Programownie I
Wykład 9

dr inż. Adam Zielonka

Instytut Matematyki,
Politechnika Śląska

Gliwice 14.12.2018

Struktura

Struktura jest złożonym typem danych pozwalającym przechowywać różne (różnego typu) informacje. Pozwala w łatwy i przejrzysty sposób grupować dane.

struct

```
struct osoba{
    string imie;
    string nazwisko;
    int wiek;
};
```

Sama definicja struktury nie rezerwuje pamięci, a jedynie jest "przepisem" na to jak stworzyć obiekt nowego typu osoba, na który składa się jej imię, nazwisko i wiek.

Struktura

"Powołanie obiektów struktury do życia"

```
osoba student, wykladowca;
```

`student` jest obiektem typu osoba o identyfikatorze student, zajmuje w pamięci `sizeof( osoba)` bajtów.

lub

Obiekty struktury można stworzyć już podczas określania definicji:

```
struct osoba{
    string imie;
    string nazwisko;
    int wiek;
} jan, janusz, kornel;
```

Struktura

Inicjalizacja

Obiekty struktury można zainicjalizować podobnie jak typ prosty:

```
osoba ja{"Adam" , "Zielonka" , 40};
```

lub

danymi innego już istniejącego obiektu:

```
osoba ja{"Adam" , "Zielonka" , 40};
osoba wykladowaca=ja;
```

`wykladowca` zawiera kopie wartości struktury `ja`

Struktura

lub

poprzez dostęp do składowych (pól) obiektu struktury:

```
osoba jan;
jan.imie="Jan";
jan.nazwisko="Kowalski";
jan.wiek=19;
```

Etykieta (nazwa) struktury jest opcjonalna:

Struktura bez nazwy

```
struct { int wysokosc; int szerokosc; } wymiar1;
wymiar1.wysokosc = 4;
wymiar1.szerokosc = 9;
```

Zagnieżdżenie struktur

Składowymi jednej struktury mogą być inne struktury

Przykład

```
struct Punkt { int x; int y; };
struct Prostokat { Punkt w1; Punkt w2; };

Prostokat prostokatA{ Punkt{1,1},Punkt{3,5} };

Punkt p1, p2;
p1.x=0; p1.y=4;
p2.x=3; p2.y=9;
Prostokat prostokatB{ p1, p2 };
prostokatA.w1.x=10;
```

Struktury a funkcje

Struktura może być przekazywana przez argument do funkcji i może być przez funkcję zwracana.

Sktruktura jako argument

```
void drukuj(osoba o){
  cout << o.imie<<' '<<o.nazwisko << endl;
  cout << " lat:" << o.wiek << endl;
}
```

Funkcja zwraca strukturę

```
Prostokat kwadrat(int bok){
  Punkt w1{0,0};
  Punkt w2{bok, bok};
  return Prostokat{w1,w2};
}
```

Wskaźniki do struktur

Wskaźnik

```
osoba kibic{"Janusz" , "Znawca" , 65 };
osoba *Kibic = &kibic;
kibic.imie = "Janek";

(*Kibic).wiek = 32;

Kibic->nazwisko = "Niepokorny";
```

Przykład

```
using Osoba = osoba*;
osoba sudent{"Ewa","Kot",23};
Osoba Student = &student;
Student->nazwisko.replace(0,1,"Mł");
```

Funkcja jako składowa struktury

Oprócz danych składowymi struktury mogą być funkcje.

