Zadanie H – Rozwijalna lista

Punktów do uzyskania: **7**

Generalia

- Zadnie polega na implementacji działań dla obsługi ciągów danych zdefiniowanego zewnętrznie typu `OBJECT_TYPE`, zwanych odtąd *obiektami*.
- Obiekty są elementami dynamicznie przydzielanych tablic.
- Wskaźniki do tablic obiektów są polami struktury nazywanej odtąd *węzłem*.
- Węzły są powiązane w implementowaną wskaźnikowo listę pojedynczo wiązaną.
- Węzły definiuje zewnętrznie struktura `NODE_STRUCT` postaci:

```
struct NODE_STRUCT {
    OBJECT_TYPE* object;
    BYTE use;
    NODE_STRUCT* next;
    }
```

... o znaczeniu pól:
- `object` - wskazuje tablicę obiektów.
- `use` - pamięta ilość aktualnie używanych obiektów tablicy.
- `next` - wskazuje następny węzeł.

- Zdefiniowany zewnętrznie typ `BYTE` jest równoważny typowi **`unsigned char`**.
- Długość przydzielanej tablicy obiektów jest zawsze stała i określona zdefiniowaną zewnętrznie wartością `SIZE` typu `BYTE`.
- Obiekty zajmują zawsze spójny początkowy fragment tablicy.
- Lista węzłów nie może zawierać tablic bez używanych obiektów.
- **Użyte dalej sformułowanie *określenie obiektu* będzie oznaczało węzeł z tablicą zawierającą obiekt oraz indeks obiektu w zawierającej tablicy.**
- Dla obiektów dostępne są operatory równości i różności.

Działania obsługi pamięci

- `NODE_STRUCT* NewNode ( void )`
  - Tworzy nowy węzeł listy typu `NODE_STRUCT`.
  - Przydziela tablicę obiektów typu `OBJECT_TYPE` do pola `object`.
  - Zwraca wskaźnik do utworzonego węzła.
  - Stanowi jedyny podprogram dopuszczający użycie operatora **`new`**.
  - Stanowi jeden z dwóch podprogramów dopuszczających użycie znaków prostokątnych nawiasów.
- `void DeleteNode ( NODE_STRUCT* )`
  - Zwraca pamięć węzła danego argumentem, wraz ze zwróceniem pamięci tablicy obiektów.
  - Stanowi jedyny podprogram dopuszczający użycie operatora **`delete`**.
  - Stanowi jeden z dwóch podprogramów dopuszczających użycie znaków prostokątnych nawiasów.
- `void Clear ( NODE_STRUCT** )`
  - Usuwa listę węzłów przekazaną argumentem wraz ze zwróceniem używanej pamięci zarówno węzłów jak i pobranych tablic obiektów.
  - Listę węzłów przekazaną argumentem ustawia na wartość `NULL`.

Działania dodające

- Dla działań dodających obowiązują reguły:
  - Jeżeli dodanie obiektu jest możliwe poprzez zwiększenie ilości obiektów w istniejącym węźle należy je wykonać w pierwszej kolejności.
  - Jeżeli dodawanie obiektu w istniejącym węźle nie jest możliwe i konieczne jest stworzenie nowego węzła, nowo tworzony węzeł może zawierać dokładnie jeden obiekt.
- `void AddFirst ( NODE_STRUCT**, OBJECT_TYPE* )`
  - Dodaje obiekt na początek ciągu.
  - Pierwszy argument wskazuje listę węzłów.
  - Drugi argument wskazuje obiekt do dodania.
  - Nowy węzeł może być dodany wyłącznie na początek listy węzłów.
- `void AddLast ( NODE_STRUCT**, OBJECT_TYPE* )`
  - Dodaje obiekt na koniec ciągu.
  - Pierwszy argument wskazuje listę węzłów.
  - Drugi argument wskazuje obiekt do dodania.
  - Nowy węzeł może być dodany wyłącznie na koniec listy węzłów.

Działania podające

- `void GetFirst ( NODE_STRUCT*, NODE_STRUCT**, BYTE* )`
  - Podaje określenie pierwszego obiektu ciągu.
  - Pierwszy argument wskazuje listę węzłów.
  - Drugi i trzeci argument przyjmują określenie pierwszego obiektu.
  - Dla pustej listy węzłów drugi argument przyjmuje wartość `NULL`, zaś zmienna przekazana trzecim argumentem pozostaje bez zmian.
- `void GetPrev ( NODE_STRUCT*, NODE_STRUCT*, BYTE, NODE_STRUCT**, BYTE* )`
  - Podaje poprzedzający obiekt względem danego obiektu.
  - Pierwszy argument wskazuje listę węzłów.
  - Drugi i trzeci argument określają dany obiekt.
  - Czwarty i piąty argument przyjmują określenie poprzedzającego obiektu.
  - Gdy poprzedzającego obiektu brak, czwarty argument przyjmuje wartość `NULL`, zaś zmienna przekazana piątym argumentem pozostaje bez zmian.
- `void GetNext ( NODE_STRUCT*, NODE_STRUCT*, BYTE, NODE_STRUCT**, BYTE* )`
  - Podaje następny obiekt względem danego obiekcie.
  - Pierwszy argument wskazuje listę węzłów.
  - Drugi i trzeci argument określają dany obiekt.
  - Czwarty i piąty argument przyjmują określenie następującego obiektu.
  - Gdy następującego obiektu brak, czwarty argument przyjmuje wartość `NULL`, zaś zmienna przekazana piątym argumentem pozostaje bez zmian.
- `void GetLast ( NODE_STRUCT*, NODE_STRUCT**, BYTE* )`
  - Podaje określenie ostatniego obiektu ciągu.
  - Pierwszy argument wskazuje listę węzłów.
  - Drugi i trzeci argument przyjmują określenie ostatniego obiektu.
  - Dla pustej listy węzłów drugi argument przyjmuje wartość `NULL`, nie zmieniając wartości przekazanej trzecim argumentem.

