

						Úvod	Spojový seznam	Start/End	Vložení/odebrání prvku	Kruhový spojový seznam	Obousměrný seznam
<h2>Spojové struktury</h2> <p>Jan Faigl</p> <p>Katedra počítačů Fakulta elektrotechnická České vysoké učení technické v Praze</p> <p>Přednáška 08 BAB36PRGA – Programování v C</p>	<h3>Přehled témat</h3> <ul style="list-style-type: none"> ■ Část 1 – Spojové struktury Spojové struktury Spojový seznam Spojový seznam s odkazem na konec seznamu Vložení/odebrání prvku Kruhový spojový seznam Obousměrný seznam <ul style="list-style-type: none"> ■ Část 2 – Zadání 7. domácího úkolu (HW7) 	<p>Část I</p> <p>Část 1 – Spojové struktury</p>									
<p>Jan Faigl, 2024</p> <p>BAB36PRGA – Přednáška 08: Spojové struktury</p> <p>1 / 50</p> <p>Úvod Spojový seznam Start/End Vložení/odebrání prvku Kruhový spojový seznam Obousměrný seznam</p> <p>Kolekce prvků (položek)</p> <ul style="list-style-type: none"> ■ V programech je velmi běžný požadavek na uchování seznamu (množiny) prvků (proměnných/struktur). ■ Základní kolekce je pole. <i>Definované jménem typu a[], například double[].</i> <ul style="list-style-type: none"> ■ Jedná se o kolekci položek (proměnných) stejného typu. + Umožňuje jednoduchý přístup k polohám indexů prvků. <i>Položky jsou stejněho typu (velikosti), kompilátor tak může vytvořit kód, ve kterém se adresa prvku spočítá z indexu a velikosti prvku.</i> – Velikost pole je určena při vytvoření pole. ■ Velikost (maximální velikost) musí být známa v době vytváření. ■ Změna velikost není přímo možná. <i>Nutné nové vytvoření (allokace paměti), voláním realloc() může dojít k rozšíření, které závisí na aktuálním stavu paměti.</i> ■ Využití pouze malé části pole (s objemnými prvky) může být plýtváním paměti. ■ V případě řazení pole přesouváme jednotlivé položky pole. ■ Vložení prvku a vyjmoutí prvku využíváme kopirování (zachování souvislosti dat). <i>Kopirování objemných prvků lze případně řešit ukládáním ukazatelů.</i> 	<p>Jan Faigl, 2024</p> <p>BAB36PRGA – Přednáška 08: Spojové struktury</p> <p>2 / 50</p> <p>Úvod Spojový seznam Start/End Vložení/odebrání prvku Kruhový spojový seznam Obousměrný seznam</p> <p>Seznam – list</p> <ul style="list-style-type: none"> ■ Seznam (proměnných nebo objektů) patří mezi základní datové struktury. <i>Základní ADT – Abstract Data Type.</i> ■ Seznam zpravidla nabízí sadu základních operací: <ul style="list-style-type: none"> ■ Vložení prvku (insert); ■ Odebrání prvku (remove); ■ Vyhledání prvku (indexOf); ■ Aktuální počet prvků v seznamu (size). ■ Implementace seznamu může být založena na poli nebo spojové struktuře. <ul style="list-style-type: none"> ■ Pole <ul style="list-style-type: none"> ■ Indexování je velmi rychlé. ■ Vložení prvku na konkrétní pozici může být pomalé. ■ Spojové seznamy <ul style="list-style-type: none"> ■ Položky seznamu jsou sekvenčně propojeny, přímý náhodný přístup není jednoduše možný. ■ Vložení nebo odebrání prvku může být velmi rychlé. 	<p>Jan Faigl, 2024</p> <p>BAB36PRGA – Přednáška 08: Spojové struktury</p> <p>3 / 50</p> <p>Úvod Spojový seznam Start/End Vložení/odebrání prvku Kruhový spojový seznam Obousměrný seznam</p> <p>Spojové seznamy</p> <ul style="list-style-type: none"> ■ Datová struktura realizující seznam dynamické délky. ■ Každý prvek seznamu obsahuje: <ul style="list-style-type: none"> ■ Datovou část (hodnota proměnné / objekt / ukazatel na data); ■ Odkaz (ukazatel) na další prvek v seznamu. <i>NULL</i> v případě posledního prvku seznamu (zarážka). ■ První prvek seznamu se zpravidla označuje jako head nebo start. <i>Realizujeme jej jako ukazatel odkazující na první prvek seznamu.</i> <div style="text-align: center;"> </div>									
<p>Jan Faigl, 2024</p> <p>BAB36PRGA – Přednáška 08: Spojové struktury</p> <p>5 / 50</p> <p>Úvod Spojový seznam Start/End Vložení/odebrání prvku Kruhový spojový seznam Obousměrný seznam</p> <p>Základní operace se spojovým seznamem</p> <ul style="list-style-type: none"> ■ Vložení prvku: <ul style="list-style-type: none"> ■ Předchozí prvek odkazuje na nový prvek; ■ Nový prvek může odkazovat na předchozí prvek, který na něj odkazuje. <i>Tzv. obousměrný spojový seznam.</i> ■ Odebrání prvku: <ul style="list-style-type: none"> ■ Předchozí prvek aktualizuje hodnotu odkazu na následující prvek; ■ Předchozí prvek nově odkazuje na následující prvek, na který odkazoval odebraný prvek. ■ Základní implementaci spojového seznamu je jednosměrný spojový seznam. 	<p>Jan Faigl, 2024</p> <p>BAB36PRGA – Přednáška 08: Spojové struktury</p> <p>6 / 50</p> <p>Úvod Spojový seznam Start/End Vložení/odebrání prvku Kruhový spojový seznam Obousměrný seznam</p> <p>Jednosměrný spojový seznam</p> <ul style="list-style-type: none"> ■ Příklad spojového seznamu pro uložení číselných hodnot. ■ Přidání nové hodnoty 50 je přidání nového prvku na konec seznamu. ■ Odebrání prvku s hodnotou 79. <ol style="list-style-type: none"> 1. Nejdříve sekvenčně najdeme prvek s hodnotou 79. 2. Následně vyjmeme prvek s hodnotou a propojíme prvek 93 s prvkem 50. <i>Položku next prvku 93 nastavíme na hodnotu next odebraného prvku, tj. na následující prvek s hodnotou 50.</i> 	<p>Jan Faigl, 2024</p> <p>BAB36PRGA – Přednáška 08: Spojové struktury</p> <p>7 / 50</p> <p>Úvod Spojový seznam Start/End Vložení/odebrání prvku Kruhový spojový seznam Obousměrný seznam</p> <p>Spojový seznam</p> <ul style="list-style-type: none"> ■ Seznam tvoří struktura prvku s dvěma základními položkami: <ul style="list-style-type: none"> ■ Data prvku (může být ukazatel); ■ Odkaz (ukazatel) na další prvek. ■ Seznam je pak: <ol style="list-style-type: none"> 1. Ukazatel na první prvek head; 2. nebo vlastní struktura pro seznam. ■ Příklad struktur pro uložení spojového seznamu celých čísel. <i>Vhodné pro uložení dalších informací, počet prvků, poslední prvek.</i> <pre> 1 typedef struct entry { 2 int value; 3 struct entry *next; 4 } entry_t; 5 entry_t *head = NULL; </pre> <ul style="list-style-type: none"> ■ Pro jednoduchost prvky seznamu obsahují celé číslo. <i>Obecně mohou obsahovat libovolná data (ukazatel na strukturu).</i> 									

Přidání prvku – příklad

- Vytvoříme nový prvek (10) seznamu a uložíme odkaz v `head`.


```
head = myMalloc(sizeof(entry_t));
head->value = 10;
head->next = NULL;
```
- Další přidávání (13) propojením s aktuálně 1. prvkem.


```
entry_t *new_entry = myMalloc(sizeof(entry_t));
new_entry->value = 13;
new_entry->next = head;
```
- a aktualizací proměnné `head`.


```
head = new_entry;
```

- Stále máme přístup na všechny prvky přes `head` a `head->next`. [lec08/my_malloc.h](#), [lec08/my_malloc.c](#)
- Inicializace položek prvku je důležitá.**
 - Hodnota `head == NULL` indikuje prázdný seznam.
 - Hodnota `entry->next == NULL` indikuje poslední prvek seznamu.

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

12 / 50

Spojový seznam – size()

- Zjištění počtu prvků v seznamu vyžaduje projít seznam až k zarázce `NULL`.
- Poslední polohu je taková, pro kterou platí `next == NULL`, nebo je seznam prázdný a `head == NULL`.
- Proměnnou `cur` používáme jako „kurzor“ pro procházení seznamu.

```
int size(const entry_t *const head)
{
    const entry_t *end = head;
    while (end && end->next) { // 1st test list is not empty
        end = end->next;
    }
    assert(end); // do not allow calling back on empty list
    return end->value;
}
```

Použijeme konstantní ukazatel na konstantní proměnnou, neboť seznam pouze procházíme a nemodifikujeme. Z hlašky funkce je tak zřejmé, že vstupní strukturu ve funkci nemodifikujeme.

