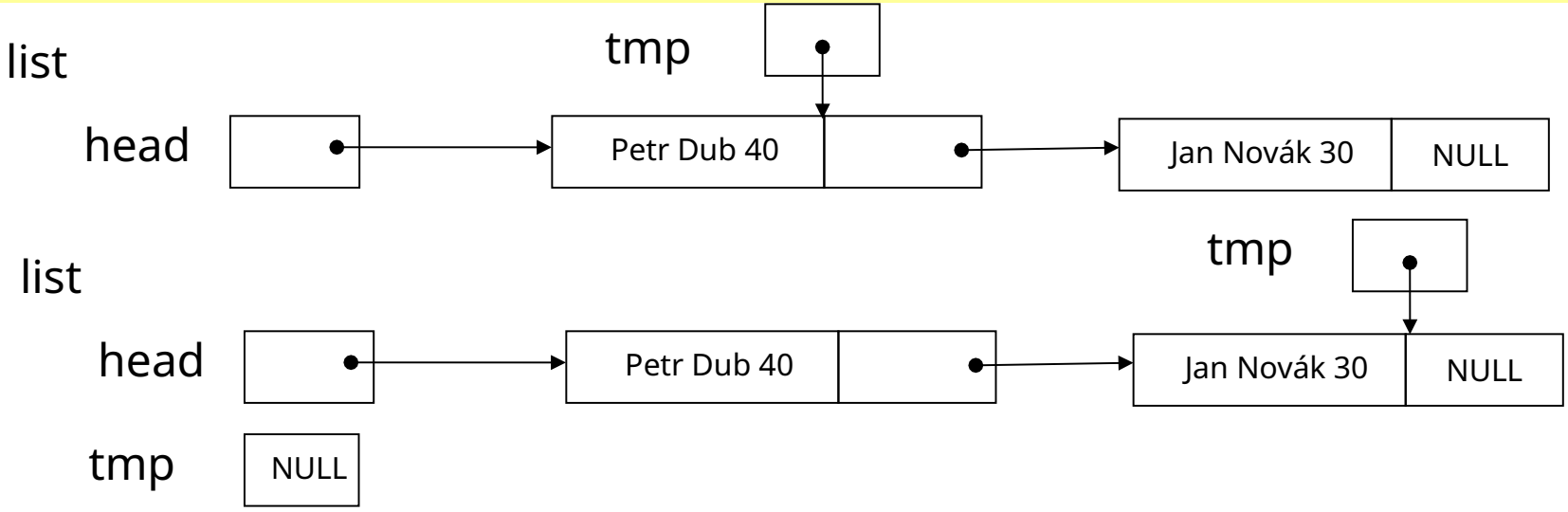
Příklad: průchod seznamem (tisk).

```
void writeList (const tlist *list)
{
  for (titem *tmp = list->head; tmp != NULL;
       tmp = tmp->next)
  {
    printf ("%s %s %d\n ", tmp->data.name,
            tmp->data.surname, tmp->data.pay);
  }
  return;
}
```

Lineární seznam

Příklad: průchod seznamem (tisk) - vysvětlení.

```
void writeList (const tlist *list)
{
  for (titem *tmp = list->head; tmp != NULL; tmp = tmp->next)
  { printf ("%s %s %d\n ", tmp->data.name, tmp->data.surname,
            tmp->data.pay); }
  return;
}
```

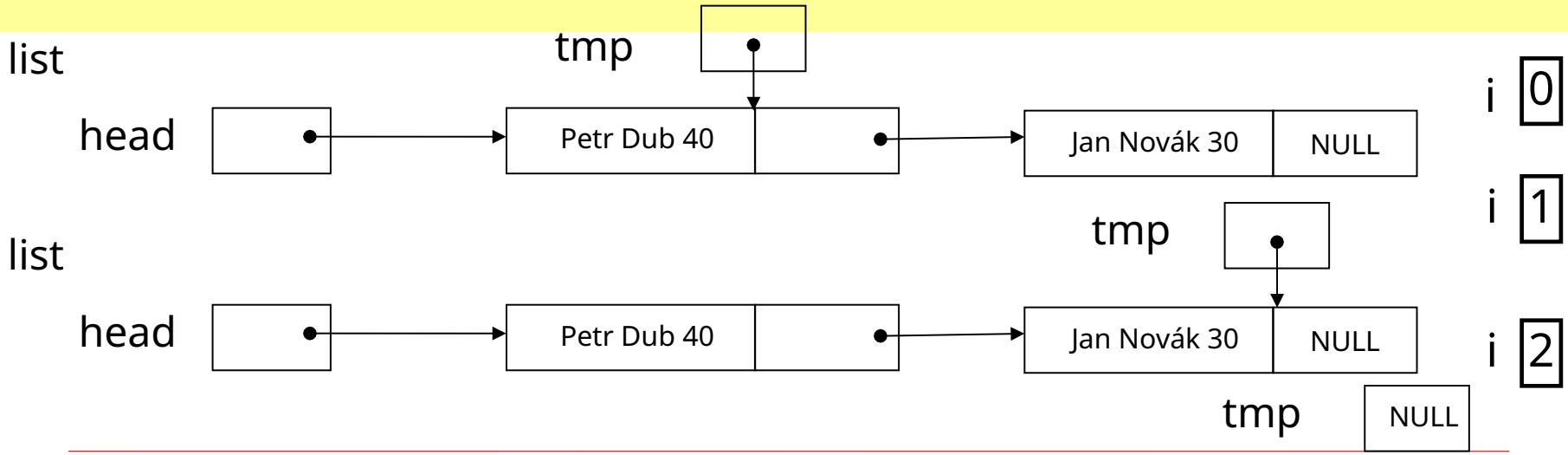


Lineární seznam

Příklad: počet prvků seznamu.

