1. Operacje na uogólnionej liście jednokierunkowej

1 Uwagi ogólne

Celem ćwiczenia jest zaimplementowanie struktury opisującej listę jednokierunkową elementów dowolnego typu oraz funkcji realizujących operacje na tej liście.

W kolejnych punktach na podstawie stworzonego szablonu będziemy tworzyć listę liczb całkowitych oraz stringów z licznikiem wystąpień.

W każdym zadaniu tworzymy listę jednokierunkową. Nagłówek listy zawiera co najmniej pola `head` oraz `tail` (jak w szablonie programu) a także wskaźniki do czterech funkcji, które ze względu na swoją specyfikę muszą zostać zdefiniowane oddzielnie dla różnych typów. Te funkcje to:

1. `dump_data()` – wydrukuj dane elementu listy
2. `free_data()` – zwolnij pamięć danych elementu listy
3. `compare_data()` – porównaj dwa elementy wg zadanego kryterium
4. `insert_sorted()` – wstaw element do listy zachowując porządek

Plan ćwiczenia

1. W elementach listy w zadaniu 1 jest używana najprostsza struktura danych – jedna liczba całkowita. Program zawiera funkcje realizujące podstawowe operacje na elementach listy.
2. W zadaniu 2 dane, które mają być zapisywane w liście są słowami (stringami) wczytywanymi ze strumienia wejściowego. W związku z tym pojawiają się dwa nowe wymagania:
   (a) Napisanie funkcji, które pobierają kolejne słowa ze strumienia wejściowego i dla każdego z nich dodają do listy nowy element.
   (b) W strukturze elementu listy, zamiast pola liczby całkowitej, powinno znaleźć się pole wskaźnika zawierającego adres pamięci, w której jest zapisany łańcuch znaków (pojedynczy słowo). Odrzucamy możliwość zapisywania słów w tablicy znakowej zdefiniowanej bezpośrednio w strukturze elementu listy.
3. W zadaniu 3 dodatkowo:
   (a) Zliczamy krotności pojawiania się każdego słowa we wczytywanym tekście. W konsekwencji konieczna jest modyfikacja struktury danych zapisanych w elemencie listy – adres pamiętanego słowa należy zastąpić adresem struktury, w której oprócz pola adresu słowa będzie pole licznika krotności tego słowa.
   (b) Elementy listy mają być uporządkowane wg alfabetycznej kolejności pamiętanych w nich słów – tu pomocna może być napisana wcześniej funkcja `insert_in_order()`.

Struktura elementu listy w zadaniach 2 i 3 może być taka sama (czyli string i licznik, który w zadaniu 2 nie jest wykorzystywany).

Elementy list w pierwszym zadaniu różnią polem danych się od elementów list z dwóch następnych zadań. Ta różnica może spowodować konieczność napisania dwóch wersji funkcji operujących na elementach listy.

Z tego powodu do wykonania zadanie stosujemy listę ogólnego przeznaczenia (o której była mowa na wykładzie 08). Szablony programów są przygotowane są przygotowanne właśnie dla tej wersji.

2 Zadania

2.1 Podstawowe operacje na elementach listy

Szablon programu zawiera jedną przykładową funkcję w postaci kompletnej, jako "inspirację" do konstrukcji pozostałych. Należy uzupełnić implementację pozostałych z wymienionych funkcji:

1. `push_front()` – dodaj element na początek listy;
2. `push_back()` – dodaj element na koniec listy;
3. `pop_front()` – usuń pierwszy element listy;
4. `reverse()` – odwróć kolejność wszystkich elementów listy;
5. `insert_in_order()` – dodaj element do listy (z założenia uporządkowanej);
6. `dump_list()` – wypisz dane ze wszystkich elementów listy.
7. `free_list()` – usuń z listy wszystkie elementy.

Pomocne uwagi:

1. Wyznaczanie relacji porządku elementów listy: porządek określa funkcja, której adres jest zapisany w nagłówku listy w polu `insert_sorted`;
2. Funkcja wskazywana wartością pola `insert_sorted`, oprócz sprawdzania relacji porządku elementów, obsługuje przypadek, gdy w liście jest już element z danymi takimi samymi jak dane dodawane. Wtedy nowy element nie jest dodawany do listy. Ponadto przyjmujemy, że gdy dane są w postaci
   - liczbowej, to żadna dodatkowa akcja nie jest wykonywana,
   - stringu (słowa), to krotność tego słowa jest inkrementowana.

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(list, value)`
2. `b value - pushBack(list, value)`
3. `i value - insertInOrder(list, value)`
4. `d - popFront(list)`
5. `r - reverse(list)`

Oceniane będą dwa zestawy poleceń: wykorzystujące lub nie listę posortowaną (oraz polecenie `'i'`).

- **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
  ```

2.2 Lista słów pobranych z tekstu

Zadanie polega na utworzeniu listy elementów zawierających informację o słowach wczytanych ze standardowego strumienia wejściowego.

Założenia:

1. Definiujemy „słowo" jako łańcuch znaków ASCII nie zawierający ograniczników (delimiterów): znaków białych oraz `.,?!:;-`.
2. Kolejność elementów listy jest zgodna z kolejnością odczytywanych słów.
3. Struktura elementu listy zawiera pole (typu wskaźnikowego) dla adresu, pod którym jest pamiętane słowo – czyli w elemencie listy nie zapisujemy słów (ponieważ nie znamy a priori ich długości).
4. Polecane funkcje: `fgets(), strtok(), strdup()`.

- **Wejście**:
  2
  linie tekstu
- **Wyjście**:
  odczytane słowa (oddzielone jednym z delimiterów) w kolejności wczytywania.
- **Przykład**:
  Wejście:

```
2
xxx!
Abc,d;   EF-gh.
```

  Wyjście:

```
xxx
Abc
d
EF
gh
```

2.3 Lista słów z tekstu j.w. – dodawanie elementów wg porządku alfabetycznego ze zliczaniem krotności

Zadanie analogiczne do poprzedniego. Różnice:

1. Struktura danych jest rozszerzona o pole licznika krotności występowania danego słowa w tekście (aczkolwiek pole to może również występować w poprzednim zadaniu).
2. Elementy są dodawane do listy tak, aby zachować alfabetyczny porządek słów (wielkość liter nie jest uwzględniana). Jeżeli dodawane słowo jest już zapisane w liście, to należy zwiększyć jego licznik krotności.

3. Program kończy się wypisaniem małymi literami, w kolejności alfabetycznej słów o zadanej (na wejściu) krotności.

Uwaga: Pojawia się potrzeba porównywania danych wg dwóch kryteriów:

1. W czasie wstawiania elementu do listy – porównywanie alfabetycznie.
2. W czasie wybierania słów do końcowego wypisania – porównywanie krotności z zadaną liczbą.

W tym celu należy rozważyć wymianę wskaźnika do funkcji porównującej w polu `compare_data`.

- **Wejście**:

  3 wybrana krotność `k` słowa
  linie tekstu

- **Wyjście**:

  słowa powtarzające się w tekście `k` razy posortowane alfabetycznie (małymi literami)

- **Przykład**:

  Wejście:

  ```
  3 2
  Xy, ABC, ab, abc,
  xY, ab - ab.
  ```

  Wyjście:

  ```
  abc
  xy
  ```