Sprawozdanie

Zadanie projektowe numer 3

Kacper Chmura
Inżynieria i analiza danych, grupa P1

1 OPIS PROBLEMU

Dokonaj implementacji struktury danych typu lista dwukierunkowa wraz z wszelkimi potrzebnymi operacjami charakterystycznimi dla tej struktury (inicjowanie struktury, dodawanie/usuwanie elementow, wyswietlanie elementow, zliczanie elementow/wyszukiwanie zadanego elementu itp.)

2 LISTY

2.1 LISTA DWUKIERUNKOWA

Rysunek 1 Lista dwukierunkowa

Lista dwukierunkowa – w każdym elemencie listy jest przechowywane odniesienie zarówno do następnika, jak i poprzednika elementu w liście. Taka reprezentacja umożliwia swobodne przemieszczanie się po liście w obie strony. Zaletą takiego rozwiazania jest fakt, iż mozemy poruszac sie zarowno do przodu jak i do tylu po naszje liscie co znacząco oszczedza czas. Wadą chociażby w stosunku do listy jednokierunkowej jest to ze lista dwukierunkowa zajmuje wiecej pamieci poniewaz musi przechowac informacje o swoim poprzedniku jak i nastepcy.

2.2 LISTA JEDNOKIERUNKOWA

Lista jednokierunkowa

Lista jednokierunkowa jest strukturą o dynamicznie zmieniającej się wielkości. Listę można opisać jako uszeregowany zbiór elementów. Każdy element zawiera jakieś dane oraz wskazuje na swojego następcę. Cechą listy jednokierunkowej jest to, że można przeglądać ją tylko w jedną stronę, od początku do końca.

2.3 LISTA DWU/JEDNOKIERUNKOWA CYKLICZNA

Listy te odrózniaja się od poprzedniczek tym ze mozemy je przechodzic cyklicznie. Dokonuje sie tego ustawiajac pierwszy element listy jako nastepce ostatniego oraz ustawiajac ostatni element listy jako poprzednika pierwszego.

Lista dwukierunkowa cykliczna

Lista jednokierunowa cykliczna

2.4 WSKAZNIKI I LICZNIK

Tworząc listę w pamięci zwykle dodatkowo rezerwuje się trzy zmienne dla jej obsługi:

-wskaźnik head – wskazuje pierwszy element listy (ang. head = głowa)

-wskaźnik tail – wskazuje ostatni element listy (ang. tail = ogon)

-licznik count – zlicza elementy na liście

2.5 LISTA A TABLICA

2.5.1 Porównanie

Tablica jest niewątpliwe alternatywa dla listy. Tak samo jak w liscie mozemy bez problemu dodac element na koniec tablicy, problem pojawia sie gdy chcemy dodac element, w środku bądź na początek tablicy wtedy konieczne jest przesunięcie wszystykich elementów, które mają występowac po naszym dodawanym elemencie. ten pojawia się rownież, gdy chcemy usunąć jakiś element wtedy też musimy przesuwać elementy, aby zapełnić luke powstałą w wyniku usuwania.

2.5.2 Wady i zalety

Zalety:

- prosta nawigacja wewnątrz tablicy

- szybki dostęp do konkretnego elementu o konkretynm numerze

- większa odporność na błędy

Wady:

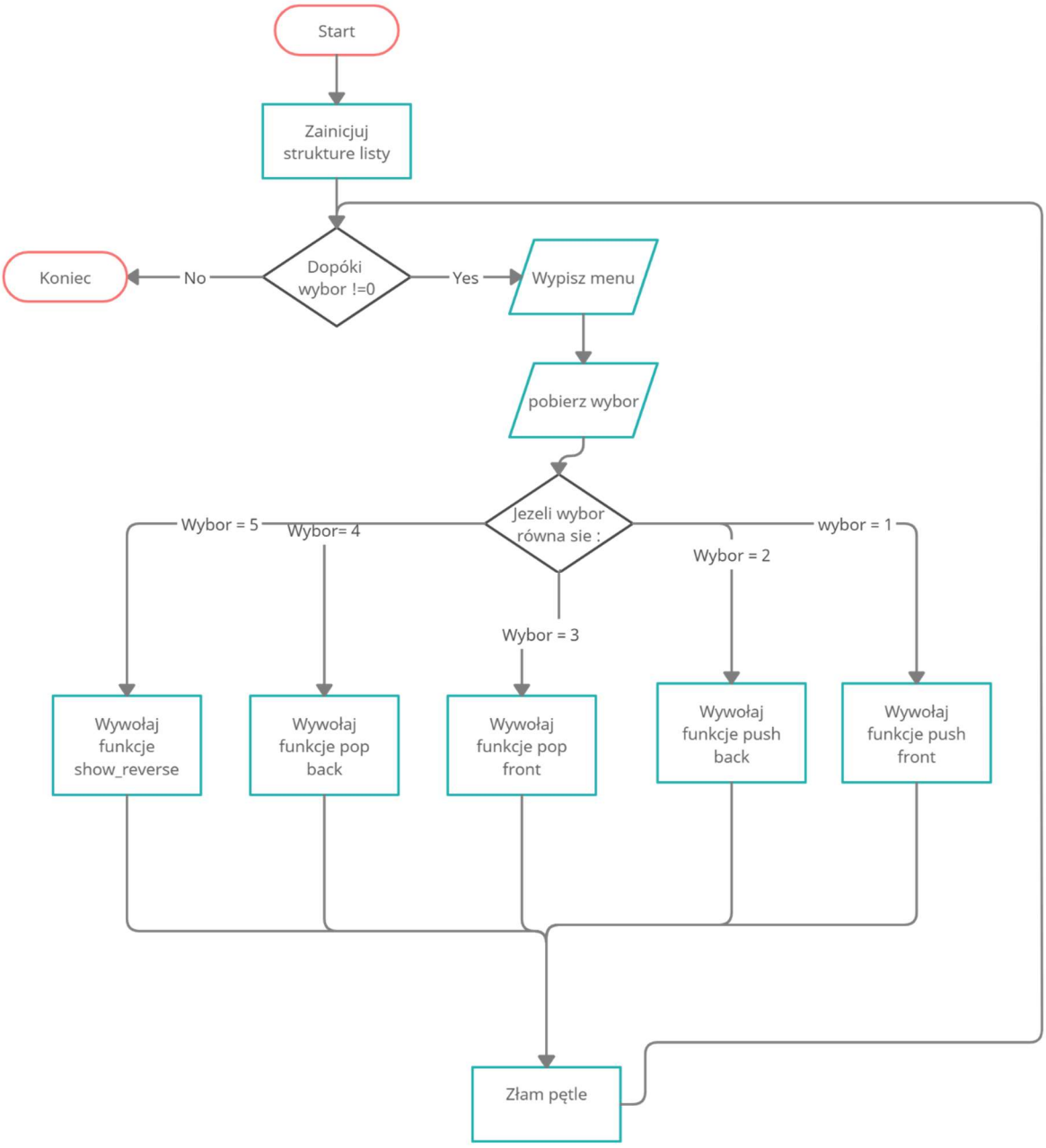
- niska elastyczność

- liniowa złożoność jeśli chodzi o operacje wstawiania i usuwania co za tym

Podsumywując tablica jest dobrą altenrnatywą dla listy jeśli bedzie ona wykorzystywana do przechowywania i odczytywania danych, na których nie będziemy wykonywać operacji. Jest szybsza w odczycie oraz znacznie stabilniejsza, lecz brak jej elastyczności jest znaczącą wadą.