Poznámka: počet prvků lze ukládat i do struktury `tlist`.

```
int countItemList (const tlist *list)
{
  int i=0;
  for (titem *tmp=list->head; tmp != NULL; tmp=tmp->next)
    i++;
  return i;
}
```



Lineární seznam

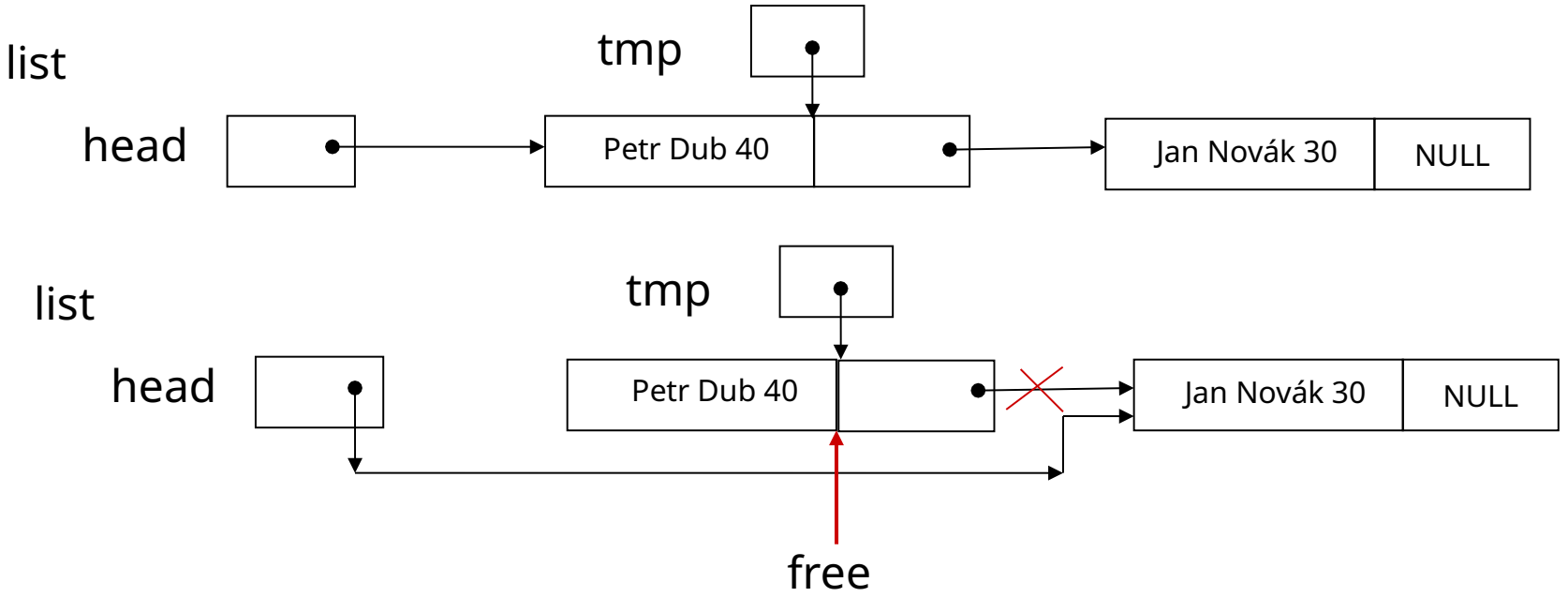
Příklad: zrušení seznamu.

```
void deleteList (tlist *list)
{
    while (list->head != NULL){
    titem *tmp;
    tmp = list->head;
    list->head =list->head->next;
    free(tmp);
  }
}
```

Lineární seznam

Příklad: zrušení seznamu - vysvětlení.

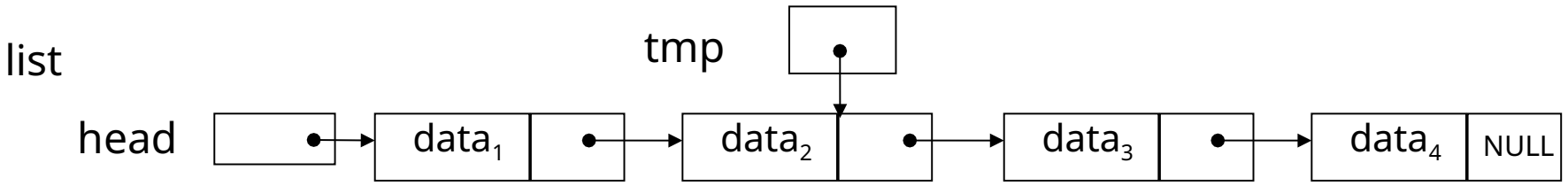
```
.
tmp = list->head;
list->head =list->head->next;
free(tmp);
.
```



Lineární seznam

- Vkládání prvku do seznamu na konkrétní místo:
  - za vybraný prvek,
  - před první prvek (funkce pro vytvoření seznamu),

Proč nejde snadno vkládat před jiný než první prvek? (obdobný problém - rušení)

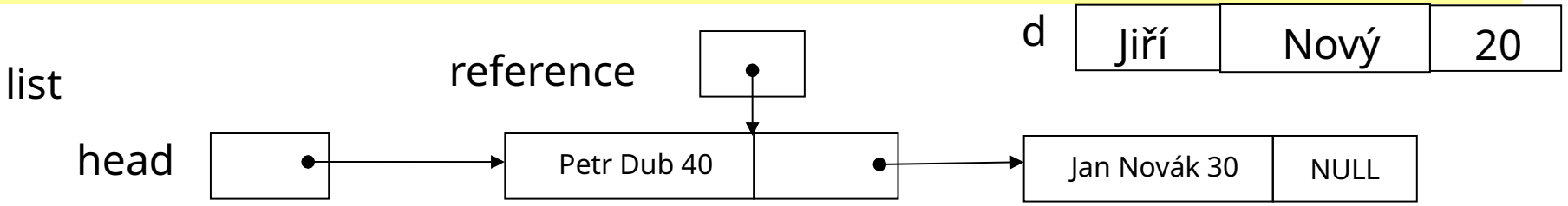


Lineární seznam

Příklad: vložení prvku do seznamu za vybraný prvek.