- Procházíme kompletní seznam (n prvků), abychom spočítali počet prvků seznamu.

Lineární složitost operace `size()` – $O(n)$.

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

15 / 50

Příklad – Spojový seznam celých čísel

```
entry_t *head;
head = NULL; // initialization is important

push(17, &head);
push(7, &head);
printf("List: ");
print(head);
push(5, &head);
printf("\nList size: %i\n", size(head));
printf("Last entry: %i\n", back(head));
printf("List: ");
print(head);
push(13, &head);
push(11, &head);
pop(&head);
printf("List:r");
print(head);
printf("\nPop until head is not empty\n");
while (head != NULL) {
    const int value = pop(&head);
    printf("Popped value %i\n", value);
}
printf("List size: %i\n", size(head));
printf("Last entry value %i\n", back(head));
printf("List: ");
print(head);

Cleanup using pop until head
is not empty
Popped value 13
Popped value 5
Popped value 17
List size: 0
lec08/linkedList-int.h
lec08/linkedList-int.c
lec08/demo-linkedList-int.c
```

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

18 / 50

Spojový seznam – push()

- Přidání prvků na začátek implementujeme ve funkci `push()`.
- Předáváme adresu, kde je uložen odkaz na start seznamu.

```
#include <stdlib.h>
void* myMalloc(size_t size)
{
    void *ret = malloc(size);
    if (!ret) {
        fprintf(stderr, "Malloc failed!\n");
        exit(-1)
    }
    return ret;
}

void push(int value, entry_t **head)
{
    entry_t *new_entry = myMalloc(sizeof(entry_t));
    if (*head == NULL) { // first entry in the list
        new_entry->next = NULL; // reset the next
    } else {
        new_entry->next = *head;
    }
    *head = new_entry; //update the head
}
```

Alternativně můžeme `push()` implementovat také jako `entry_t push(int value, entry_t *head)`.*

- Přidání prvků není závislé na počtu prvků v seznamu.

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

13 / 50

Spojový seznam – back()

- Vrácení hodnoty posledního prvku ze seznamu – `back()`.

```
int back(const entry_t *const head)
{
    const entry_t *end = head;
    while (end && end->next) { // 1st test list is not empty
        end = end->next;
    }
    assert(end); // do not allow calling back on empty list
    return end->value;
}
```

Kontrolou assert() vynucujeme, že při implementaci programu ladíme, že volání `back()` nebude provádět pro prázdný seznam. To musíme zajistit programově.

- Musíme projít všechny prvky seznamu.

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

16 / 50

Spojový seznam – zrychljení operací `size()` and `back()`

- Operace `size()` a `back()` procházejí kompletní seznam.
- Operaci `size()` můžeme urychlit udržováním aktuálního počtu prvků v seznamu.
 - Zavedeme datovou položku `int counter`.
 - Počet prvků inkrementujeme při každém přidání prvku a dekrementuje při každém odebrání prvku.
- Operaci `back()` můžeme urychlit proměnou odkazující na poslední prvek.
- Zavedeme strukturu pro vlastní spojový seznam s položkami `head`, `counter`, and `tail`.

```
typedef struct {
    entry_t *head;
    entry_t *tail;
    int counter;
} linked_list_t;

V případě přidání prvků na začátek, aktualizujeme tail pouze pokud byl seznam dosud prázdný.
Můžeme tail aktualizovat při přidání prvků na konec nebo vymístit posledního prvku.
```

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

20 / 50

Spojový seznam – pop()

- Odebrání prvního prvku ze seznamu

Kdy použijeme `assert()` a kdy `myAssert()`?

```
int pop(entry_t **head)
{
    // linked list must be non-empty
    assert(head != NULL && *head != NULL);
    entry_t *prev_head = *head; // save the current head
    int ret = prev_head->value; // retrieve data from the current head
    *head = prev_head->next; // set to NULL if the last item is popped
    free(prev_head); // release memory of the popped entry
    return ret;
}
```

*Alternativně například také jako `int pop(entry_t *head)`, ale nenastaví `head` na `NULL` v případě výjmutí posledního prvku.*

- Odebrání prvků není závislé na počtu prvků v seznamu.

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

14 / 50

Spojový seznam – procházení seznamu

- Procházení seznamu demonstrujeme na funkci `print()`.

```
void print(const entry_t *const head)
{
    const entry_t *cur = head; // set the cursor to head
    while (cur != NULL) {
        printf("%i%s", cur->value, cur->next ? " " : "\n");
        cur = cur->next; // move in the linked list
    }
}
```

Použijeme konstantní ukazatel na konstantní proměnnou, neboť seznam pouze procházíme a nemodifikujeme.

Z hlašky funkce je zřejmé, že vstupní strukturu nemodifikujeme.

- Prvky seznamu tiskneme za sebou oddělené mezerou a poslední prvek je zakončen znakem nového řádku.

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

17 / 50

Spojový seznam – urychlený `size()`

- Samosvatná struktura pro seznam.
- Položky `head` a `counter`.
- `head` je ukazatel na `entry_t`.
- Ve funkci `size()` předpokládáme validní odkaz na seznam.
- Proto voláme `assert(list)`.
- Přímá inicializace `linked_list_t linked_list = { NULL, 0 }`;
- Do funkcií `push()` a `pop()` stačí předávat pouze ukazatel, proto použijeme proměnnou `list linked_list_t *list = &linked_list;`
- Inkrementujeme a dekrementujeme proměnnou `counter` ve funkciích `push()` a `pop()`.

```
void push(int data, linked_list_t *list)
{
    ...
    list->counter += 1;
    ...
    return ret;
}
```

