11 Implementacja list jednokierunkowych

- **Założenia:**
  - W każdym zadaniu tworzymy listy jednokierunkowe, ogólnego przeznaczenia (patrz – wykład).
  - Nagłówek każdej listy jest typu `List` – zapisany w szablonie programu. W stosunku do przykładu podanego na wykładzie został on rozszerzony o pole wskaźnika na funkcję modyfikującą dane elementu listy
- **Plan ćwiczenia** (wg narastającego stopnia trudności):
  - W elementach listy w zadaniu 11.1 jest używana najprostsza struktura danych – jedna liczba całkowita.
  - Zadanie 11.2 – porównanie czasu wykonywania 3 różnych algorytmów – ma charakter poglądowy i nie wymaga pisania definicji dodatkowych funkcji. To zadanie jest ważne, choć nie będzie oceniane.
  - W zadaniu 11.3 dane, które mają być zapisywane w liście są wyrazami wczytywanymi ze strumienia wejściowego. Dodatkowo, w zadaniu 11.4, dane w każdym elemencie listy są uzupełnione krotnością pamiętanego w nim wyrazu. Dla zmniejszenia nakładu pracy można zastosować strukturę danych obejmującą krotność wyrazów już w zadaniu 11.3.
- W tym tygodniu, z założenia i celowo, nie jest jeszcze przewidziane sortowanie elementów listy. Dlatego w tych zadaniach, w których jest wymagane uporządkowanie, należy w trakcie wykonywania programu wprowadzać dane tak, aby porządek był zachowany.

11.1 Funkcje podstawowe

Szablon programu należy uzupełnić o definicje funkcji (dugi parametr przekazuje do funkcji adres pamięci przydzielonej dla "nowej" danej `a`):

1. `void pushFront(List *list, void *a)` – dodaj element na początek listy;
2. `void pushBack(List *list, void *a)` – dodaj element na koniec listy;
3. `void popFront(List *list)` – usuń pierwszy element listy;
4. `void reverse(List *list)` – odwróć kolejność wszystkich elementów listy w jednorazowym przebiegu pętli po wszystkich elementach (jedyną instrukcją sterującą w tej funkcji jest jedna instrukcja `while()`);
5. `void insertInOrder(List *list, void *a)` – dodaj element do listy (z założenia - uporządkowanej). Jeżeli "nowa" dana jest już zapisana w jednym z elementów listy, to dane tego elementu są modyfikowane funkcją wskazaną w polu `modifyData` nagłówka listy, a pamięć przydzielone danej `a` jest zwalniana. Jeżeli w liście takiego elementu nie ma, to nowy element (z `a`) jest tworzony i dopisany do listy z zachowaniem uporządkowania listy. Przydatną – po niewielkich modyfikacjach – może się tu okazać funkcja `findInsertionPoint()` (poznana na wykładzie 08[1]).

[1] `https://home.agh.edu.pl/~pszwed/wiki/lib/exe/fetch.php?media=08-imperatywne-jezyk-c-dynamiczna-alokacja-pamieci.pdf`

Zadanie 11.1. Podstawowe operacje na elementach listy z danymi liczbowymi

W polu `data` elementu listy jest zapisywany adres danej typu `DataInt` (typ jest zdefiniowany w szablonie programu).

Szablon programu należy uzupełnić o definicje funkcji:

1. `void *create_data_int(int v)` – przydziela pamięć dla danej i zapisuje w niej wartość v, zwraca adres przydzielonej pamięci;
2. `void dump_int(const void *d)`,
3. `void free_int(const void *d)`,
4. `void cmp_int(const void *a, const void *b)`,

Ogólna postać danych (do każdego podpunktu):
numer zadania
`n` – liczba poleceń
`n` linii poleceń

Każde polecenie składa się z litery (kodu polecenia) i pozostałych danych (w zależności od typu polecenia).

Lista poleceń:

1. `f value - pushFront()`
2. `b value - pushBack()`
3. `d - popFront()`
4. `r - reverse()`
5. `i value - insertInOrder()`

Oceniane automatycznie będą dwa zestawy poleceń – bez i z listą uporządkowaną, czyli bez i z poleceniem „i" (funkcją `insertInOrder()`) .

- **Wejście**
  1 liczba poleceń
  kolejne polecenia
- **Wyjście**
  liczby zapisane w kolejnych elementach listy
- **Przykład 1:**

  Wejście:
  1
  4
  b 10
  f 5
  r
  b 3
  Wyjście:
  10 5 3

- **Przykład 2** (z zachowaniem porządku rosnącego):

  Wejście:
  1
  6
  f 5
  b 10
  i 7
  d
  i 13
  i 1
  Wyjście:
  1 7 10 13

Zadanie 11.2. Porównanie efektywności dodawania elementu na końcu listy bez stosowania i z zastosowaniem wskaźnika na ostatni element listy

Należy wykorzystać funkcje napisane w ramach zadania poprzedniego oraz uzupełnić szablon programu o definicję funkcji `pushBack_v0(list,value)`, która – podobnie jak funkcja `pushBack(list,value)` – dopisuje element z liczbą `value` na końcu listy. Różnica polega na tym, że funkcja `pushBack_v0(list,value)` nie korzysta z pola `tail` nagłówka listy, w którym jest pamiętany adres ostatniego elementu listy.