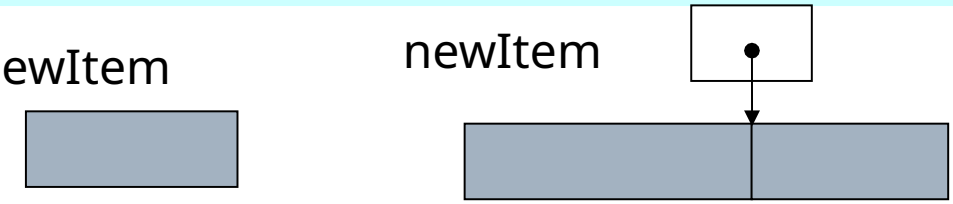
```
void insertBehind(titem *reference, tdata d)
{
  if (reference == NULL) exit(EXIT_FAILURE);
  titem *newItem;
  if ((newItem = malloc(sizeof(titem))) == NULL)
    exit(EXIT_FAILURE);
  newItem->data = d;

  titem *tmp = reference->next;
  reference->next = newItem;
  newItem->next = tmp;
}
```

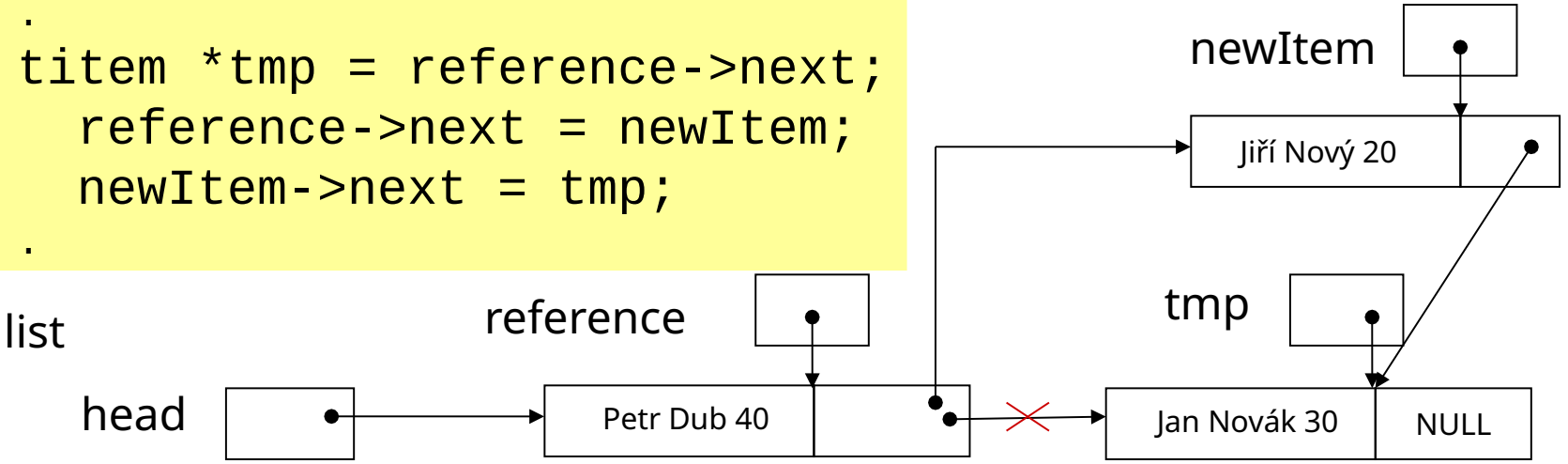


Lineární seznam

Příklad: vložení prvku do seznamu (za vybraný prvek) - vysvětlení



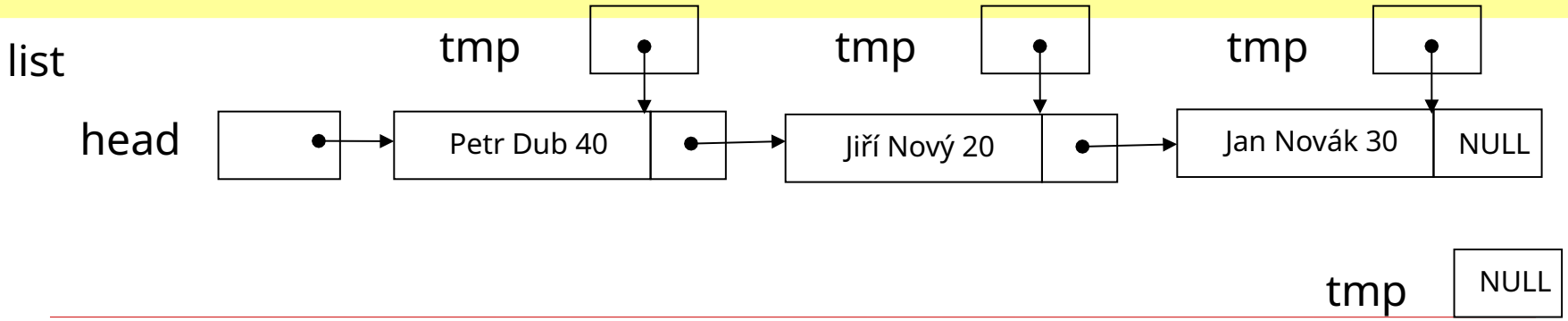

```
.
titem *tmp = reference->next;
  reference->next = newItem;
  newItem->next = tmp;
.
```



Lineární seznam

Příklad: vyhledání prvku v seznamu.

