Zestaw 4

Lista II

Algorytmy i strukury danych I

Lista

Proszę zapoznać się z prezentacją Bjarna Stroustrupa, twórcą języka C++, dotyczącej różnicy w wydajności między implementacją tablicową i wskaźnikową listy.

Interfejs

```
class List {
    void push_front(int x);     // Dołącza element na początek listy
    int pop_front();            // Usuwa i zwraca element z początku listy
    void push_back(int x);      // Dołącza element na koniec listy
    int pop_back();             // Usuwa i zwraca element z końca listy
    int size();                 // Zwraca liczbę elementów w liście
    bool empty();               // Zwraca true gdy lista jest pusta
    void clear();               // Czyści listę
    int find(int x);            // Zwraca pozycję pierwszego elementu o wartości x
    int erase(int i);           // Usuwa i zwraca element na pozycji i
    void insert(int i, int x);  // Wstawia element x przed pozycję i
    int remove(int x);          // Usuwa wystąpienia x i zwraca ich liczbę
};
```

Uwagi

- Zdefiniować konstruktor tworzący pustą listę.
- Złożoność obliczeniowa operacji powinna być *optymalna* dla danej implementacji.
- Funkcje usuwające elementy, w przypadku gdy nie jest to możliwe, powinny wyrzucać wyjątek.
- Funkcja `find()` zwraca `-1` gdy element nie występuje.
- Nie należy używać kontenera `std::vector`.

Zadanie 1. Implementacja tablicowa listy

Napisać implementację tablicową listy (`ArrayList.hpp`). Można przyjąć sztywny maksymalny rozmiar listy. Elementy tablicy przechowują jedynie wartości elementów.

Program `ArrayList.cpp` ma wczytywać dane wejściowe ze standardowego wejścia, wykonać odpowiednie operacje wykorzystując implementację tablicową listy i wypisać rezultat na standardowe wyjście. Format danych wejściowych jest taki sam jak w poprzednim zestawie.

Zadanie 2. Implementacja kursorowa

Napisać jednokierunkową implementacją kursorową listy (`CursorList.hpp`). Implementacja kursorowa łączy ze sobą cechy implementacji tablicowej (elementy są umieszczone w jednej tablicy) i wskaźnikowej

(elementy nie są ułożone sekwencyjnie). Można przyjąć sztywny maksymalny rozmiar listy. Podobnie jak w implementacji tablicowej, węzły są umieszczone w tablicy, jednak oprócz przechowywanego obiektu węzeł pamięta również indeks kolejnego węzła.

Program `CursorList.cpp` ma wczytywać dane wejściowe ze standardowego wejścia, wykonać odpowiednie operacje wykorzystując implementację kursorową listy i wypisać rezultat na standardowe wyjście. Format danych wejściowych jest taki sam jak w poprzednim zestawie.

Uwaga: Operacje `push_front` i `push_back` mają mieć złożoność $O(1)$. Chociaż lista jest jednokierunkowa można dodać pole `tail`, aby przyspieszyć operację `push_back`. Nieużyte węzły należny powiązać w listę.

Przykład

```
class CursorList {  // Klasa listy
  struct Node {     // Zagnieżdżona klasa węzła
    int x;          // Element przechowywany przez węzeł listy
    int next;       // Indeks kolejnego węzła
  };
  Node arr[...];    // Tablica węzłów
  int head;         // Indeks pierwszego węzła
  int tail;         // Indeks ostatniego węzła
  int size;         // Rozmiar listy
  int spare;        // Indeks pierwszego nieużytego elementu tablicy
};
```

> *Tablica węzłów w rzeczywistości przechowuje dwie listy:*
>
> - *listę właściwą, która rozpoczyna się od węzła o indeksie* `head`,
> - *listę wolnych węzłów, która zaczyna się od węzła o indeksie* `spare`.

Dodatkowe punkty

Dodatkowe punkty (po 1 pkt) można zdobyć za:

- Implementacja iteratora (2 pkt)
- Napisanie szablonów klas, konstruktorów (domyślny, kopiujący i przenoszący), destruktorów, operatory przypisania (kopiujący i przenoszący)
- Wykorzystanie referencji do r-wartości, semantyki przenoszenia, uniwersalnych referencji, doskonałego przekazywanie
- Napisanie testera

Pytania

1. Jakie są zalety implementacji wskaźnikowej, a jakie implementacji tablicowej?
2. Jakie są zalety implementacji kursorowej?

Uwagi

- Na platformę Pegaz należy wysłać spakowany katalog w formacie `.tar.gz` lub `zip`.
- Katalog musi się nazywać `Zestaw03` i zawierać tylko pliki źródłowe i `Makefile`.

- Pliki źródłowe muszą mieć podaną nazwę, a programy wykonywalne muszą mieć *rozszerzenie* `.x`.
- Wywołanie komendy `make` w tym katalogu powinno kompilować wszystkie programy i tylko kompilować.
- Kompilacja musi przebiegać bez błędów ani ostrzeżeń.
- Należy używać własnych implementacji typów danych w programach.
- Programy nie powinny wypisywać niczego ponad to co opisano w instrukcji. Proszę dokładnie czytać opis formatu danych wejściowych i wyjściowych.
- Implementacje klas mogą znajdować się w pliku nagłówkowym. Taka konstrukcja jest konieczna w przypadku szablonów klas.

Andrzej Görlich
a.goerlich@outlook.com