Sprawozdanie z projektu

Zadanie projektowe numer 3

Jakub Frydlewicz, 169776
Inżynieria i analiza danych, grupa P2

1 OPIS PROBLEMU

1.1 TREŚĆ ZADANIA

Zadanie 4. Dokonaj implementacji struktury danych typu lista dwukierunkowa wraz z wszelkimi potrzebnymi operacjami charakterystycznymi dla tej struktury (inicjowanie struktury, dodawanie/usuwanie elementów, wyświetlanie elementów, zliczanie elementów/wyszukiwanie zadanego elementu itp.)

a) Przyjąć, że podstawowym typem danych przechowywanym w elemencie struktury będzie struktura z jednym polem typu int.
b) w funkcji main() przedstawić możliwości napisanej przez siebie biblioteki
c) kod powinien być opatrzony stosownymi komentarzami

1.2 DEFINICJE ZASTOSOWANYCH STRUKTUR

Lista (ang. list) jest dynamiczną strukturą danych, co oznacza, iż jej rozmiar (ilość zawartych na liście elementów) może się dowolnie (z ograniczeniem pamięci komputera) zmieniać w czasie działania programu. Lista składa się z połączonych ze sobą elementów.

Lista dwukierunkowa (ang. double conected list) - każdy element listy połączony jest z elementem następnym oraz poprzednim. Listę dwukierunkową możemy przechodzić w obu kierunkach - od pierwszego elementu do ostatniego lub od ostatniego do pierwszego. Co ważniejsze, lista dwukierunkowa pozwala nam się cofać, a jest to bardzo przydatne w wielu algorytmach.[1]

W celu realizacji struktury listy jej elementy muszą posiadać odpowiednią budowę. Połączenie z elementem następnym i poprzednim (w liście jednokierunkowej tylko z następnym) uzyskuje się zapamiętując w elemencie listy odpowiednie adresy. Dlatego element listy składa się z dwóch części:

- **adresowej** - służącej do obsługi listy
- **danych** - służącej do przechowywania danych związanych z elementem listy.

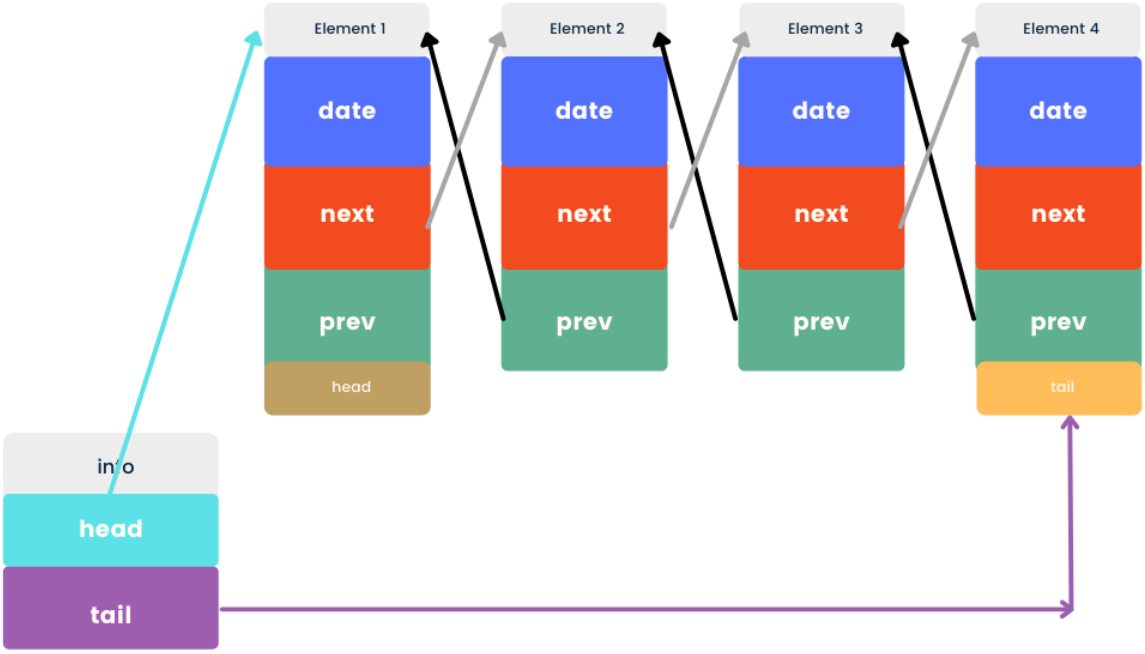
W części adresowej umieszczamy dwa pola będące wskaźnikami elementów listy:

- **next** - wskazuje następny element na liście. Jeśli zawiera wartość NULL, to dany element nie posiada następnika, czyli jest ostatnim elementem listy.
- **prev** - wskazuje poprzedni element listy dwukierunkowej. Jeśli zawiera NULL, to dany element nie posiada poprzednika, czyli jest pierwszym elementem listy. Jeśli chcemy wyszukiwać elementy listy, to często dodaje się jeszcze pole:
- **key** - zawiera tzw. klucz, czyli wartość, wg której elementy listy tworzą pewien porządek, ciąg wartości. Zawartość pola key może być wykorzystywana przez algorytmy sortujące listę lub wyszukujące na niej elementy największe lub najmniejsze.

Wskaźnik jest typem zmiennej. Przechowuje adres obiektu w pamięci i służy do uzyskiwania dostępu do tego obiektu. Nieprzetworzone wskaźnikiem jest wskaźnik, którego okres istnienia nie jest kontrolowany przez hermetyzowany obiekt, taki jak inteligentny wskaźnik.

[1] https://eduinf.waw.pl/inf/utils/010_2010/0513.php

Hermetyzacja polega na ukrywaniu pewnych danych składowych lub metod obiektów danej klasy tak, aby były one dostępne tylko metodom wewnętrznym danej klasy lub funkcjom zaprzyjaźnionym.



1 Rysunek przedstawiający strukturę listy dwukierunkowej

1.3 KRÓTKI OPIS PROGRAMU

Po uruchomieniu programu wyświetlana jest cała zawartość listy. Za pomocą Menu Głównego można:

- dodawać,
- usuwać,
- zliczać,
- szukać elementów,
- sprawdzić czy lista jest pusta
- usunąć wszystkie elementy.