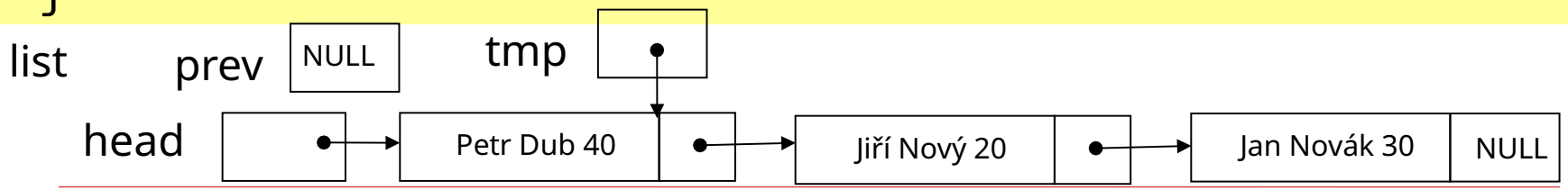
```
// Vrací ukazatel na nalezený prvek nebo NULL
titem *searchList(const tlist *list, tdata d)
{
  titem *tmp = list->head;
  while (tmp != NULL && !isEqual(tmp->data, d))
    tmp = tmp->next;
  return tmp;
}
```



Lineární seznam

Příklad: odstranění prvku ze seznamu.

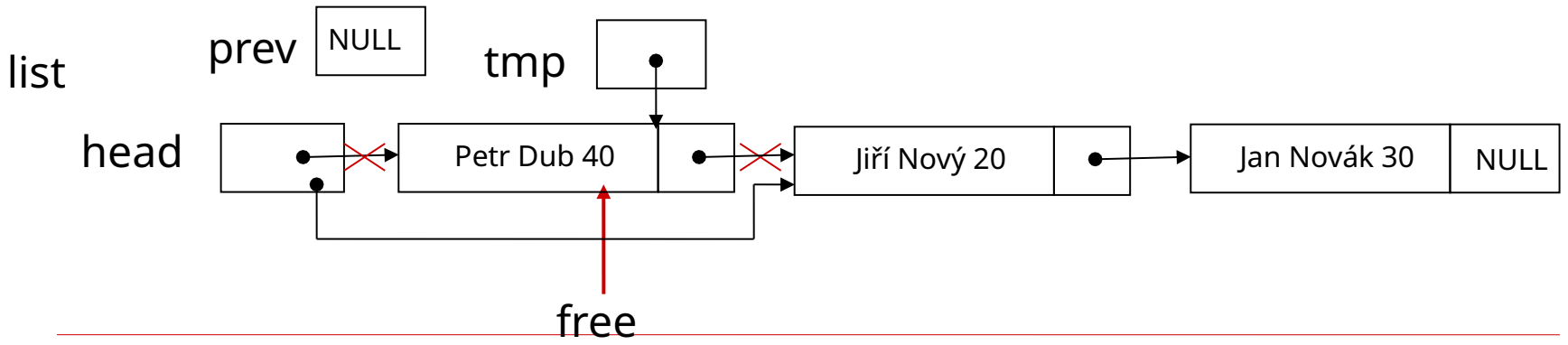
```
void deleteItem(tlist *list, tdata d)
{
  if (list == NULL || list->head == NULL)
    //  vypsat hlášení o chybě
    exit(EXIT_FAILURE);
  titem *tmp = list->head;
  titem *prev = NULL;
  while (tmp != NULL && !isEqual(tmp->data, d))
  { // hledáme prvek
    prev = tmp;
    tmp = tmp->next;
  }
```



Lineární seznam

Příklad: odstranění prvku ze seznamu - pokračování.

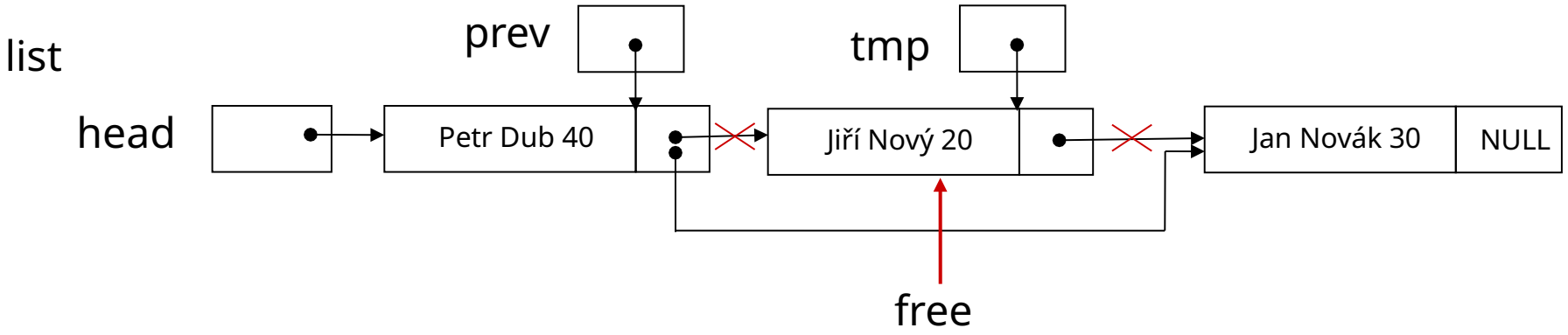
```
if (tmp == NULL) return;
// seznam byl prázdný nebo se hledaný prvek
// v seznamu nenašel
if (tmp == list->head)
{ // hledaným prvkem je první prvek
list->head = tmp->next;
}
```



Lineární seznam

Příklad: odstranění prvku ze seznamu - pokračování.

```
else
  { // hledaným prvkem je prostřední nebo
    // poslední prvek
    prev->next = tmp->next;
  }
  free(tmp);
}
```



Lineární seznam

Příklad: volání jednotlivých funkcí.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
  char name[10];
  char surname[15];
  int pay;
} tdata;
typedef struct item titem;
struct item {
  tdata data;
  titem *next;
};
typedef struct {
  titem *head;
  titem *tail;
} tlist;
```

Lineární seznam

Příklad: volání jednotlivých funkcí - pokračování.

```
int main (void)
{
  tlist list;
  listInit (&list);
  readList (&list);
  writeList (&list);

  printf ("\n polozek je: %d \n ",countItemList(&list));
  tdata temp1;
  read(&temp1);
  titem *tmp;
  tmp = searchList(&list, temp1);
  // operace s nalezenou položkou, pokud byla nalezena
```

Lineární seznam

Příklad: volání jednotlivých funkcí - pokračování.