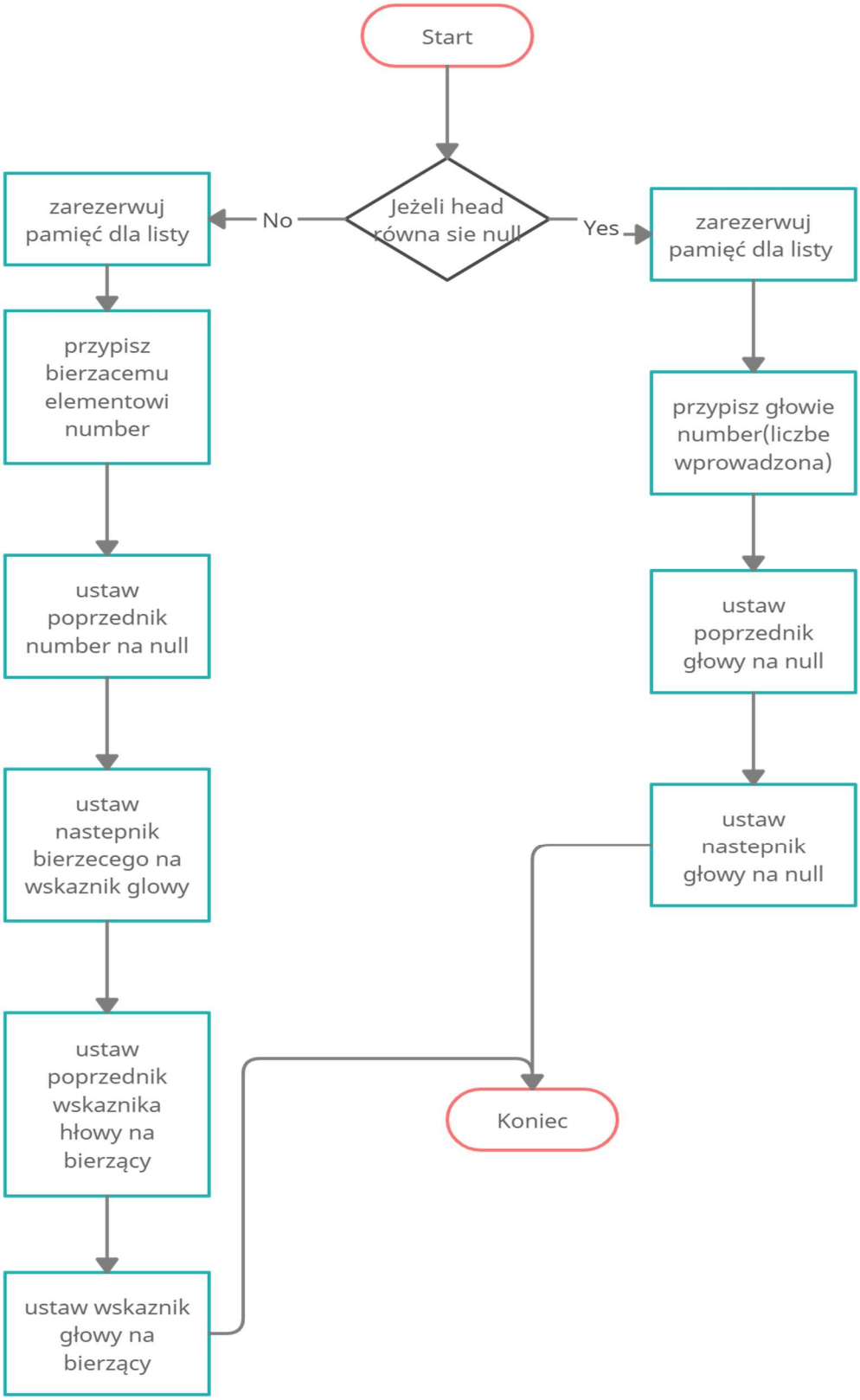
3 SCHEMATY BLOKOWE

3.1 SCHEMAT BLOKOWY

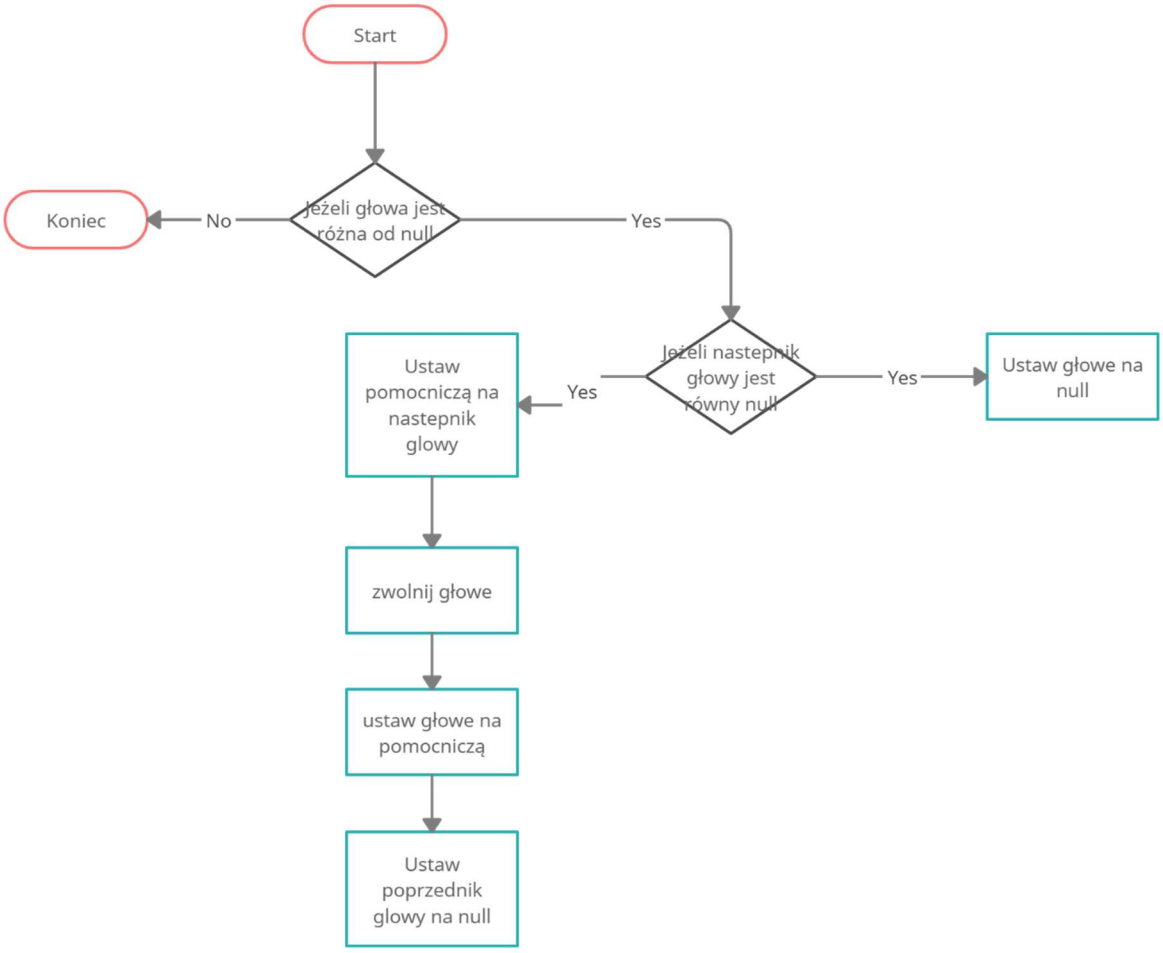


Schemat blokowy

3.2 SCHEMAT BLOKOWY FUNKCJI PUSH_FRONT



3.3 SCHEMAT BLOKOWY FUNKCJI POP_FRONT



4 IMPLEMENTACJA LISTY DWUKIERUNKOWEJ

4.1 UTWORZENIE LIST

```
typedef struct ListElement {
    int data;
    struct ListElement * previous;   //deklaracja struktury listy
    struct ListElement * next;
} ListElement_type;
```

4.2 DODAWANIE ELEMENTU NA POCZĄTEK

```
void push_front(ListElement_type **head, int number)
{

    if(*head==NULL) { //tylko wtedy gdy lista jest pusta czyli dodajemy pierwszy element
        *head = (ListElement_type *)malloc(sizeof(ListElement_type)); //alokacja pamieci na liste o danym rozmiarze
        (*head)->data = number; //liczba z klawiatury                  // jesli element jest pierwszy zostaje "glowa"
        (*head)->previous=NULL; //poniewaz jest to pierwszy element nie ma elementu nastepnego ani poprzedniego
        (*head)->next = NULL;
    } else {
        ListElement_type *current;
                current=(ListElement_type *)malloc(sizeof(ListElement_type));
                current->data=number;