Zadanie polega na utworzeniu listy o zadanej liczbie elementów i wyznaczeniu czasu wykonywania 3 algorytmów, które prowadzą do tego samego rezultatu końcowego – utworzenia listy o tej samej kolejności elementów:

1. z funkcją `pushBack_v0(list,value)` dodającą elementy na końcu listy bez użycia wskaźnika na ostatni element – szablon programu należy uzupełnić o definicję tej funkcji;
2. z funkcją `pushFront(list,value)` tworzącą wszystkie elementy listy oraz funkcją `reverse(list)` odwracającą kolejność elementów listy;
3. z funkcją `pushBack(list,value)` dodającą elementy na końcu listy z wykorzystaniem wskaźnika na ostatni element.

Wyznaczone czasy są mierzone zegarem czasu rzeczywistego, a nie czasem wykonywania przez procesor wskazanego algorytmu. Dlatego pomiar czasu jest obarczony dużym błędem pochodzącym od wliczenia do niego także czasu wykonywania przez procesor innych zadań. Zatem – dla uśrednienia wyników – zadanie należy powtórzyć klika razy dla każdej liczby elementów (np. dla 30 i 30000).

- Wejście:
  2
  liczba elementów listy
- Wyjście:
  Czasy wykonania trzech algorytmów
- Przykład:
  Wejście:

2 3000
Wyjście:
n = 3000. Back bez tail. Czas = 0.018505 s.
n = 3000. Front + revers. Czas = 7.6e-05 s.
n = 3000. Back z tail. Czas = 6.6e-05 s.

Zadanie 11.3. Lista wyrazów wczytanych ze strumienia wejściowego (bez znaków diakrytycznych)

Założenia:

1. Przyjmijmy określenie „wyraz": łańcuch znaków ASCII nie zawierający ograniczników (delimiterów): znaków białych oraz `.,?!:;-`.
2. Kolejność elementów listy ma być zgodna z kolejnością odczytywanych wyrazów.
3. W polu `data` elementu listy jest zapisywany adres danej typu `DataWord` (typ jest zdefiniowany w szablonie programu). Struktura elementu listy zawiera pole (typu wskaźnikowego) dla adresu, pod którym jest pamiętany wyraz.

Szablon programu należy uzupełnić o definicje funkcji:

1. `void *create_data_word(char *string, int counter)` – przydziela pamięć dla łańcucha `string` i struktury typu `DataWord`, do przydzielonej pamięci wpisuje odpowiednie dane, zwraca adres struktury.
2. `void dump_word (const void *d)`
3. `void free_word(void *d)`

- Wejście:
  3
  linie tekstu
  Ctrl-D
- Wyjście:
  odczytane wyrazy (oddzielone spacją, bez innych delimiterów) w kolejności wczytywania[2]
- Przykład:
  Wejście:
  3
  Abc,d; zz EF-gh.
  xxx!
  Ctrl-D
  Wyjście:
  Abc d zz EF gh xxx

[2]Nie należy zwracać uwagi na aspekt użyteczności takiego programu (ten sam wynik można uzyskać bez tworzenia listy) – jest to wstępny etap dla kolejnych zadań.

Zadanie 11.4. Lista wyrazów z tekstu j.w. – dodawanie elementów wg porządku alfabetycznego ze zliczaniem krotności

Zadanie jest analogiczne do poprzedniego. Różnice:

- Struktura danych jest rozszerzona o pole licznika krotności występowania danego wyrazu w tekście.
- Elementy są dodawane do listy tak, aby zachować alfabetyczny porządek wyrazów (wielkość liter – mała czy wielka – nie ma tu znaczenia). Jeżeli dodawany wyraz jest już zapisany w liście, to należy zwiększyć jego licznik krotności.
- Program kończy się wypisaniem w kolejności alfabetycznej wyrazów o zadanej (na wejściu) krotności. Przyjmijmy, że każdy wyraz jest wypisywany małymi literami.

Szablon programu należy uzupełnić o definicje funkcji:

1. `int cmp_word_alphabet(const void *a, const void *b)`
2. `int cmp_word_counter(const void *a, const void *b)`
3. `void modify_word(void *a)` – modyfikacja danej sprowadza się do inkrementacji licznika krotności `counter`
4. `void dumpList_word_if(List *plist, int n)` – Wypisuje dane elementów spełniających warunek równości sprawdzany funkcją wskazywaną w polu compareData nagłówka listy.

- Wejście:
  4
  wybrana krotność `k` wyrazu linie tekstu
- Wyjście:
  wyrazy powtarzające się w tekście `k` razy (małymi literami, w porządku alfabetycznym)
- Przykład:
  Wejście:
  4 2
  Xy, ABC, ab, abc,
  xY, ab - ab.
  Wyjście:
  abc xy