```
  read(&temp1);
  insertBehind(tmp, temp1);
  writeList (&list);

  deleteItem(&list,temp1);
  writeList (&list);

  deleteList (&list);
  if (list.head == NULL)
    printf ("je prazdny \n ");

  return 0;
}
```

Lineární seznam

- Seznam se musí inicializovat.
- Seznam je zadán ukazatelem na první/poslední prvek.
- Aktuální stav seznamu bývá někdy určen ukazatelem na prvek, se kterým se právě pracuje (aktivní prvek, vybraný prvek).
- Operace nutné pro práci se seznamem (lineárním jednosměrným):
  - vložit prvek na začátek seznamu,
  - vložit prvek za vybraný prvek,
  - zrušit první prvek,
  - zrušit prvek za vybraným prvkem,
  - získat hodnotu (data) prvku.

Lineární seznam

- Další operace:
  - spojení dvou seznamů,
  - rozdělení seznamu na dva seznamy,
  - vytvoření kopie seznamu,
  - relace ekvivalence nad dvěma seznamy podle hodnot klíčů odpovídajících položek (princip této operace je obdobný jako princip relace nad dvěma textovými řetězci),
  - test, zda je seznam uspořádaný podle určité relace uspořádání nad zadaným klíčem,
  - uspořádání (seřazení) položek seznamu podle relace uspořádání nad určitou složkou položky (podle klíče).