                current->previous=NULL; //nadajemy null poprzednikowi wstawianego elementu poniewaz jest on teraz pierwszy
                current->next=(*head);
                (*head)->previous=current;
                *head=current;
    }

}
```

4.3 DODAWANIE ELEMENTU NA KONIEC

```
void push_back(ListElement_type **head, int number)
{


    if(*head==NULL) //tylko wtedy gdy lista jest pusta czyli dodajemy pierwszy element
    {
        *head = (ListElement_type *)malloc(sizeof(ListElement_type));//alokacja pamieci na liste o danym rozmiarze
        (*head)->data = number;
        (*head)->previous = NULL;
        (*head)->next = NULL;
    }else
    {
        ListElement_type *current=*head; //wskaznik glowy rowna sie wskaznikowi bierzacego elementu


        while (current->next != NULL) { //dopóki nastepnik bierzacego elementu jest różny od null
            current = current->next;    //Birzacy element rowna sie swojemu nastepnikowi
        }

        current->next = (ListElement_type *)malloc(sizeof(ListElement_type));
        current->next->data = number; //nadajemy liczbe z klawiatury
        current->next->previous=current;//ustawiamy poprzednik
        current->next->next = NULL;//ustawiamy jako nastepnik dodanego elementu null bo po nim juz nic nie ma
    }


}
```