2 PROGRAM

2.1 KOD

```
#include <iostream>
#include <list>
#include <windows.h>
#include <iomanip>
using namespace std;
list <int> lista;
int wybor;

void wyswietl(){
    system("CLS");
    cout << " ZAWARTOSC LISTY: " << endl <<endl;
    for(list<int>::iterator i=lista.begin(); i!= lista.end(); ++i)       cout << setw(4) << *i;
    cout << endl << endl;
}
void push_front(){
    int liczba;
    cout<<"Podaj jaka liczbe wstawic na poczatek listy: ";
    cin>>liczba;
    lista.push_front(liczba);
}
void sort(){
    cout<<"Nastapi posortowanie listy! ";
    lista.sort();
    Sleep(4000);
}
```

```
void push_back(){
    int liczba;
    cout<<"Podaj jaka liczbe wstawic na koniec listy: ";
    cin>>liczba;
    lista.push_back(liczba);
}
void pop_back(){
    cout<<"Nastapi usuniecie liczby z konca listy";
    Sleep(2000);
    lista.pop_back();
}
void size(){
    cout<<"Liczb na liscie: "<<lista.size();
    Sleep(4000);
}
void max_size(){
    cout<<"Maksymalna ilosc liczb na liscie: "<<lista.max_size();
    Sleep(5000);
}
void empty(){
    cout<<"Czy lista pusta? -> ";
    if (lista.empty()==1) cout<<"TRUE"; else cout<<"FALSE";
    Sleep(4000);
}
void wyczysc(){
    cout << "Wyczyszczam zawartosc listy."<<endl;
    lista.clear();
    Sleep(2000);
}
```

```
void remove(){
   int liczba;
   cout<<"Usun z listy wszystkie liczby rowne: ";
   cin>>liczba;
   lista.remove(liczba);
}
void nastepny(){
   int liczba;
  cout << "Podaj liczbe: ";
   cin >> liczba;
   for( list<int>::iterator i=lista.begin(); i!= lista.end(); ++i )
         if ( *i == liczba  )     cout << setw(4) << *next(i,1);
   Sleep(4000);
}
void poprzedni(){
   int liczba;
  cout << "Podaj liczbe: ";
   cin >> liczba;
   for( list<int>::iterator i=lista.begin(); i!= lista.end(); ++i )
         if ( *i == liczba  )     cout << setw(4) << *prev(i,1);
   Sleep(4000);
}
void zlicz(){
   int liczba, counter=0;
   cout << "Podaj liczbe: ";   cin >> liczba;
   for( list<int>::iterator i=lista.begin(); i!= lista.end(); ++i )      {
         if (*i == liczba)          counter++;
     }
   cout << "Element " << liczba << " wystapila: " << counter << " razy." << endl;
   Sleep(4000);   }
```

```
void znajdz()
{
  int liczba, counter=0;
  cout << "Podaj liczbe: ";
  cin >> liczba;
  cout << "Pozycja: ";
  for( list<int>::iterator i=lista.begin(); i!= lista.end(); ++i )
  {
     counter++;
     if(*i == liczba)   cout << setw(4) << counter;
  } Sleep(4000);
}
void exit(){
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),12);
    cout<<"Koniec programu!";
    Sleep(2000);
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),0);
}
  void pop_front()
{
    cout<<"Nastapi usuniecie liczby z poczatku listy";
    Sleep(2000);
    lista.pop_front();
}
void reverse(){
    cout<<"Nastapi odwrocenie kolejnosci liczb!";
    lista.reverse();
    Sleep(4000);
}
```

```
int main(){
   do{
      wyswietl();
      cout << "1.  Dodaj na poczatek"<<endl;
      cout << "2.  Dodaj na koniec"<<endl;
      cout << "3.  Usun pierwszy element"<<endl;
      cout << "4.  Usun ostatni element"<<endl;
      cout << "5.  Wyswietl rozmiar"<<endl;
      cout << "6.  Wyswietl maksymalny mozliwy rozmiar listy"<<endl;
      cout << "7.  Czy lista jest pusta?"<<endl;
      cout << "8.  Usun z listy"<<endl;
      cout << "9.  Posortuj"<<endl;
      cout << "10. Odwroc elementy"<<endl;
      cout << "11. Pokaz kolejny element"<<endl;
      cout << "12. Pokaz poprzedni element"<<endl;
      cout << "13. Zlicz ile razy wystapil element"<<endl;
      cout << "14. Znajdz pozycje na liscie"<<endl;
      cout << "15. Wyczysc zawartosc listy"<<endl;
      cout << "16. Wyjdz"<<endl;
      cout << endl << endl;
      cout << "> ";
      cin >> wybor;

switch (wybor)
         {
            case 1:  push_front(); break;
            case 2:  push_back();  break;
            case 3:  pop_front();  break;
            case 4:  pop_back();   break;
            case 5:  size();       break;
```

```
            case 6:  max_size();   break;
            case 7:  empty();      break;
            case 8:  remove();     break;
            case 9:  sort();       break;
            case 10: reverse();    break;
            case 11: nastepny();     break;
            case 12: poprzedni();    break;
            case 13: zlicz();      break;
            case 14: znajdz();     break;
            case 15: wyczysc();    break;
            case 16: exit();       break;
          default:
            cout<<"POMYLKA!";
            Sleep(2000);
            break;
          }
    }
    while(wybor!=16);
       return 0;
}
```

2.2 ZAPROGRAMOWANE OPERACJE

- Dodawanie elementów do listy z przodu lub tyłu,
- Usuwanie elementów z tablicy z przodu, tyłu lub danej liczby,
- Wyświetlanie zawartości listy, aktualnego rozmiaru listy, maksymalnego możliwego rozmiaru listy,
- Funkcja sprawdzająca czy lista jest pusta,
- Funkcja odwracająca listę, sortująca listę
- Funkcja pokazująca następny element i poprzedni
- Funkcja zliczająca elementy w liście
- Funkcja znajdująca pozycję na liście
- Funkcja czyszcząca listę

2.3 Wyniki działania programu

```
ZAWARTOSC LISTY:



1. Dodaj na poczatek
2. Dodaj na koniec
3. Usun pierwszy element
4. Usun ostatni element
5. Wyswietl rozmiar
6. Wyswietl maksymalny mozliwy rozmiar listy
7. Czy lista jest pusta?
8. Usun z listy
9. Posortuj
10. Odwroc elementy
11. Pokaz kolejny element
12. Pokaz poprzedni element
13. Zlicz ile razy wystapil element
14. Znajdz pozycje na liscie
15. Wyczysc zawartosc listy
16. Wyjdz


>
```

2.4 PSEUDOKOD

Dodaj element na początek listy

Utwórz nowy element

Do pola prev pierwszego elementu dodajemy wskaźnik nowego elementu.