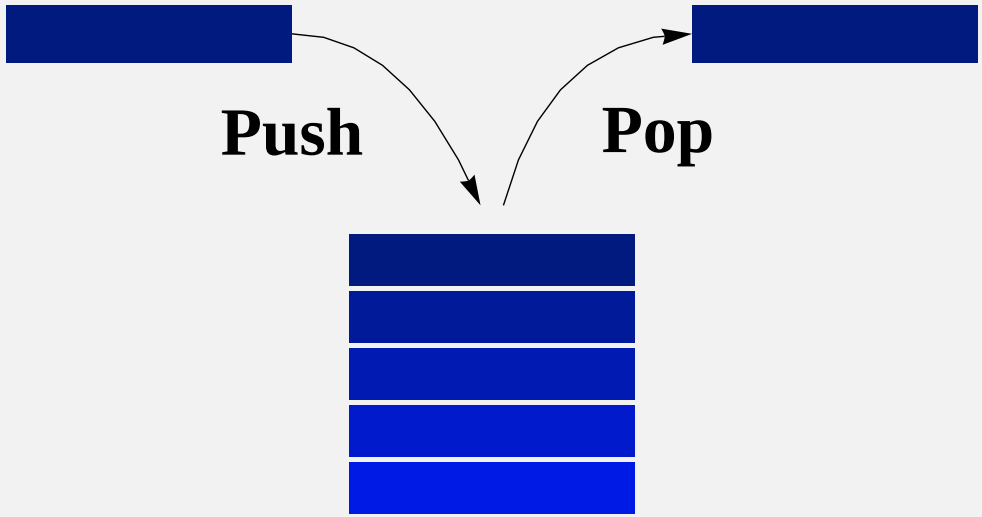
```
struct osoba
{
    string imie;
    string nazwisko;
    int wiek;
    void drukuj();
};
void osoba::drukuj()
{
    string napis{ imie };
    napis+=' ';
    napis+=nazwisko;
    cout << napis;
}
```

Przykład 1

STOS

Stos (*stack*)

Stos abstrakcyjna liniowa struktura danych typu **LIFO** (*Last In, First Out*) ostatni na wejściu, pierwszy na wyjściu.



Interfejs publiczny stosu:

```
push( stos , wartosc ) - dodaje wartość na stos
pop( stos )            - zdejmuje ze stosu
top( stos )            - zwraca element ze szczytu
isEmpty( stos )        - czy pusty
count( stos )          - ilość elementów na stosie
```

Określenie segmentu

```
struct segment
{
    int element;            //wartość elementu
    segment *poprzedni;     //wskaźnik poprzedniego
};

using Segment=segment*;
```

Deklaracja funkcji

```
Segment push( Segment ostatni, int liczba );

Segment pop( Segment ostatni );

int top( Segment ostatni );

bool isEmpty( Segment ostatni );

int count( Segment ostatni );
```

Funkcje

```
Segment push( Segment ostatni, int liczba )
{
    Segment nowy = new segment;
    nowy -> element = liczba;
    nowy -> poprzedni = ostatni;
    return nowy;
}
Segment pop( Segment ostatni )
{
    Segment poprzedni;
    if (ostatni == nullptr) return nullptr;
    poprzedni = ostatni->poprzedni;
    delete ostatni;
    return poprzedni;
}
```

Funkcje

```
int top( Segment ostatni )
{
    return ostatni->element;
}

bool isEmpty( Segment ostatni )
{
    if(ostatni==nullptr)
    return true;
    return false;
}
```

Funkcje

```
int count( Segment ostatni )
{
    int n=0;
    while( ostatni != nullptr )
        {
            n++;
            ostatni = ostatni->poprzedni;
        }
        return n;
}
```

Użycie

```
int main(){
        Segment stos = nullptr;

        stos = push( stos, -3 );
        stos = push( stos, -4 );
        stos = push( stos, 12 );

        cout << "Wysokosc stosu: ";
        cout << count( stos ) << endl;

        while(!isEmpty(stos)){
            cout << top( stos ) << endl;
            stos = pop( stos );
        }
        system("PAUSE");
}
```

Przykład 2

STOS

wykorzystanie stosu do pobrania pobrania dowolnie długiej tablicy liczb naturalnych.