4.4 USUWANIE ELEMENTU Z POCZĄTKU

```
void pop_front(ListElement_type **head)
{

    if (*head!=NULL) {//jesli nie ma elementow w liscie
        if((*head)->next==NULL) {//jesli jest jeden element w liscie
            *head=NULL;
        } else {
            ListElement_type *pom;
            pom=(*head)->next;//pom rowna sie nastepnik glowy
            free(*head);//zwolnienie glowy
            *head=pom;//nowa glowa rowna sie pom
            (*head)->previous=NULL;//poprzednik nowej glowy ustawiamy na null
        }

    }

}
```

4.5 USUWANIE ELEMENTU Z KOŃCA

```
void pop_back(ListElement_type **head) //wskaznik wskaznika
{

    if((*head)->next==NULL)//jezeli nastepnik glowy jest null
    {
        *head=NULL; //to przypisz wskaznikowi glowy null
    }else
    {
        ListElement_type *current=*head;
        while (current->next->next!= NULL) { //wykonuj dopóki nastepnik nastepnki bierzacego elementu jest rozny od null
        current = current->next; //przypisz bierzacemu elementowi jego nastepnik
        }
         free(current->next); //zwolnij nastepnik bierzacego
         current->next=NULL;
    }
}
```

4.6 WYŚWIETLANIE

```
void show(ListElement_type *head)
{
    printf(" ");
    if(head==NULL) printf("Lista jest pusta");//jesli nie ma glowy
    else
    {
        ListElement_type *current=head;
        do {
            printf("%i", current->data); //wypisz bierzacy element
            printf(" ");
            current = current->next; //wez za bierzacy element jego nastepnik
        }while (current != NULL);  //petla wykonuje sie dopoki bierzacy element bedzie rozny od null
    }
}
```

4.7 WYŚWIETLANIE ODWRÓCONE

```
void show_reverse(ListElement_type *head)
{
    printf(" ");
    if(head==NULL) printf("List is empty");
    else
    {
        ListElement_type *current=head;
        while (current->next != NULL) {
            current = current->next; //idziemy na koniec listy
        }


        do {
            printf("%i", current->data);
            printf(" ");
            current = current->previous;
        }while(current!=NULL);

    }
}
```

4.8 FUNKCJA MAIN I MENU

```
int main()
{
    ListElement_type *head;
    head = (ListElement_type *)malloc(sizeof(ListElement_type));
    head=NULL;


    int wybor;
    int number;
    int ile=0;
    int i;
    while(wybor!=0)
    {

    system("cls");
    printf("\nAktualny stan listy: ");
    show(head);

    printf("Co chcesz zrobic?\n");
    printf("1. Dodac element na poczatek listy\n");
    printf("2. Dodac element na koniec listy\n");
    printf("3. Usunac element z poczatku listy\n");
    printf("4. Usunac element z konca listy\n");
    printf("5. Wyswietl odwrocona liste\n");
    printf("0. Zakonczyc program.\n");



    scanf("%i", &wybor);
```

4.9 MENU CD.

```
switch (wybor)
{
case 0:
    return 0;
    break;

case 1:
    printf("Podaj ile liczb chcesz dodac");
    scanf("%d", &ile);
    for(i=0;i<ile;i++)
        {
        printf("Wpisz liczbe jaka chcesz dodac: ");
        scanf("%i", &number);
        push_front(&head, number);
        }

    break;
 case 2:
    printf("Podaj ile liczb chcesz dodac");
    scanf("%d", &ile);
    for(i=0;i<ile;i++)
        {
        printf("Wpisz liczbe jaka chcesz dodac: ");
        scanf("%i", &number);
        push_back(&head, number);
        }
    break;
case 3:
    printf("Podaj ile liczb chcesz dodac");
    scanf("%d", &ile);
    for(i=0;i<ile;i++)
        {
    pop_front(&head);
        }
```

5 PODSUMOWANIE

Listy dwukierunkowe choć nieco bardziej skąplikowane od jednokierunkowy, co za tym idzie szybsze, maja wadę jaką jest zajmowanie wiekszej ilosci pamieci ponieważ potrzebny jest jeszcze im dodatkowy rekord na „poprzednika-prev”. Implementując liste programista powinien odpowiedziec sobie na pytanie czy potrzebuje prędkości w odczycie danych czy woli postawic na wolniejsza liste, lecz z mniejszym zużyciem pamieci.