Nowemu elementowi do date przypisujemy wpisaną liczbę całkowitą.

Nowemu elementowi do prev przypisujemy wartość NULL.

Nowemu elementowi do next przypisujemy wskaźnik do pierwszego elementu listy do pola next

Zwiększamy licznik elementów o jeden.

Dodaj element na koniec listy

Utwórz nowy element

Do pola next ostatniego elementu dodajemy wskaźnik do nowego elementu.

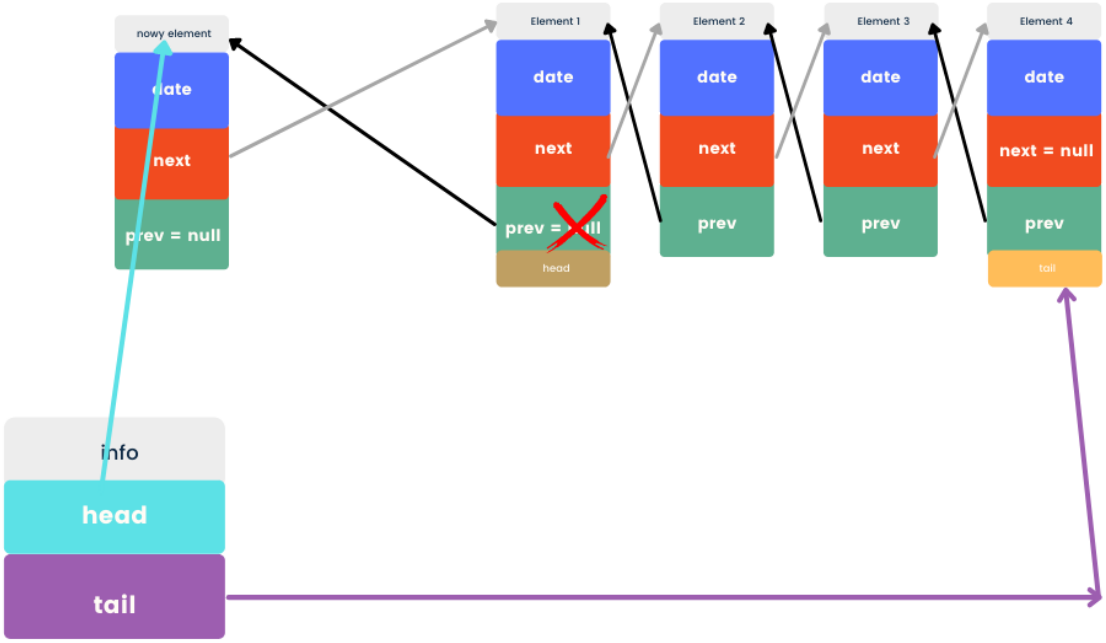
Nowemu elementowi do date przypisujemy wpisaną liczbę całkowitą.

Nowemu elementowi do next przypisujemy wartość NULL.