Jan Faigl, 2024

BAB36PRGA – Přednáška 08: Spojové struktury

21 / 50

Úvod	Spojový seznam	Start/End	Vložení/odebrání prvku	Kruhový spojový seznam	Obousměrný seznam	Úvod	Spojový seznam	Start/End	Vložení/odebrání prvku	Kruhový spojový seznam	Obousměrný seznam	Úvod	Spojový seznam	Start/End	Vložení/odebrání prvku	Kruhový spojový seznam	Obousměrný seznam															
Spojový seznam – push() s odkazem na konec seznamu																																
1 void push(int value, linked_list_t *list) 2 { // add new entry at front 3 assert(list); 4 entry_t *new_entry = myMalloc(sizeof(entry_t)); 5 new_entry->value = value; // set data; exit is called if myMalloc fails 6 if (list->head) { // an entry already in the list 7 new_entry->next = list->head; 8 } else { //list is empty 9 new_entry->next = NULL; // reset the next 10 list->tail = new_entry; //1st entry is the tail 11 } 12 list->head = new_entry; //update the head 13 list->counter += 1; // keep counter up to date 14 }	Hodnotu ukazatele tail nastavujeme pouze pokud byl seznam prázdný, protože prvky přidáváme na začátek.					Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	22 / 50	Jan Faigl, 2024					Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	23 / 50	Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	24 / 50													
Spojový seznam – pop() s odkazem na konec seznamu																																
1 int pop(linked_list_t *list) 2 { 3 assert(list); 4 myAssert(list->head, __LINE__, __FILE__); // non-empty list 5 entry_t *prev_head = list->head; // save head 6 list->head = prev_head->next; 7 list->counter -= 1; // keep counter up to date 8 int ret = prev_head->value; 9 free(prev_head); // release the memory 10 if (list->head == NULL) { // end has been popped 11 list->tail = NULL; 12 } 13 return ret; Hodnotu proměnné tail nastavujeme pouze pokud byl odebrán poslední prvek, protože prvky oddebráme ze začátku. 14 }	myAssert() ■ Prí volání musí být odkaz na spojový seznam platný (nikoliv NULL). ■ assert() testujeme správnost volání, že jsme ve struktuře programu neudělali chybu. Po odladění můžeme test vypustit, např. NDEBUG . ■ myAssert() testuje, že data jsou za běhu programu správné. Pokud ne, ukončujeme program a reportujeme. V následujícím případě možně také zajistit programové použitím assert() , a podobně volání pop() , např. if (!is_empty(list)).					1 #ifndef __MY_ASSERT_H__ 2 #define __MY_ASSERT_H__ 3 #include <stdio.h> //fprintf() 4 #include <stdlib.h> //exit() and malloc() 5 #include <assert.h> 6 #define myAssert(x, line, file) \n7 if (!x) { \n8 fprintf(stderr, "myAssert fail, \n9 line: %d, file: %s\n", line, file); \n10 exit(-1); \n11 } \n12 #endif	Výpis chyby s číslem rádu a jménem zdrojového souboru pro rychlejší nalezení kontextu a případnou opravy.					Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	23 / 50	Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	24 / 50															
Spojový seznamu – pushEnd()																																
1 void pushEnd(int value, linked_list_t *list) 2 { 3 assert(list); 4 entry_t *new_entry = myMalloc(sizeof(entry_t)); 5 new_entry->value = value; // set data 6 new_entry->next = NULL; // set the next 7 if (list->tail == NULL) { //adding the 1st entry 8 list->head = list->tail = new_entry; 9 } else { 10 list->tail->next = new_entry; //update the current tail 11 list->tail = new_entry; 12 } 13 list->counter += 1;	■ Přidání prvku na konec seznamu.					Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	25 / 50	Jan Faigl, 2024	Bab36PRGA – Přednáška 08: Spojové struktury					Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	26 / 50	Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	27 / 50												
Spojový seznam – Vložení prvku do seznamu																																
■ Vložení do seznamu: ■ na začátek – modifikujeme proměnnou head (funkce push()); ■ na konec – modifikujeme proměnnou posledního prvku a nastavujeme nový konec tail (funkce pushEnd()); ■ obecně – potřebujeme prvek (entry), za který chceme nový prvek (new_entry) vložit. entry_t *new_entry = myMalloc(sizeof(entry_t)); new_entry->value = value; // nastavení hodnoty new_entry->next = entry->next; //propojení s nasledujicim entry->next = new_entry; //propojení entry	■ Nalezení prvku na pozici index . ■ Pokud je index větší než počet prvků v poli, návrat posledního prvku.					Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	29 / 50	Jan Faigl, 2024	Bab36PRGA – Přednáška 08: Spojové struktury					Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	30 / 50	Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	31 / 50												
Spojový seznam – getEntry()																																
■ Do seznamu můžeme chtít prvek vložit na konkrétní pozici, tj. podle indexu v seznamu. Případně můžeme také požadovat vložení podle hodnoty prvek, tj. vložit před prvek s příslušnou hodnotou. Např. vložením prvku vždy před první prvek, který je větší vytvoříme uspořádaný seznam – realizujeme tak řazení vkládáním (insert sort). ■ Nalezení první implementujeme „privátní“ (static) funkci getEntry() .	static entry_t* getEntry(int index, const linked_list_t *list) { // here, we assume index >= 0 entry_t *cur = list->head; int i = 0; while (i < index && cur != NULL && cur->next != NULL) { cur = cur->next; i += 1; } return cur; //return entry at the index or the last entry } Pokud je seznam prázdný vrátí NULL , tj. list->head == NULL . ■ Funkci getEntry() chceme používat privátně pouze v rámci jednoho modulu (linked_list.c). ■ Proto ji definujeme s modifikátorem static .					Viz lec08/linked_list.c	Jan Faigl, 2024					Jan Faigl, 2024	Bab36PRGA – Přednáška 08: Spojové struktury					Jan Faigl, 2024	Bab36PRGA – Přednáška 08: Spojové struktury													
Spojový seznam – back() s odkazem na konec seznamu																																
■ Proměnná tail je buď NULL nebo odkazuje na poslední prvek seznamu.																																
1 int back(const linked_list_t *const list) 2 { // const - we do not modify the linked list 3 // we do not allow to call back on empty list that has to be assured programmatically 4 assert(list && list->tail); 5 return list->tail->value; 6 }	myAssert() ■ assert() testuje správnost volání, že jsme ve struktuře programu neudělali chybu. Po odladění můžeme test vypustit, např. NDEBUG . ■ myAssert() testuje, že data jsou za běhu programu správné. Pokud ne, ukončujeme program a reportujeme. V následujícím případě možně také zajistit programové použitím assert() , a podobně volání pop() , např. if (!is_empty(list)).					1 #ifndef __MY_ASSERT_H__ 2 #define __MY_ASSERT_H__ 3 #include <stdio.h> //fprintf() 4 #include <stdlib.h> //exit() and malloc() 5 #include <assert.h> 6 #define myAssert(x, line, file) \n7 if (!x) { \n8 fprintf(stderr, "myAssert fail, \n9 line: %d, file: %s\n", line, file); \n10 exit(-1); \n11 } \n12 #endif	Výpis chyby s číslem rádu a jménem zdrojového souboru pro rychlejší nalezení kontextu a případnou opravy.					Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	23 / 50	Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	24 / 50	Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury	24 / 50												
Příklad použití																																
■ Příklad použití na seznam hodnot typu int . ■ Výstup programu																																
1 #include "linked_list.h" 2 \$ clang linked_list.c demo-linked_list.c && ./out 3 linked_list_t list = { NULL, NULL, 0 }; 4 linked_list_t *list = &list; 5 push(10, list); push(5, list); push(17, list); 6 push(7, list); pushEnd(21, list); 7 print(list); 8 printf("Pop 1st entry: %i\n", pop(list)); 9 printf("List: "); print(list); 10 printf("Back of the list: %i\n", back(list)); 11 printf("Pop from the end: %i\n", popEnd(list)); 12 printf("List: "); print(list); 13 printf("Back of the list: %i\n", back(list)); 14 printf("Pop from the end: %i\n", popEnd(list)); 15 printf("List: "); print(list); 16 free_list(list); // cleanup!!! 17 18 lec08/demo-linked_list.c	■ linked_list.c demo-linked_list.c 7 5 10 17 21 Pop 1st entry: 7 List: 5 10 17 21 Back of the list: 21 Pop from the end: 21 Lst: 5 10 17 21 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 559 560 561 562 563 564 565 566 567 568 569 569 570 571 572 573 574 575 576 577 578 579 579 580 581 582 583 584 585 586 587 587 588 589 589 590 591 592 593 594 595 596 597 598 599 599 600 601 602 603 604 605 606 607 608 609 609 610 611 612 613 614 615 616 617 617 618 619 619 620 621 622 623 624 625 626 627 628 629 629 630 631 632 633 634 635 636 637 638 639 639 640 641 642 643 644 645 646 647 647 648 649 649 650 651 652 653 654 655 656 657 658 659 659 660 661 662 663 664 665 666 667 667 668 669 669 670 671 672 673 674 675 676 677 677 678 679 679 680 681 682 683 684 685 686 687 687 688 689 689 690 691 692 693 694 695 696 697 697 698 699 699 700 701 702 703 704 705 706 707 708 709 709 710 711 712 713 714 715 716 717 717 718 719 719 720 721 722 723 724 725 726 727 727 728 729 729 730 731 732 733 734 735 736 737 737 738 739 739 740 741 742 743 744 745 746 747 747 748 749 749 750 751 752 753 754 755 756 757 757 758 759 759 760 761 762 763 764 765 766 767 767 768 769 769 770 771 772 773 774 775 776 777 777 778 779 779 780 781 782 783 784 785 786 787 787 788 789 789 790 791 792 793 794 795 796 797 797 798 799 799 800 801 802 803 804 805 806 807 808 809 809 810 811 812 813 814 815 816 817 817 818 819 819 820 821 822 823 824 825 826 827 827 828 829 829 830 831 832 833 834 835 836 837 837 838 839 839 840 841 842 843 844 845 846 847 847 848 849 849 850 851 852 853 854 855 856 857 857 858 859 859 860 861 862 863 864 865 866 867 867 868 869 869 870 871 872 873 874 875 876 877 877 878 879 879 880 881 882 883 884 885 886 887 887 888 889 889 890 891 892 893 894 895 896 897 897 898 899 899 900 901 902 903 904 905 906 907 907 908<br																															

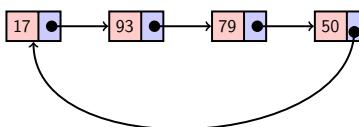
Úvod	Spojový seznam	Start/End	Vložení/odebrání prvku	Kruhový spojový seznam	Obousměrný seznam
Příklad vložení prvků do seznamu – insertAt()					
■ Příklad vložení do seznamu čísel	■ Výstup programu				
<pre>linked_list_t list = { NULL, NULL, 0 }; linked_list_t *lst = &list; push(10, lst); push(5, lst); push(17, lst); push(7, lst); push(21, lst); print(lst); insertAt(55, 2, lst); print(lst); insertAt(0, 0, lst); print(lst); insertAt(100, 10, lst); print(lst); free_list(lst); // cleanup!!! </pre>	\$ clang linked_list.c demo-insertat.c && ./a.out 21 7 17 5 10 21 7 55 17 5 10 0 7 55 17 5 10 0 7 55 17 5 10 100	32 / 50	lec08/demo-insertat.c	Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury

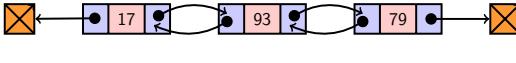
Úvod	Spojový seznam	Start/End	Vložení/odebrání prvku	Kruhový spojový seznam	Obousměrný seznam
Spojový seznam – removeAt(int index)					
■ Odebrání prvku na pozici <code>int index</code> a navázání seznamu.					
■ Pokud <code>index > size - 1</code> , smaže poslední prvek, viz <code>getEntry()</code> .					
■ Pro navázání seznamu potřebujeme prvek na pozici <code>index - 1</code> .					
<pre>void removeAt(int index, linked_list_t *list) { // check the arguments first if (index < 0 list == NULL list->head == NULL) { return; } if (index == 0) { pop(list); } else { entry_t *entry_prev = getEntry(index - 1, list); entry_t *entry = entry_prev->next; if (entry != NULL) { // handle connection entry_prev->next = entry->next->next; } if (entry == list->tail) { list->tail = entry_prev; } free(entry); list->count -= 1; } }</pre>	Složitost v nejlepším případě $O(n)$, protože nejdříve musíme prvek najít.	33 / 50	lec08/demo-removeat.c	Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury

Úvod	Spojový seznam	Start/End	Vložení/odebrání prvku	Kruhový spojový seznam	Obousměrný seznam
Příklad použití removeAt(int index)					
■ Výstup programu	■ Výstup programu				
<pre>void removeAndPrint(int index, linked_list_t *list) { entry_t *e = getAt(index, list); printf("Remove entry at %i (%i)\n", index, e->value); e->value = -1; removeAt(index, list); print(list); }</pre>	<pre>1 \$ clang linked_list.c demo-removeat.c && ./a.out 2 21 7 17 5 10 3 Remove entry at 3 (5) 4 21 7 17 10 5 Remove entry at 3 (10) 6 21 7 17 7 Remove entry at 0 (21) 8 7 17 </pre>	34 / 50	lec08/demo-removeat.c	Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury

Úvod	Spojový seznam	Start/End	Vložení/odebrání prvku	Kruhový spojový seznam	Obousměrný seznam
Příklad spojový seznamu textových řetězců					
■ Je nutné zvolit přístup pro alokaci hodnot textových řetězců.	Literály vs. proměnné.				
■ Příklad použití.	V lec08/linked_list-str.c je zvolena alokace paměti a kopírování hodnot .				
<pre>#include "linked_list-str.h" linked_list_t list = { NULL }; linked_list_t *lst = &list; push("FEE", lst); push("CTU", lst); push("PRG", lst); push("Lec08", lst); print(lst); for (int i = 0; i < 2; ++i) { char *str = pop(lst); printf("%d: %s\n", i+1, str); free(str); // Free is needed! } // demo-linked_list-str Lec08 PRG CTU 1. popped string Lec08 2. popped string PRG Popped value "CTU" String of the popped value is at address 0 x800240c10 Due to selected implementation the memory must be explicitly deallocated! </pre>	<pre>static entry_t* allocate_entry(const char *value) { entry_t *entry = myMalloc(sizeof(entry_t)); if (value) { entry->value = myMalloc(sizeof(char)*(strlen(value) + 1)); strcpy(entry->value, value); } else { entry->value = NULL; } return entry; } void push(const char *value, linked_list_t *list) { // add new entry at front assert(list); // debug assert!!! entry_t *new_entry = allocate_entry(value); if (list->head) { // an entry already in the list new_entry->next = list->head; list->head->prev = new_entry; } else { // list is empty new_entry->next = NULL; // reset the next } list->head = new_entry; //update the head }</pre>	35 / 50	lec08/demo-linked_list-str.c	Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury

Úvod	Spojový seznam	Start/End	Vložení/odebrání prvku	Kruhový spojový seznam	Obousměrný seznam
Spojový seznam s hodnotami typu textový řetězec					
■ Zajištění správné alokace a uvolnění paměti je v našem příkladě náročnější.					
■ V případě volání <code>pop()</code> je nutné následně paměť uvolnit.					
<pre>/* WARNING printf("Popped value \"%s\"\n", pop(lst)); */ /* Note, using this will cause memory leakage since we lost the address value to free the memory!!! */ char *str = pop(lst); printf("Popped value \"%s\"\n", str); free(str); /* str must be deallocated */ Při práci s dynamickou pamětí a datovými strukturami je nutné zvolit vhodný model (např. kopírování dat) a zajistit správné uvolnění paměti. </pre>	V C++ lze řešit například prostřednictvím „smart pointers“.	36 / 50	lec08/linked_list-str.c, lec08/doubly_linked_list-str.c	Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury

Úvod	Spojový seznam	Start/End	Vložení/odebrání prvku	Kruhový spojový seznam	Obousměrný seznam
Kruhový spojový seznam					
■ Položka <code>next</code> posledního prvku může odkazovat na první prvek.					
■ Tak vznikne kruhový spojový seznam.					
					

Úvod	Spojový seznam	Start/End	Vložení/odebrání prvku	Kruhový spojový seznam	Obousměrný seznam
Obousměrný spojový seznam					
■ Každý prvek obsahuje odkaz na následující a předešlé položky v seznamu, položky <code>prev</code> a <code>next</code> .					
■ První prvek má nastavenou položku <code>prev</code> na hodnotu <code>NULL</code> .					
■ Poslední prvek má <code>next</code> nastavenou na <code>NULL</code> .					
■ Příklad obousměrného seznamu celých čísel.					
					

Úvod	Spojový seznam	Start/End	Vložení/odebrání prvku	Kruhový spojový seznam	Obousměrný seznam
Příklad – Obousměrný spojový seznam					
■ Prvek listu má hodnotu (<code>value</code>) a dva odkazy (<code>prev</code> a <code>next</code>).	■ Alokaci prvku provedeme funkcí s inicializací na základní hodnoty.				
<pre>typedef struct dll_entry { int value; struct dll_entry *prev; struct dll_entry *next; } dll_entry_t;</pre>	<pre>dll_entry_t* allocate_dll_entry(int value) { dll_entry_t *new_entry = myMalloc(sizeof(dll_entry_t)); new_entry->value = value; new_entry->next = NULL; new_entry->prev = NULL; return new_entry; }</pre>	35 / 50	lec08/doubly_linked_list.h, lec08/doubly_linked_list.c	Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury

Úvod	Spojový seznam	Start/End	Vložení/odebrání prvku	Kruhový spojový seznam	Obousměrný seznam
Obousměrný spojový seznam – vložení prvku					
■ Vložení prvku před prvek <code>cur</code> :					
1. Napojení vloženého prvku do seznamu, hodnoty <code>prev</code> a <code>next</code> ;	1. Napojení vloženého prvku do seznamu, hodnoty <code>prev</code> a <code>next</code> ;				
2. Aktualizace <code>next</code> předešlého prvku k prvku <code>cur</code> ;	2. Aktualizace <code>next</code> předešlého prvku k prvku <code>cur</code> ;				
3. Aktualizace <code>prev</code> proměnného prvku <code>cur</code> .	3. Aktualizace <code>prev</code> proměnného prvku <code>cur</code> .				
<pre>void insert_dll(int value, dll_entry_t *cur) { assert(cur); dll_entry_t *new_entry = allocate_dll_entry(value); new_entry->next = cur; new_entry->prev = cur->prev; if (cur->prev != NULL) { cur->prev->next = new_entry; } cur->prev = new_entry; }</pre>	<pre>void insert_dll(int value, dll_entry_t *cur) { assert(cur); dll_entry_t *new_entry = allocate_dll_entry(value); new_entry->next = cur; new_entry->prev = cur->prev; if (cur->prev != NULL) { cur->prev->next = new_entry; } cur->prev = new_entry; }</pre>	38 / 50	lec08/doubly_linked_list.c	Jan Faigl, 2024	BAB36PRGA – Přednáška 08: Spojové struktury

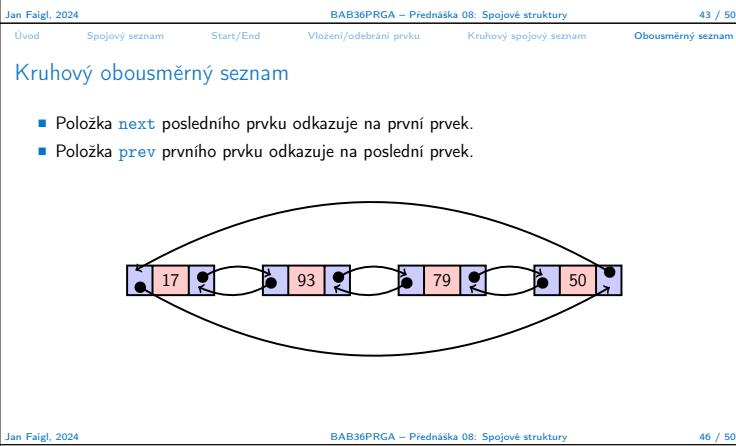
Obousměrný spojový seznam – přidání prvku na začátek push()

```

void push_dll(int value, doubly_linked_list_t *list)
{
    assert(list);
    dll_entry_t *new_entry = allocate_dll_entry(value);
    if (list->head) { // an entry already in the list
        new_entry->next = list->head; // connect new -> head
        list->head->prev = new_entry; // connect new <- head
    } else { //list is empty
        list->tail = new_entry;
    }
    list->head = new_entry; //update the head
}