Działania wstawiające

- Działania wstawiające podlegają regułom działań dodających.
- `void InsertPrev ( NODE_STRUCT**, NODE_STRUCT*, BYTE, OBJECT_TYPE* )`
  - Wstawia obiekt przed danym obiektem.
  - Pierwszy argument wskazuje listę węzłów.
  - Drugi i trzeci argument określają dany obiekt.
  - Czwarty argument wskazuje wstawiany obiekt.
  - Nowy węzeł może być dodany wyłącznie przed węzłem przekazanym drugim argumentem.
- `void InsertNext ( NODE_STRUCT*, NODE_STRUCT*, BYTE, OBJECT_TYPE* )`
  - Wstawia obiekt za danym obiektem.
  - Drugi i trzeci argument określają dany obiekt.
  - Czwarty argument wskazuje wstawiany obiekt.
  - Nowy węzeł może być dodany wyłącznie za węzłem przekazanym drugim argumentem.

Działania usuwające

- Dla działań usuwających obowiązują poniższe reguły.
  - Wszystkie działania usuwające w pierwszej kolejności usuwają obiekt poprzez zmniejszenie ilości obiektów w tablicy węzła.
  - Węzły są usuwane osiągając zerową ilość używanych obiektów.
- `void RemoveFirst ( NODE_STRUCT** )`
  - Usuwa pierwszy obiekt ciągu.
  - Dla pustej listy nie jest wykonywane żadne działanie.
- `void RemovePrev ( NODE_STRUCT**, NODE_STRUCT*, BYTE )`
  - Usuwa obiekt przed danym obiektem.
  - Pierwszy argument wskazuje listę węzłów.
  - Drugi i trzeci argument określają dany obiekt.
  - W przypadku braku obiektu poprzedzającego nie jest wykonywane żadne działanie.
- `void RemoveCurrent ( NODE_STRUCT**, NODE_STRUCT*, BYTE )`
  - Usuwa z ciągu dany obiekt.
  - Pierwszy argument wskazuje listę węzłów.
  - Drugi i trzeci argument określają dany obiekt.
- `void RemoveNext ( NODE_STRUCT*, NODE_STRUCT*, BYTE )`
  - Usuwa obiekt za danym obiektem.
  - Pierwszy argument przekazuje listę węzłów.
  - Drugi i trzeci argument określają dany obiekt.
  - W przypadku braku obiektu następującego nie jest wykonywane żadne działanie.
- `void RemoveLast ( NODE_STRUCT** )`
  - Usuwa ostatni obiekt ciągu.
  - Dla pustej listy nie jest wykonywane żadne działanie.

Działania dodatkowe

- `void Find ( NODE_STRUCT*, OBJECT_TYPE*, NODE_STRUCT**, BYTE* )`
  - Wyszukuje pierwszy w kolejności obiekt równy danemu obiektowi.
  - Pierwszy argument przekazuje listę węzłów.
  - Drugi argument wskazuje dany obiekt do wyszukania.
  - Trzeci i czwarty przyjmują określenie poszukiwanego obiektu.
  - Jeżeli poszukiwanego obiektu brak, w trzecim argumencie przekazuje wartość `NULL`, zaś zmienna przekazana czwartym argumentem pozostaje bez zmian.
- `void Compress ( NODE_STRUCT* )`
  Kompresuje listę węzłów daną argumentem do postaci z wszystkimi węzłami mającymi pełne tablice obiektów z wyjątkiem ostatniego.
- `void Reverse ( NODE_STRUCT* )`
  Odwraca i kompresuje listę węzłów daną argumentem.

Dodatkowe wymagania

- Zabronione jest włączanie jakichkolwiek plików nagłówkowych.
- Używanie znaków kwadratowych nawiasów `[]` jest dopuszczalne wyłącznie w podprogramach `NewNode` oraz `DeleteNode`.
- Operator **`new`** może być użyty wyłącznie w podprogramie `NewNode`.
- Operator **`delete`** może być użyty wyłącznie w podprogramie `DeleteNode`.
- W kodzie rozwiązania zabronione jest używanie:
  - Standardowych kontenerów sekwencyjnych w rodzaju `array`, `list`, `queue`, `vector` czy `forward_list`.
  - Słów `string` oraz **`class`**.
  - Zmiennych globalnych.
  - Własnych podprogramów rozpoczynających się znakiem podkreślenia.
- **Niedopuszczalne są wycieki pamięci.**