Lineární seznam

- používá se jako pomocná datová struktura

Příklad: koncept indexsekvenčního pole

- pro implementaci zásobníku a fronty
- nepoužívá se, když by režie pro uložení ukazatelů byla větší než je únosné

Příklad: lineární seznam znaků

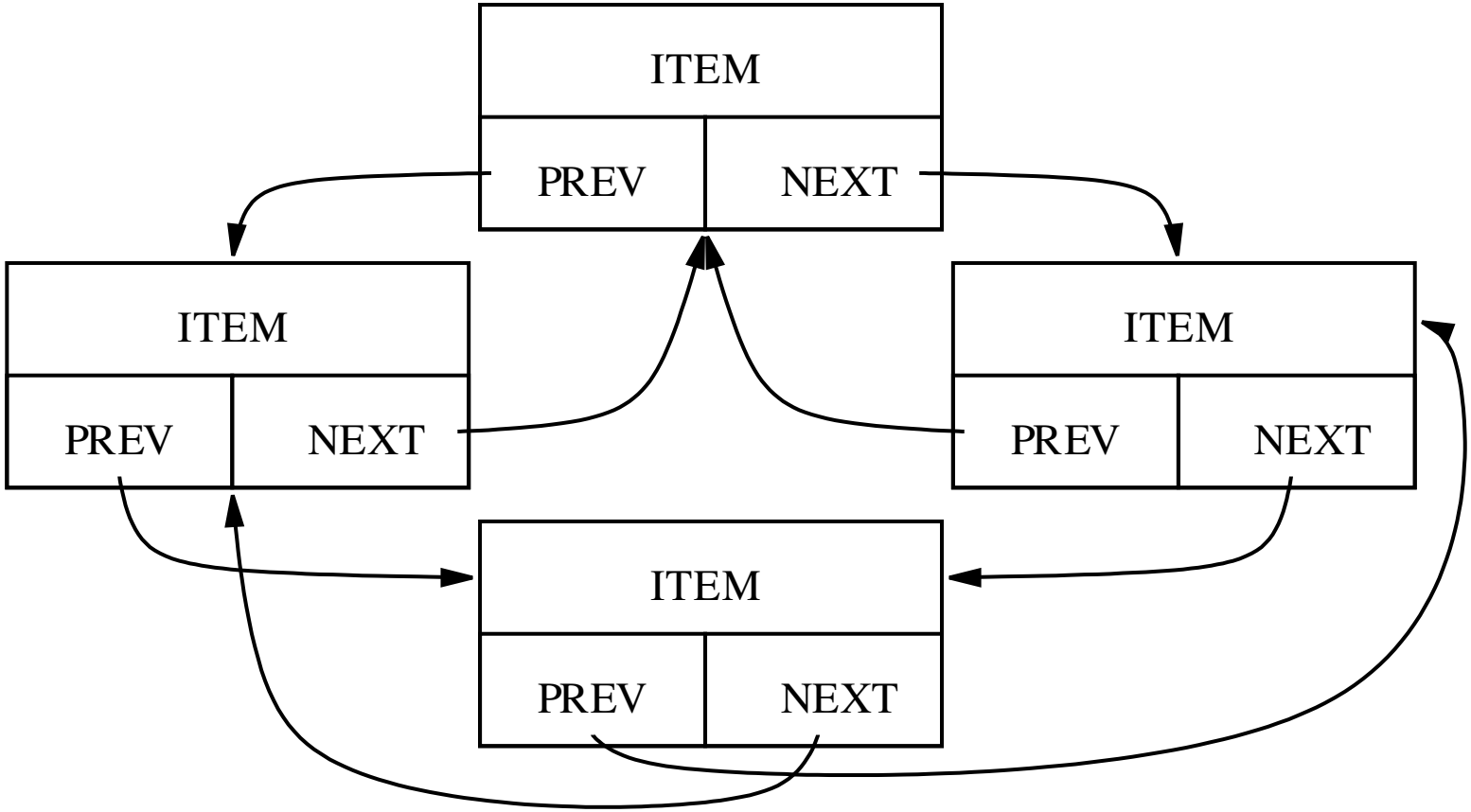
- zvažuje se
  - rychlost přístupu
  - paměťové nároky
  - operace

IZP 14

Obousměrně vázané a cyklické seznamy

- sousední prvky jsou zřetězeny obousměrně
- cyklický seznam **h** první a poslední jsou vzájemně propojeny
- aktuální prvek **h** prvek, se kterým právě pracujeme
- nevýhoda - větší nároky na paměť

Obousměrně vázaný cyklický seznam

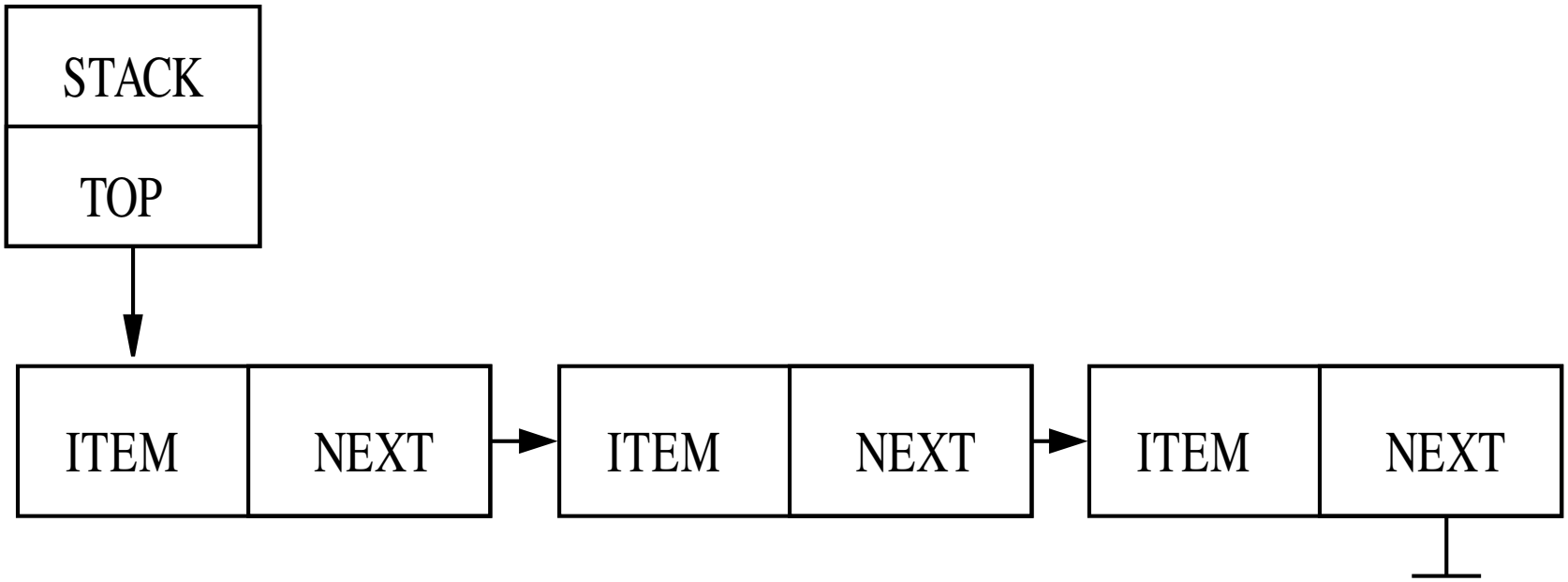


Zásobník (stack)

LIFO - Last In, First Out

- první prvek – vrchol zásobníku
- Operace:
  - Push – vlož nový prvek na vrchol zásobníku
  - Pop – odstraň prvek z vrcholu zásobníku a (Top) vrať jeho hodnotu
- vhodné použití - zpracování dat v opačném pořadí, než jsme je načetli
  - iterační algoritmy nahrazující rekurzi
  - implicitní zásobník - oblast paměti programu určená pro volání funkcí

Zásobník (stack)