```

lec08/doubly_linked_list.c



Shrnutí přednášky

Obousměrný spojový seznam – tisk seznamu

```

void print_dll(const doubly_linked_list_t *list)
{
    if (list && list->head) {
        dll_entry_t *cur = list->head;
        while (cur) {
            printf("%i%s", cur->value, cur->next ? " " : "\n");
            cur = cur->next;
        }
    }
}

void printReverse(const doubly_linked_list_t *list)
{
    if (list && list->tail) {
        dll_entry_t *cur = list->tail;
        while (cur) {
            printf("%i%s", cur->value, cur->prev? " " : "\n");
            cur = cur->prev;
        }
    }
}

```

lec08/doubly_linked_list.c

Část II

Část 2 – Zadání 7. domácího úkolu (HW7)

Příklad použití

- Výstup programu

```

#include "doubly_linked_list.h"

doubly_linked_list_t list = { NULL, NULL };
doubly_linked_list_t *lst = &list;

push_dll(17, lst); push_dll(93, lst);
push_dll(79, lst); push_dll(11, lst);

printf("Regular print: ");
print_dll(lst);

printf("Revert print: ");
printReverse(lst);

free_dll(lst);

```

lec08/doubly_linked_list.c
lec08/demo-doubly_linked_list.c

Zadání 7. domácího úkolu HW7

Téma: Kruhová fronta v poli

Povinné zadání: 3b; Volitelné zadání: 4b; Bonusové zadání: není

- **Motivace:** Práce s pamětí a datovými strukturami.
- **Cíl:** Prohloubit si znalost paměťové reprezentace a dynamické alokace paměti s uvoľňováním.
- **Zadání:** <https://cw.fel.cvut.cz/wiki/courses/bab36prga/hw/hw7>
 - Implementace kruhové fronty s využitím predalokovaného pole pro vkládané prvky.
 - Volitelné zadání rozšiřuje úlohu o dynamické zvětšování a zmenšování kapacity fronty podle aktuálních požadavků na pocet vkládaných/odebiraných prvků.
- **Termín odevzdání:** 27.04.2024, 23:59:59 PDT.

Diskutovaná témata

- Spojové struktury
 - Jednosměrný spojový seznam;
 - Obousměrný spojový seznam;
 - Kruhový obousměrný spojový seznam.
- Implementace operací **push()**, **pop()**, **size()**, **back()**, **pushEnd()**, **popEnd()**, **insertAt()**, **getEntry()**, **getAt()**, **removeAt()**, **indexOf()**.
- Použití spojového seznamu pro dynamicky alokované hodnoty prvků seznamu.
- **Příště abstraktní datový typ (ADT).**

Část IV

Appendix

Kódovací příklad – Dynamická knihovna

Kódovací příklad – Dynamická knihovna

- Knihovna s implementací (spojového) seznamu pro ukládání dynamicky alokovaných položek.
- Pro jednoduchosť při chybě dynamické alokace program ukončíme s výstupem na **stderr**.
- Implementujeme funkci **push()**, která neprídá hodnoty **NULL**. Návrhová volba!
- Prázdný seznam je indikován návratovou hodnotou **NULL** funkce **pop()**.
- Implementujeme nastavení funkce porovnání položek **setLess()**, kterou využijeme při vkládání položek do seznamu. Uspořádání položek dojde při volání **push()**. Implementujeme **insert sort**.
- Vytvořime dvě verze knihovny s/bez uspořádání položek, které budeme linkovat dynamicky.

```

1 #ifndef __LIST_H__
2 #define __LIST_H__

4 void* create(void); // void* - konkrétní typ považujeme za vnitřní záležitost knihovny.
5 void release(void* list); // argument list musí odpovídat typu z volání create()!
6
7 void setLess(void *list, void (*isLess)(const void *, const void *));
8
9 void push(void *list, void *value);
10 void* pop(void *list);

12#endif

```

list.h

Jan Faigl, 2024 BAB36PRGA – Přednáška 08: Spojové struktury

53 / 50

Kódovací příklad – list.c 1/2

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>

5 #include "list.h"

7 struct item {
8     void *value;
9     struct item *next;
10};

12 struct list {
13     struct item *root;
14     bool (*isLess)(const void *, const void *);
15};

17 enum { ERROR_MEM = 101 };

```

list.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 08: Spojové struktury

Kódovací příklad – list.c 2/3

```

19 void* create(void)
20 {
21     struct list *ret = malloc(sizeof(struct list));
22     if (!ret) {
23         fprintf(stderr, "ERROR: cannot allocate memory for list!\n");
24         exit(ERROR_MEM);
25     }
26     ret->root = NULL;
27     ret->isLess = NULL;
28     return ret;
29 }

30 struct list {
31     struct item *root;
32     bool (*isLess)(const void *, const void *);
33 };

```

list.c

- Při chybě alokace program končí voláním **exit()** s návratovou hodnotou **101**.
- Definici složených typů implementujeme pouze v souboru **list.c**, může se případně v budoucnu měnit, proto není struktu **item** součást rozhraní v **list.h**.

list.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 08: Spojové struktury

54 / 50

```

34 void* release(void* list)
35 {
36     if (list == NULL) {
37         struct list *list = (struct list*)list;
38         struct item *cur = list->root;
39         while (cur != NULL) {
40             struct item *t = cur;
41             cur = cur->next;
42             free(t->value); //Případě specifická funkce
43             free(t); // pro složený typ s ukazateli
44         }
45         free(list);
46         list = NULL;
47     }
48 }

```

- Funkce **setLess()** nastavuje ukazatel na funkci.
- Funkce **allocate_item()** nastavuje **next** na **NULL**.
- Položka **value** je adresa dynamicky alokované paměti.

list.c

Jan Faigl, 2024 BAB36PRGA – Přednáška 08: Spojové struktury

55 / 50

Kódovací příklad – list.c 3/3

```

6 static void pushLess(struct list *list, struct item *item)
7 {
8     if (!list->root || list->isLess(item->value, list->root->value)) {
9         item->next = list->root; // ok i pro root == NULL
10        list->root = item;
11        return;
12    }
13
14    struct item *prev = list->root;
15    struct item *cur = prev->next; // list->root není NULL
16    while (cur && cur->isLess(item->value, cur->value)) {
17        prev = cur;
18        cur = cur->next;
19    }
20    item->next = cur; //item bude poslední if cur == NULL
21    prev->next = item;
22}

```

- Privátní funkce **pushLess()** v rámci **list.c**.
- Funkce využívá nastavenou funkci **list->isLess**.
- Funkce vkládá **item** před první položku seznamu, která je větší (dle **isLess()**).

Např. clang -fWITH_LESS list.c