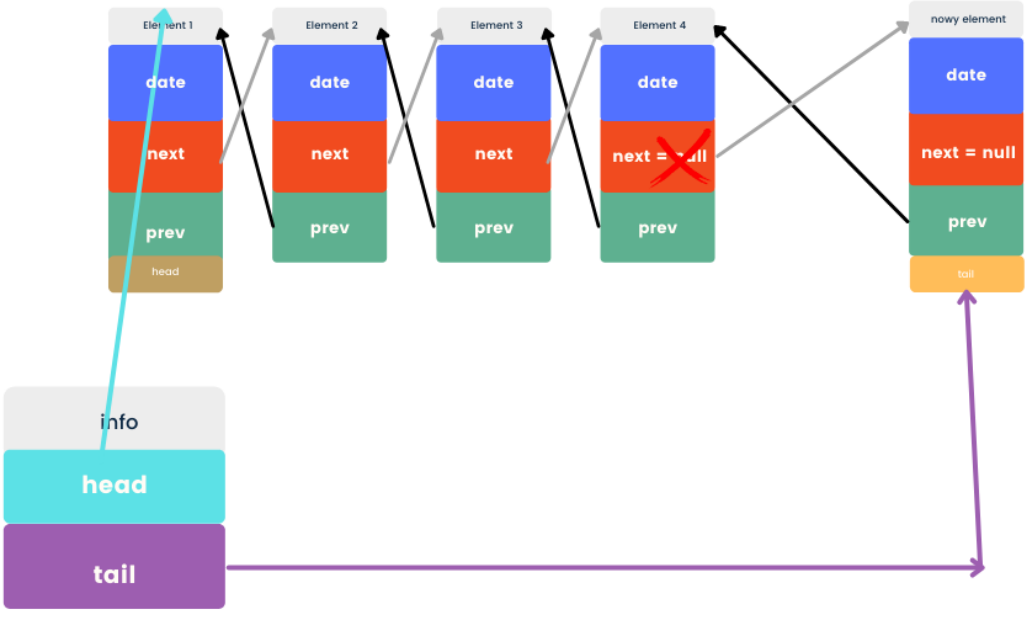
Nowemu elementowi do prev przypisujemy wskaźnik do pierwszego elementu listy do pola next

Zwiększamy licznik elementów o jeden.

2.5 SCHEMAT BLOKOWY



2 Schemat przedstawiający dodanie nowego elementu na początek listy



3 Schemat przedstawiający dodanie nowego elementu na koniec listy

3 DOKUMENTACJA Z DOŚWIADCZEŃ

Z dokumentacji z doświadczeń zostało usunięte wypisywanie Menu Głównego.

3.1 DODAWANIE ELEMENTÓW TABLICY

```
ZAWARTOSC LISTY:
  2  5  2  6  5  1  2  3  5  5
Liczb na liscie: 10
Maksymalna ilosc liczb na liscie: 768614336404564650


>
```

3.2 WYŚWIETLANIE ROZMIARU TABLICY ORAZ MAKSYMALNEGO MOŻLIWEGO ROZMIARU LISTY

```
ZAWARTOSC LISTY:
  2  5  2  6  5  1  2  3  5  5


>
```

3.3 WYNIK FUNKCJI CZY LISTA JEST PUSTA

```
ZAWARTOSC LISTY:
  2  5  2  6  5  1  2  3  5  5
> 7
Czy lista pusta? -> FALSE
```

3.4 Wynik działania sortowania

```
ZAWARTOSC LISTY:
  1   2   2   2   3    5   5   5   6


>
```

3.5 Wynik działania funkcji odwracającej tablice

```
ZAWARTOSC LISTY:
  6   5   5   5   3    2    2    2    1
> 11
Podaj liczbe: 2
```

3.6 Wynik działania funkcji pokazującej następny element

```
ZAWARTOSC LISTY:
  6   5   5   5   3    2    2    2    1
Podaj liczbe: 2
  2   2   1
>
```

3.7 Wynik działania funkcji zliczającej dana liczbę

```
ZAWARTOSC LISTY:
 1   2   2   2   3   5   5   5   6
Podaj liczbe: 5
Element 5 wystapil: 3 razy.
>
```

3.8 Wynik działania funkcji pokazujący pozycję danej liczby w liście

```
ZAWARTOSC LISTY:
 1  2  2  2  3  5  5  5  6
Podaj liczbe: 2
Pozycja:   2  3  4
>
```

4 Wnioski i podsumowanie

Największą zaletą zastosowania listy dwukierunkowej jest zdecydowanie to, że nie musimy deklarować z góry jej rozmiaru, tylko korzystamy z kolejnego możliwego do skorzystania miejsca w pamięci RAM. Dynamiczne dwukierunkowe listy liniowe mogą posłużyć do budowy stosu lub kolejek. W pewnych zastosowaniach listy dwukierunkowe mają pewną przewagę nad listami jednokierunkowymi - podwójne powiązanie każdego z elementów. Ma to znaczenie np. w przypadku systemów plików, gdzie takie listy reprezentują plik. W takim przypadku są one tworzone nie w pamięci operacyjnej komputera lecz w pamięci masowej, np. na twardym dysku.

5 SPIS TREŚCI