Przykład

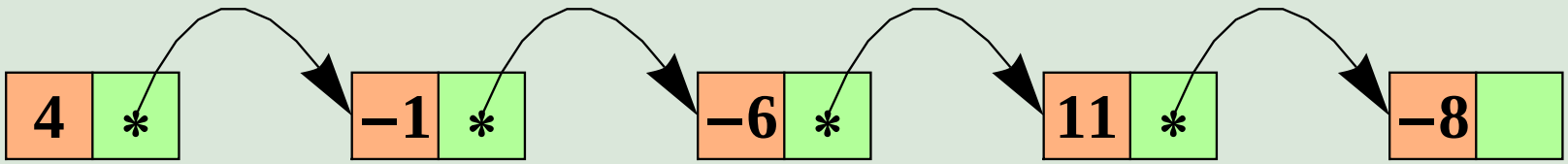
```
int main()
{
    Segment stosLiczb = nullptr;
    int a;
    cin >> a;
    while( a > 0 )
    {
        stosLiczb=push( stosLiczb, a );
        cin >> a;
    }
    int n=count(stosLiczb);
```

c.d.

```
    int *tablica = new int[n];
    while( n > 0 )
    {
        tablica[--n]=top( stosLiczb );
        stosLiczb=pop(stosLiczb);
    }
        system("PAUSE");
}
```

Przykład 3

LISTA jednokierunkowa



Lista jednokierunkowa

Struktura pojedynczego elementu listy

```
struct osoba{
    string imie;
    string nazwisko;
    int wiek;

    osoba* kolejna;

    void drukuj();
};
using Osoba=osoba*;
```

drukuj()

```
void osoba::drukuj(){
    std::cout<<imie<<' '<<nazwisko<<' '<<wiek;
}
```

Struktura listy

```
struct ListaOsob
{

    osoba* pierwsza;

    void dodaj( osoba );

    void usun(int nr);

    void sortuj();

    void drukujListe();
};
```

Funkcja dodaj

```
void ListaOsob::dodaj( osoba o )
{
    //stworzenie nowego obiektu osoba
    Osoba Nowa=new osoba;

    Nowa->imie = o.imie;
    Nowa->nazwisko = o.nazwisko;
    Nowa->wiek = o.wiek;

    //sprawdzenie czy już coś jest w liście
    if(pierwsza == nullptr)
            pierwsza = Nowa;
    else
    {
```

Funkcja dodaj c.d.

```
    else
    {
        //ustawienie wskaźnika na końcu listy
        Osoba biezaca=pierwsza;

        while(biezaca->kolejna!=nullptr)
             biezaca=biezaca->kolejna;

        //powiazanie nowego elementu z ostatnim
        //dotychczas elementem listy

        biezaca->kolejna=nowa;
    }
}
```

drukujListe

```
void ListaOsob::drukujListe()
{
    Osoba biezaca = pierwsza;
    while (biezaca != nullptr)
    {
        biezaca->drukuj();
        std::cout << std::endl;
        biezaca = biezaca->kolejna;
    }
}
```

usun

```
void ListaOsob::usun(int nr){
    Osoba wczesniejsza, biezaca;
    int ind = 0;
    if (pierwsza == nullptr) return;
    if (nr == 0){
        biezaca = pierwsza->kolejna;
        delete pierwsza;
        pierwsza = biezaca;
    }
    else{
        biezaca = pierwsza;
        wczesniejsza = pierwsza;
        while (biezaca->kolejna != nullptr&&ind < nr){
            wczesniejsza = biezaca;
            biezaca = biezaca->kolejna;
            ind++;
        }
        if (biezaca != nullptr){
            wczesniejsza->kolejna = biezaca->kolejna;
            delete biezaca;
        }
    }
}
```

Użycie

```
int main()
{
    ListaOsob lista;
    lista.pierwsza = nullptr;
    lista.dodaj(osoba{"Adam","Lis",32});
    lista.dodaj(osoba{ "Wiktor","Kot",22 });
    lista.dodaj(osoba{"Michal","Ptak",23});
    lista.drukujListe();
    cout << "--------" << endl;
    lista.usun(2);
    lista.drukujListe();
    system("PAUSE");
}
```