```

23 void push(void *list, void *value)
24 {
25     if (!list || !value) {
26         struct list *list = (struct list*)list;
27         if (!list || !value) {
28             return; // mení podporováno (ignorujeme)
29         }
30
31         struct item *n = allocate_item(value);
32         #ifdef WITH_LESS // isless pouze pokud WITH_LESS
33         if (list->isLess) {
34             pushLess(list, n);
35         } else {
36             #endif
37             if (list->root) {
38                 if (list->root) {
39                     if (list->root->value > value) {
40                         list->root = n;
41                     } else {
42                         int ret = EXIT_SUCCESS;
43                         char *line = NULL; // musí být NULL, alokace v getline()
44                         size_t linecap = 0; // délka alokované paměti v getline()
45                         ssize_t ln = 0; // délka načteného řetězce
46                         void *list = create(); // vytvoření seznamu z list.h
47                         setLess(list, isLess); // nastavení funkce isLess() demo.c
48                         while ((ln = getline(&line, linecap, stdin)) > 0) {
49                             if (line[ln-1] == '\n') { // ln vždy slepou ?
50                                 line[ln-1] = '\0'; // ignorujeme konec řádku
51                                 if (line[0] == '0') { // ukládání hodnot NULL
52                                     push(list, line);
53                                     linecap = 0;
54                                     line = NULL; // vynucujeme novou alokaci
55                                 }
56                             }
57                         }
58                         while ((line = pop(list))) {
59                             if (strcmp(a, b) < 0) { // lexicografické porovnání
60                                 free(line); // uvolňujeme Fádek z paměti
61                             }
62                         }
63                         release(list);
64                         return ret;
65                     }
66                 }
67             }
68         }
69     }
70 }

```

- Využívá implementaci **push()** vkládá hodnoty na začátek seznamu.
- Pokud je **list.c** komplikován s **WITH_LESS**, dochází k využití **list->isLess()**.
- Pokud je **list.c** komplikován s **WITH_LESS**, dochází k využití **list->isLess()**.

Jan Faigl, 2024 BAB36PRGA – Přednáška 08: Spojové struktury

56 / 50

Kódovací příklad – Volání rozhraní seznamu 1/3

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <list.h>

6 void push(void *list, void *value)
7 {
8     if (!list || !value) {
9         struct list *list = (struct list*)list;
10        if (!list || !value) {
11            return; // mení podporováno (ignorujeme)
12        }
13
14        struct item *n = allocate_item(value);
15        #ifdef WITH_LESS // isless pouze pokud WITH_LESS
16        if (list->isLess) {
17            pushLess(list, n);
18        } else {
19            #endif
20            if (list->root) {
21                if (list->root) {
22                    if (list->root->value > value) {
23                        list->root = n;
24                    } else {
25                        int ret = EXIT_SUCCESS;
26                        char *line = NULL; // musí být NULL, alokace v getline()
27                        size_t linecap = 0; // délka alokované paměti v getline()
28                        ssize_t ln = 0; // délka načteného řetězce
29                        void *list = create(); // vytvoření seznamu z list.h
30                        setLess(list, isLess); // nastavení funkce isLess() demo.c
31                        while ((ln = getline(&line, linecap, stdin)) > 0) {
32                            if (line[ln-1] == '\n') { // ln vždy slepou ?
33                                line[ln-1] = '\0'; // ignorujeme konec řádku
34                                if (line[0] == '0') { // ukládání hodnot NULL
35                                    push(list, line);
36                                    linecap = 0;
37                                    line = NULL; // vynucujeme novou alokaci
38                                }
39                            }
40                        }
41                        while ((line = pop(list))) {
42                            if (strcmp(a, b) < 0) { // lexicografické porovnání
43                                free(line); // uvolňujeme Fádek z paměti
44                            }
45                        }
46                        release(list);
47                        return ret;
48                    }
49                }
50            }
51        }
52    }
53 }

```

- Hodnoty textových řetězců jsou dynamicky alokované.
- Načítání hodnot realizujeme funkci **getline()**, která alokuje potřebnou paměť dynamicky.
- Seznam vytvoříme voláním funkce **create()**.
- Nastavíme funkce **isLess()**.