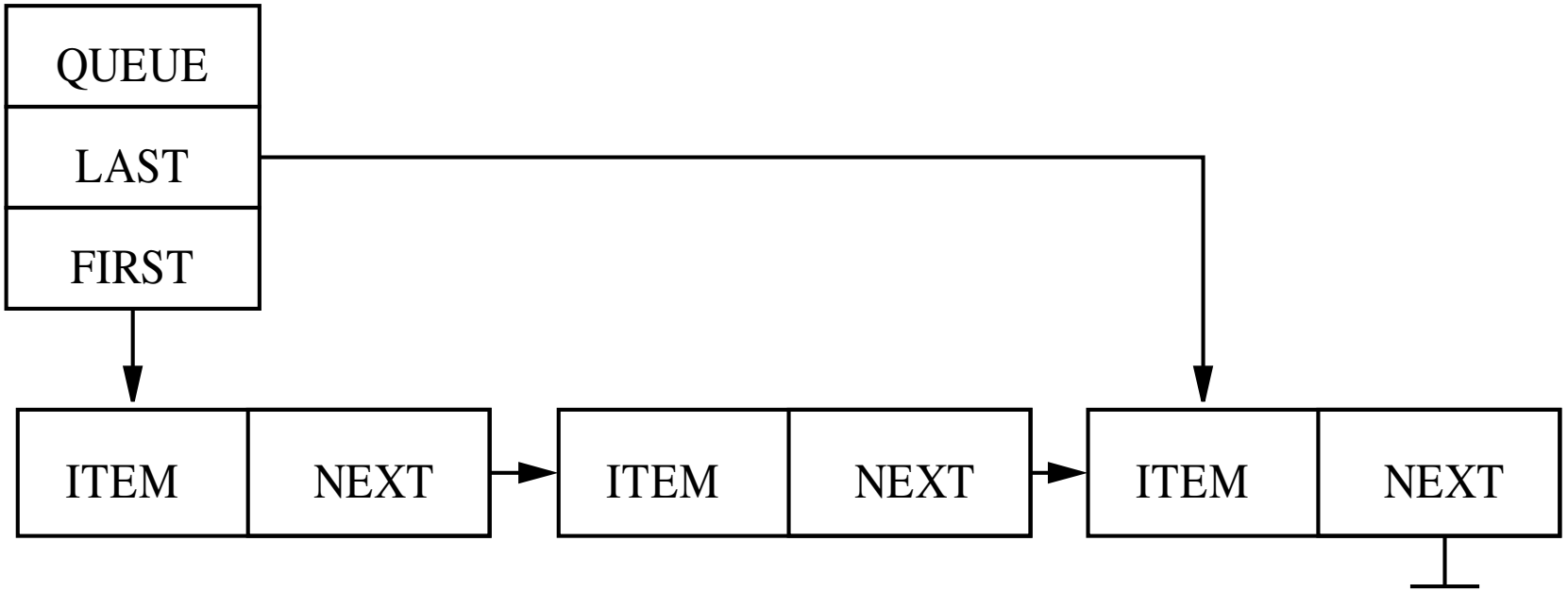


Fronta (Queue)

FIFO - First In, First Out

- fronta má počátek a konec
- Operace:
  - vložení prvku na konec fronty (odpovídá počátečnímu prvku seznamu)
  - výběr prvku ze začátku fronty (odpovídá koncovému prvku seznamu)
- vhodné použití – úlohy, kde potřebujeme zachovat pořadí příchozích prvků
  - plynulé zpracování dat programem - buffer

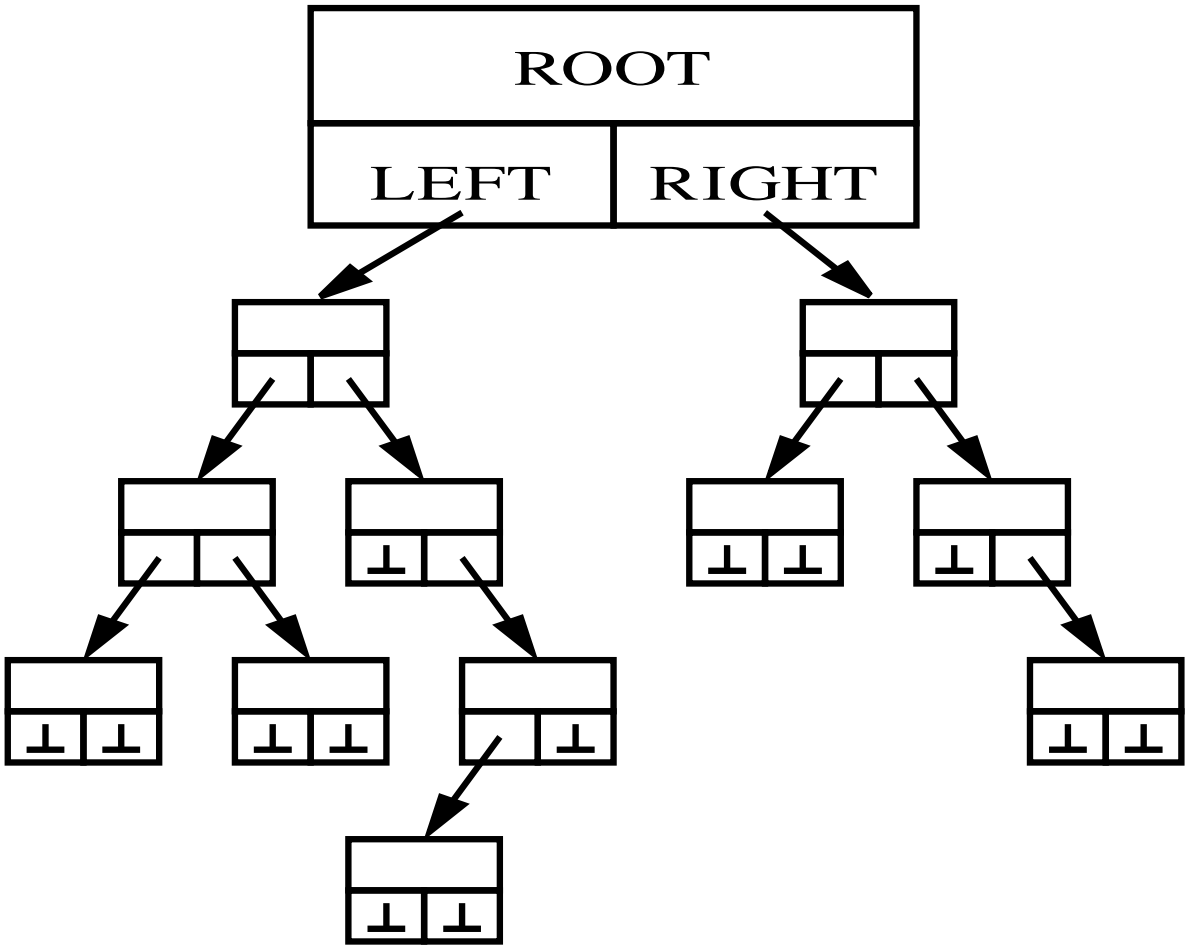
Fronta (FIFO)



Nelineární struktura - strom

- prvek (uzel) obsahuje dva ukazatele – na levý a pravý podstrom, typy prvků:
  - Kořenový prvek (root) – prvek z něhož se lze dostat ke kterémukoli jinému prvku stromu
  - Listy – koncové prvky, které neobsahují žádný podstrom
- binární strom, n-ární strom
- vyvážený strom

Nelineární struktura - strom



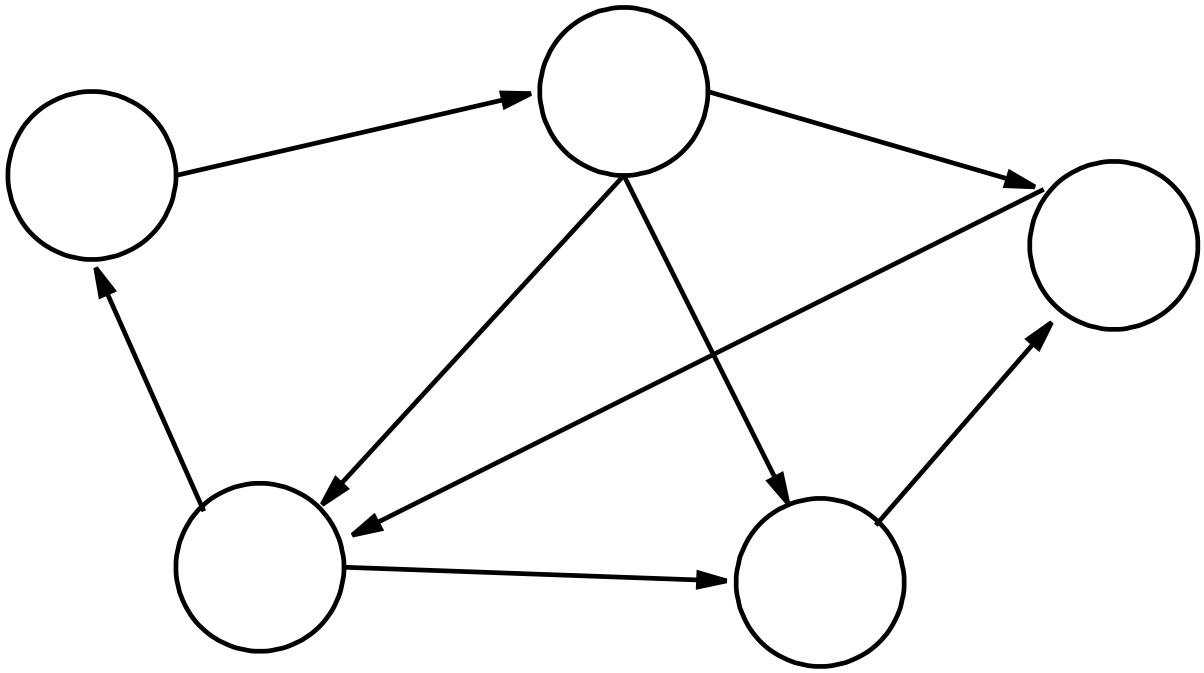
Nelineární struktura - strom

- vhodné použití
  - vyhledávání dat
  - překladače počítačových jazyků
  - různé analyzátory zdrojového kódu
  - zpracovávání XML, nebo HTML dokumentů
  - zkoumáním sémantické správnosti navržených programů

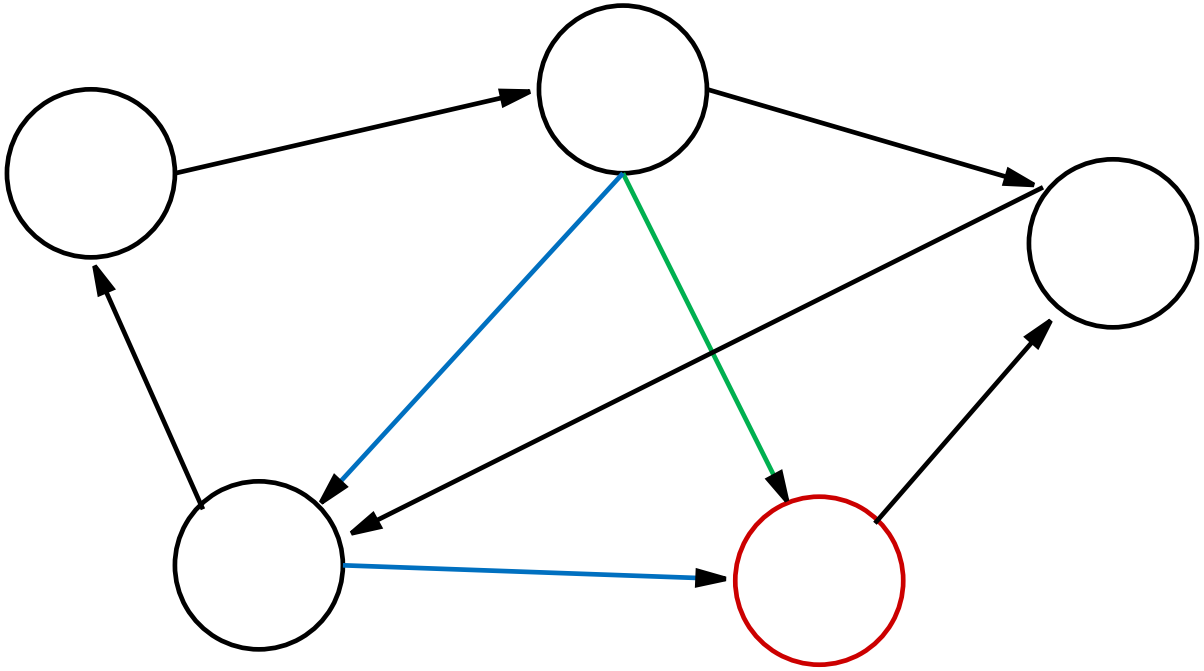
Nehomogenní struktury – obecný graf

- Jakákoliv složitá struktura navzájem svázaných prvků – uzlů, libovolně propojenými orientovanými nebo neorientovanými cestami – hranami.
- Prvky mohou být různých typů a mohou vstupovat do libovolných nehierarchických vztahů.
- Do jednotlivých uzlů grafu se můžeme dostat různými cestami.

Nehomogenní struktury – obecný graf



Nehomogenní struktury – obecný graf



Nehomogenní struktury – obecný graf

- vhodné použití
  - grafické aplikace
  - počítačové hry
  - počítačová simulace
  - síťové aplikace

Dynamické datové struktury

IZP 14

Kontrolní otázky

1. Jaké jsou základní výhody a nevýhody dynamických datových struktur?
2. Popište princip abstraktního datového typu seznam. Uveďte příklady reálného využití.
3. Popište princip abstraktního datového typu zásobník. Uveďte příklady reálného využití.
4. Popište princip abstraktního datového typu fronta. Uveďte příklady reálného využití.
5. Popište princip abstraktního datového typu strom. Uveďte příklady reálného využití.

IZP 14

Úkoly k procvičení

1. Implementujte aplikaci, která bude uchovávat informace o studijních výsledcích studentů pomocí abstraktního datového typu seznam.