Jan Faigl, 2024 BAB36PRGA – Přednáška 08: Spojové struktury

Kódovací příklad – Volání rozhraní seznamu 2/3

```

55 clang -fPIC -c list.c
56 clang -shared -o liblist.so list.o
57 cat in.txt
58 ./list
59
60 $ clang -g -fWITH_LESS -fPIC -c list.c
61 $ clang -shared -o liblist.so list.o
62
63 $ ln -s liblist.so.1 liblist.so
64
65 $ ./list
66
67 $ clang -g -L . -Wl,-rpath= . -list -o demo demo.c
68
69 $ ldd demo
70 liblist.so => ./liblist.so (0x80024d000)
71 libc.so.7 => /lib/libc.so.7 (0x800251000)
72
73 $ ./demo < in.txt 2>&1 | head -n 12
74 DEBUG: read "4"
75 DEBUG: read "2"
76 DEBUG: read "1"
77 DEBUG: read "13"
78 DEBUG: read "6"
79 DEBUG: read "1"
80 DEBUG: read "3"
81 DEBUG: read "5"
82 DEBUG: read "9"
83 DEBUG: read "15"
84 DEBUG: read "15"
85 DEBUG: read "1"
86 DEBUG: read "4"
87 DEBUG: read "5"
88 DEBUG: read "9"
89 DEBUG: read "15"
90 DEBUG: read "9"
91 DEBUG: read "15"
92 DEBUG: read "9"
93 DEBUG: read "9"
94 DEBUG: read "15"
95 DEBUG: read "9"
96 DEBUG: read "9"
97 DEBUG: read "9"
98 DEBUG: read "9"
99 DEBUG: read "9"
100 DEBUG: read "9"
101 DEBUG: read "9"
102 DEBUG: read "9"
103 DEBUG: read "9"
104 DEBUG: read "9"
105 DEBUG: read "9"
106 DEBUG: read "9"
107 DEBUG: read "9"
108 DEBUG: read "9"
109 DEBUG: read "9"
110 DEBUG: read "9"
111 DEBUG: read "9"
112 DEBUG: read "9"
113 DEBUG: read "9"
114 DEBUG: read "9"
115 DEBUG: read "9"
116 DEBUG: read "9"
117 DEBUG: read "9"
118 DEBUG: read "9"
119 DEBUG: read "9"
120 DEBUG: read "9"
121 DEBUG: read "9"
122 DEBUG: read "9"
123 DEBUG: read "9"
124 DEBUG: read "9"
125 DEBUG: read "9"
126 DEBUG: read "9"
127 DEBUG: read "9"
128 DEBUG: read "9"
129 DEBUG: read "9"
130 DEBUG: read "9"
131 DEBUG: read "9"
132 DEBUG: read "9"
133 DEBUG: read "9"
134 DEBUG: read "9"
135 DEBUG: read "9"
136 DEBUG: read "9"
137 DEBUG: read "9"
138 DEBUG: read "9"
139 DEBUG: read "9"
140 DEBUG: read "9"
141 DEBUG: read "9"
142 DEBUG: read "9"
143 DEBUG: read "9"
144 DEBUG: read "9"
145 DEBUG: read "9"
146 DEBUG: read "9"
147 DEBUG: read "9"
148 DEBUG: read "9"
149 DEBUG: read "9"
150 DEBUG: read "9"
151 DEBUG: read "9"
152 DEBUG: read "9"
153 DEBUG: read "9"
154 DEBUG: read "9"
155 DEBUG: read "9"
156 DEBUG: read "9"
157 DEBUG: read "9"
158 DEBUG: read "9"
159 DEBUG: read "9"
160 DEBUG: read "9"
161 DEBUG: read "9"
162 DEBUG: read "9"
163 DEBUG: read "9"
164 DEBUG: read "9"
165 DEBUG: read "9"
166 DEBUG: read "9"
167 DEBUG: read "9"
168 DEBUG: read "9"
169 DEBUG: read "9"
170 DEBUG: read "9"
171 DEBUG: read "9"
172 DEBUG: read "9"
173 DEBUG: read "9"
174 DEBUG: read "9"
175 DEBUG: read "9"
176 DEBUG: read "9"
177 DEBUG: read "9"
178 DEBUG: read "9"
179 DEBUG: read "9"
180 DEBUG: read "9"
181 DEBUG: read "9"
182 DEBUG: read "9"
183 DEBUG: read "9"
184 DEBUG: read "9"
185 DEBUG: read "9"
186 DEBUG: read "9"
187 DEBUG: read "9"
188 DEBUG: read "9"
189 DEBUG: read "9"
190 DEBUG: read "9"
191 DEBUG: read "9"
192 DEBUG: read "9"
193 DEBUG: read "9"
194 DEBUG: read "9"
195 DEBUG: read "9"
196 DEBUG: read "9"
197 DEBUG: read "9"
198 DEBUG: read "9"
199 DEBUG: read "9"
200 DEBUG: read "9"
201 DEBUG: read "9"
202 DEBUG: read "9"
203 DEBUG: read "9"
204 DEBUG: read "9"
205 DEBUG: read "9"
206 DEBUG: read "9"
207 DEBUG: read "9"
208 DEBUG: read "9"
209 DEBUG: read "9"
210 DEBUG: read "9"
211 DEBUG: read "9"
212 DEBUG: read "9"
213 DEBUG: read "9"
214 DEBUG: read "9"
215 DEBUG: read "9"
216 DEBUG: read "9"
217 DEBUG: read "9"
218 DEBUG: read "9"
219 DEBUG: read "9"
220 DEBUG: read "9"
221 DEBUG: read "9"
222 DEBUG: read "9"
223 DEBUG: read "9"
224 DEBUG: read "9"
225 DEBUG: read "9"
226 DEBUG: read "9"
227 DEBUG: read "9"
228 DEBUG: read "9"
229 DEBUG: read "9"
230 DEBUG: read "9"
231 DEBUG: read "9"
232 DEBUG: read "9"
233 DEBUG: read "9"
234 DEBUG: read "9"
235 DEBUG: read "9"
236 DEBUG: read "9"
237 DEBUG: read "9"
238 DEBUG: read "9"
239 DEBUG: read "9"
240 DEBUG: read "9"
241 DEBUG: read "9"
242 DEBUG: read "9"
243 DEBUG: read "9"
244 DEBUG: read "9"
245 DEBUG: read "9"
246 DEBUG: read "9"
247 DEBUG: read "9"
248 DEBUG: read "9"
249 DEBUG: read "9"
250 DEBUG: read "9"
251 DEBUG: read "9"
252 DEBUG: read "9"
253 DEBUG: read "9"
254 DEBUG: read "9"
255 DEBUG: read "9"
256 DEBUG: read "9"
257 DEBUG: read "9"
258 DEBUG: read "9"
259 DEBUG: read "9"
260 DEBUG: read "9"
261 DEBUG: read "9"
262 DEBUG: read "9"
263 DEBUG: read "9"
264 DEBUG: read "9"
265 DEBUG: read "9"
266 DEBUG: read "9"
267 DEBUG: read "9"
268 DEBUG: read "9"
269 DEBUG: read "9"
270 DEBUG: read "9"
271 DEBUG: read "9"
272 DEBUG: read "9"
273 DEBUG: read "9"
274 DEBUG: read "9"
275 DEBUG: read "9"
276 DEBUG: read "9"
277 DEBUG: read "9"
278 DEBUG: read "9"
279 DEBUG: read "9"
280 DEBUG: read "9"
281 DEBUG: read "9"
282 DEBUG: read "9"
283 DEBUG: read "9"
284 DEBUG: read "9"
285 DEBUG: read "9"
286 DEBUG: read "9"
287 DEBUG: read "9"
288 DEBUG: read "9"
289 DEBUG: read "9"
290 DEBUG: read "9"
291 DEBUG: read "9"
292 DEBUG: read "9"
293 DEBUG: read "9"
294 DEBUG: read "9"
295 DEBUG: read "9"
296 DEBUG: read "9"
297 DEBUG: read "9"
298 DEBUG: read "9"
299 DEBUG: read "9"
300 DEBUG: read "9"
301 DEBUG: read "9"
302 DEBUG: read "9"
303 DEBUG: read "9"
304 DEBUG: read "9"
305 DEBUG: read "9"
306 DEBUG: read "9"
307 DEBUG: read "9"
308 DEBUG: read "9"
309 DEBUG: read "9"
310 DEBUG: read "9"
311 DEBUG: read "9"
312 DEBUG: read "9"
313 DEBUG: read "9"
314 DEBUG: read "9"
315 DEBUG: read "9"
316 DEBUG: read "9"
317 DEBUG: read "9"
318 DEBUG: read "9"
319 DEBUG: read "9"
320 DEBUG: read "9"
321 DEBUG: read "9"
322 DEBUG: read "9"
323 DEBUG: read "9"
324 DEBUG: read "9"
325 DEBUG: read "9"
326 DEBUG: read "9"
327 DEBUG: read "9"
328 DEBUG: read "9"
329 DEBUG: read "9"
330 DEBUG: read "9"
331 DEBUG: read "9"
332 DEBUG: read "9"
333 DEBUG: read "9"
334 DEBUG: read "9"
335 DEBUG: read "9"
336 DEBUG: read "9"
337 DEBUG: read "9"
338 DEBUG: read "9"
339 DEBUG: read "9"
340 DEBUG: read "9"
341 DEBUG: read "9"
342 DEBUG: read "9"
343 DEBUG: read "9"
344 DEBUG: read "9"
345 DEBUG: read "9"
346 DEBUG: read "9"
347 DEBUG: read "9"
348 DEBUG: read "9"
349 DEBUG: read "9"
350 DEBUG: read "9"
351 DEBUG: read "9"
352 DEBUG: read "9"
353 DEBUG: read "9"
354 DEBUG: read "9"
355 DEBUG: read "9"
356 DEBUG: read "9"
357 DEBUG: read "9"
358 DEBUG: read "9"
359 DEBUG: read "9"
360 DEBUG: read "9"
361 DEBUG: read "9"
362 DEBUG: read "9"
363 DEBUG: read "9"
364 DEBUG: read "9"
365 DEBUG: read "9"
366 DEBUG: read "9"
367 DEBUG: read "9"
368 DEBUG: read "9"
369 DEBUG: read "9"
370 DEBUG: read "9"
371 DEBUG: read "9"
372 DEBUG: read "9"
373 DEBUG: read "9"
374 DEBUG: read "9"
375 DEBUG: read "9"
376 DEBUG: read "9"
377 DEBUG: read "9"
378 DEBUG: read "9"
379 DEBUG: read "9"
380 DEBUG: read "9"
381 DEBUG: read "9"
382 DEBUG: read "9"
383 DEBUG: read "9"
384 DEBUG: read "9"
385 DEBUG: read "9"
386 DEBUG: read "9"
387 DEBUG: read "9"
388 DEBUG: read "9"
389 DEBUG: read "9"
390 DEBUG: read "9"
391 DEBUG: read "9"
392 DEBUG: read "9"
393 DEBUG: read "9"
394 DEBUG: read "9"
395 DEBUG: read "9"
396 DEBUG: read "9"
397 DEBUG: read "9"
398 DEBUG: read "9"
399 DEBUG: read "9"
400 DEBUG: read "9"
401 DEBUG: read "9"
402 DEBUG: read "9"
403 DEBUG: read "9"
404 DEBUG: read "9"
405 DEBUG: read "9"
406 DEBUG: read "9"
407 DEBUG: read "9"
408 DEBUG: read "9"
409 DEBUG: read "9"
410 DEBUG: read "9"
411 DEBUG: read "9"
412 DEBUG: read "9"
413 DEBUG: read "9"
414 DEBUG: read "9"
415 DEBUG: read "9"
416 DEBUG: read "9"
417 DEBUG: read "9"
418 DEBUG: read "9"
419 DEBUG: read "9"
420 DEBUG: read "9"
421 DEBUG: read "9"
422 DEBUG: read "9"
423 DEBUG: read "9"
424 DEBUG: read "9"
425 DEBUG: read "9"
426 DEBUG: read "9"
427 DEBUG: read "9"
428 DEBUG: read "9"
429 DEBUG: read "9"
430 DEBUG: read "9"
431 DEBUG: read "9"
432 DEBUG: read "9"
433 DEBUG: read "9"
434 DEBUG: read "9"
435 DEBUG: read "9"
436 DEBUG: read "9"
437 DEBUG: read "9"
438 DEBUG: read "9"
439 DEBUG: read "9"
440 DEBUG: read "9"
441 DEBUG: read "9"
442 DEBUG: read "9"
443 DEBUG: read "9"
444 DEBUG: read "9"
445 DEBUG: read "9"
446 DEBUG: read "9"
447 DEBUG: read "9"
448 DEBUG: read "9"
449 DEBUG: read "9"
450 DEBUG: read "9"
451 DEBUG: read "9"
452 DEBUG: read "9"
453 DEBUG: read "9"
454 DEBUG: read "9"
455 DEBUG: read "9"
456 DEBUG: read "9"
457 DEBUG: read "9"
458 DEBUG: read "9"
459 DEBUG: read "9"
460 DEBUG: read "9"
461 DEBUG: read "9"
462 DEBUG: read "9"
463 DEBUG: read "9"
464 DEBUG: read "9"
465 DEBUG: read "9"
466 DEBUG: read "9"
467 DEBUG: read "9"
468 DEBUG: read "9"
469 DEBUG: read "9"
470 DEBUG: read "9"
471 DEBUG: read "9"
472 DEBUG: read "9"
473 DEBUG: read "9"
474 DEBUG: read "9"
475 DEBUG: read "9"
476 DEBUG: read "9"
477 DEBUG: read "9"
478 DEBUG: read "9"
479 DEBUG: read "9"
480 DEBUG: read "9"
481 DEBUG: read "9"
482 DEBUG: read "9"
483 DEBUG: read "9"
484 DEBUG: read "9"
485 DEBUG: read "9"
486 DEBUG: read "9"
487 DEBUG: read "9"
488 DEBUG: read "9"
489 DEBUG: read "9"
490 DEBUG: read "9"
491 DEBUG: read "9"
492 DEBUG: read "9"
493 DEBUG: read "9"
494 DEBUG: read "9"
495 DEBUG: read "9"
496 DEBUG: read "9"
497 DEBUG: read "9"
498 DEBUG: read "9"
499 DEBUG: read "9"
500 DEBUG: read "9"
501 DEBUG: read "9"
502 DEBUG: read "9"
503 DEBUG: read "9"
504 DEBUG: read "9"
505 DEBUG: read "9"
506 DEBUG: read "9"
507 DEBUG: read "9"
508 DEBUG: read "9"
509 DEBUG: read "9"
510 DEBUG: read "9"
511 DEBUG: read "9"
512 DEBUG: read "9"
513 DEBUG: read "9"
514 DEBUG: read "9"
515 DEBUG: read "9"
516 DEBUG: read "9"
517 DEBUG: read "9"
518 DEBUG: read "9"
519 DEBUG: read "9"
520 DEBUG: read "9"
521 DEBUG: read "9"
522 DEBUG: read "9"
523 DEBUG: read "9"
524 DEBUG: read "9"
525 DEBUG: read "9"
526 DEBUG: read "9"
527 DEBUG: read "9"
528 DEBUG: read "9"
529 DEBUG: read "9"
530 DEBUG: read "9"
531 DEBUG: read "9"
532 DEBUG: read "9"
533 DEBUG: read "9"
534 DEBUG: read "9"
535 DEBUG: read "9"
536 DEBUG: read "9"
537 DEBUG: read "9"
538 DEBUG: read "9"
539 DEBUG: read "9"
540 DEBUG: read "9"
541 DEBUG: read "9"
542 DEBUG: read "9"
543 DEBUG: read "9"
544 DEBUG: read "9"
545 DEBUG: read "9"
546 DEBUG: read "9"
547 DEBUG: read "9"
548 DEBUG: read "9"
549 DEBUG: read "9"
550 DEBUG: read "9"
551 DEBUG: read "9"
552 DEBUG: read "9"
553 DEBUG: read "9"
554 DEBUG: read "9"
555 DEBUG: read "9"
556 DEBUG: read "9"
557 DEBUG: read "9"
558 DEBUG: read "9"
559 DEBUG: read "9"
560 DEBUG: read "9"
561 DEBUG: read "9"
562 DEBUG: read "9"
563 DEBUG: read "9"
564 DEBUG: read "9"
565 DEBUG: read "9"
566 DEBUG: read "9"
567 DEBUG: read "9"
568 DEBUG: read "9"
569 DEBUG: read "9"
570 DEBUG: read "9"
571 DEBUG: read "9"
572 DEBUG: read "9"
573 DEBUG: read "9"
574 DEBUG: read "9"
575 DEBUG: read "9"
576 DEBUG: read "9"
577 DEBUG: read "9"
578 DEBUG: read "9"
579 DEBUG: read "9"
580 DEBUG: read "9"
581 DEBUG: read "9"
582 DEBUG: read "9"
583 DEBUG: read "9"
584 DEBUG: read "9"
585 DEBUG: read "9"
586 DEBUG: read "9"
587 DEBUG: read "9"
588 DEBUG: read "9"
589 DEBUG: read "9"
590 DEBUG: read "9"
591 DEBUG: read "9"
592 DEBUG: read "9"
593 DEBUG: read "9"
594 DEBUG: read "9"
595 DEBUG: read "9"
596 DEBUG: read "9"
597 DEBUG: read "9"
598 DEBUG: read "9"
599 DEBUG: read "9"
600 DEBUG: read "9"
601 DEBUG: read "9"
602 DEBUG: read "9"
603 DEBUG: read "9"
604 DEBUG: read "9"
605 DEBUG: read "9"
606 DEBUG: read "9"
607 DEBUG: read "9"
608 DEBUG: read "9"
609 DEBUG: read "9"
610 DEBUG: read "9"
611 DEBUG: read "9"
612 DEBUG: read "9"
613 DEBUG: read "9"
614 DEBUG: read "9"
615 DEBUG: read "9"
616 DEBUG: read "9"
617 DEBUG: read "9"
618 DEBUG: read "9"
619 DEBUG: read "9"
620 DEBUG: read "9"
621 DEBUG: read "9"
622 DEBUG: read "9"
623 DEBUG: read "9"
624 DEBUG: read "9"
625 DEBUG: read "9"
626 DEBUG: read "9"
627 DEBUG: read "9"
628 DEBUG: read "9"
629 DEBUG: read "9"
630 DEBUG: read "9"
631 DEBUG: read "9"
632 DEBUG: read "9"
633 DEBUG: read "9"
634 DEBUG: read "9"
635 DEBUG: read "9"
636 DEBUG: read "9"
637 DEBUG: read "9"
638 DEBUG: read "9"
639 DEBUG: read "9"
640 DEBUG: read "9"
641 DEBUG: read "9"
642 DEBUG: read "9"
643 DEBUG: read "9"
644 DEBUG: read "9"
645 DEBUG: read "9"
646 DEBUG: read "9"
647 DEBUG: read "9"
648 DEBUG: read "9"
649 DEBUG: read "9"
650 DEBUG: read "9"
651 DEBUG: read "9"
652 DEBUG: read "9"
653 DEBUG: read "9"
654 DEBUG: read "9"
655 DEBUG: read "9"
656 DEBUG: read "9"
657 DEBUG: read "9"
658 DEBUG: read "9"
659 DEBUG: read "9"
660 DEBUG: read "9"
661 DEBUG: read "9"
662 DEBUG: read "9"
663 DEBUG: read "9"
664 DEBUG: read "9"
665 DEBUG: read "9"
666 DEBUG: read "9"
667 DEBUG: read "9"
668 DEBUG: read "9"
669 DEBUG: read "9"
670 DEBUG: read "9"
671 DEBUG: read "9"
672 DEBUG: read "9"
673 DEBUG: read "9"
674 DEBUG: read "9"
675 DEBUG: read "9"
676 DEBUG: read "9"
677 DEBUG: read "9"
678 DEBUG: read "9"
679 DEBUG: read "9"
680 DEBUG: read "9"
681 DEBUG: read "9"
682 DEBUG: read "9"
683 DEBUG: read "9"
684 DEBUG: read "9"
685 DEBUG: read "9"
686 DEBUG: read "9"
687 DEBUG: read "9"
688 DEBUG: read "9"
689 DEBUG: read "9"
690 DEBUG: read "9"
691 DEBUG: read "9"
692 DEBUG: read "9"
693 DEBUG: read "9"
694 DEBUG: read "9"
695 DEBUG: read "9"
696 DEBUG: read "9"
697 DEBUG: read "9"
698 DEBUG: read "9"
699 DEBUG: read "9"
700 DEBUG: read "9"
701 DEBUG: read "9"
702 DEBUG: read "9"
703 DEBUG: read "9"
704 DEBUG: read "9"
705 DEBUG: read "9"
706 DEBUG: read "9"
707 DEBUG: read "9"
708 DEBUG: read "9"
709 DEBUG: read "9"
710 DEBUG: read "9"
711 DEBUG: read "9"
712 DEBUG: read "9"
713 DEBUG: read "9"
714 DEBUG: read "9"
715 DEBUG: read "9"
716 DEBUG: read "9"
717 DEBUG: read "9"
718 DEBUG: read "9"
719 DEBUG: read "9"
720 DEBUG: read "9"
721 DEBUG: read "9"
722 DEBUG: read "9"
723 DEBUG: read "9"
724 DEBUG: read "9"
725 DEBUG: read "9"
726 DEBUG: read "9"
727 DEBUG: read "9"
728 DEBUG: read "9"
729 DEBUG: read "9"
730 DEBUG: read "9"
731 DEBUG: read "9"
732 DEBUG: read "9"
733 DEBUG: read "9"
734 DEBUG: read "9"
735 DEBUG: read "9"
736 DEBUG: read "9"
737 DEBUG: read "9"
738 DEBUG: read "9"
739 DEBUG: read "9"
740 DEBUG: read "9"
741 DEBUG: read "9"
742 DEBUG: read "9"
743 DEBUG: read "9"
744 DEBUG: read "9"
745 DEBUG: read "9"
746 DEBUG: read "9"
747 DEBUG: read "9"
748 DEBUG: read "9"
749 DEBUG: read "9"
750 DEBUG: read "9"
751 DEBUG: read "9"
752 DEBUG: read "9"
753 DEBUG: read "9"
754 DEBUG: read "9"
755 DEBUG: read "9"
756 DEBUG: read "9"
757 DEBUG: read "9"
758 DEBUG: read "9"
759 DEBUG: read "9"
760 DEBUG: read "9"
761 DEBUG: read "9"
762 DEBUG: read "9"
763 DEBUG: read "9"
764 DEBUG: read "9"
765 DEBUG: read "9"
766 DEBUG: read "9"
767 DEBUG: read "9"
768 DEBUG: read "9"
769 DEBUG: read "9"
770 DEBUG: read "9"
771 DEBUG: read "9"
772 DEBUG: read "9"
773 DEBUG: read "9"
774 DEBUG: read "9"
775 DEBUG: read "9"
776 DEBUG: read "9"
777 DEBUG: read "9"
778 DEBUG: read "9"
779 DEBUG: read "9"
780 DEBUG: read "9"
781 DEBUG: read "9"
782 DEBUG: read "9"
783 DEBUG: read "9"
784 DEBUG: read "9"
785 DEBUG: read "9"
786 DEBUG: read "9"
787 DEBUG: read "9"
788 DEBUG: read "9"
789 DEBUG: read "9"
790 DEBUG: read "9"
791 DEBUG: read "9"
792 DEBUG: read
```