

Informatyka 2 (ES1D300 017)

Politechnika Białostocka - Wydział Elektryczny
Elektrotechnika, semestr III, studia stacjonarne I stopnia
Rok akademicki 2019/2020

Wykład nr 4 (22.10.2019)

dr inż. Jarosław Forenc

Plan wykładu nr 4

- Wskaźniki
 - operacje na wskaźnikach
- Dynamiczny przydział pamięci
 - funkcje calloc, malloc, free
 - przydział pamięci na wektor
 - dynamiczny przydział pamięci na macierz
- Dynamiczne struktury danych
 - stos, kolejka, lista, drzewo

Operacje na wskaźnikach (1)

- **Przypisanie** - wskaźnikowi można przypisać:
 - adres zmiennej (nazwa zmiennej poprzedzona znakiem &)
 - inny wskaźnik
 - tablicę (nazwa to jej adres)

```
int tab[3] = {1, 2, 3};  
int x = 10, *ptr1, *ptr2, *ptr3;  
  
ptr1 = &x;  
ptr2 = ptr1;  
ptr3 = tab;
```

- Typ adresu i wskaźnika muszą być zgodne

Operacje na wskaźnikach (2)

- **Pobranie wartości (dereferencja)**
 - otrzymanie wartości przechowywanej w pamięci, w miejscu wskazywanym przez wskaźnik
 - operator pobrania wartości (dereferencji, wyluskania): *

```
int x = 10, *ptr, y;  
  
ptr = &x;  
y = *ptr;  
printf("Wartosc x i y: %d\n",y);
```

```
Wartosc x i y: 10
```

Operacje na wskaźnikach (3)

- Pobranie adresu wskaźnika
 - tak jak inne zmienne, także wskaźniki posiadają wartość i adres

```
int x = 10, *ptr;

ptr = &x;
printf("Adres zmiennej x: %p\n", ptr);
printf("Adres wskaźnika ptr: %p\n", &ptr);
```

```
Adres zmiennej x: 002CF920
Adres wskaźnika ptr: 002CF914
```

Operacje na wskaźnikach (4)

- Dodanie liczby całkowitej do wskaźnika
 - przed dodaniem liczby całkowitej jest ona mnożona przez liczbę bajtów zajmowanych przez wartość wskazywanego typu

```
int tab[5] = {0,1,2,3,4};

printf("Adres tab: %p\n", tab);
printf("Adres tab+2: %p\n", (tab+2));
printf("tab[0]: %d\n", *tab);
printf("tab[2]: %d\n", *(tab+2));
```

```
Adres tab: 002CFC60
Adres tab+2: 002CFC68
tab[0]: 0
tab[2]: 2
```

Operacje na wskaźnikach (5)

- Zwiększenie wskaźnika (inkrementacja)
 - do wskaźnika można dodać 1 lub zastosować operator ++
 - wskaźnik będzie pokazywał na kolejny element tablicy

```
int tab[5] = {0,1,2,3,4}, *ptr;

ptr = tab;
printf("tab[0]: %d\n", *ptr);
ptr++;
printf("tab[1]: %d\n", *ptr);
ptr = ptr + 1;
printf("tab[2]: %d\n", *ptr);
```

```
tab[0]: 0
tab[1]: 1
tab[2]: 2
```

Operacje na wskaźnikach (5)

- Zwiększenie wskaźnika (inkrementacja)
 - do wskaźnika można dodać 1 lub zastosować operator ++
 - wskaźnik będzie pokazywał na kolejny element tablicy

```
int tab[5] = {0,1,2,3,4};

printf("tab[0]: %d\n", *tab);
tab++;
printf("tab[1]: %d\n", *tab);
```

error C2105: '++' needs l-value

Operacje na wskaźnikach (6/7)

- **Odjęcie liczby całkowitej od wskaźnika**
 - działa analogicznie jak dodanie liczby całkowitej do wskaźnika, ale wskaźnik musi być lewym operandem odejmowania
- **Zmniejszenie wskaźnika (dekrementacja)**
 - działa analogicznie jak inkrementacja

Operacje na wskaźnikach (8)

- **Odejmowanie wskaźników**
 - różnicę między dwoma wskaźnikami oblicza się najczęściej dla wskaźników należących do tej samej tablicy
 - różnica ta określa jak daleko od siebie znajdują się elementy tablicy

```
int tab[5] = {0,1,2,3,4}, *ptr;  
ptr = tab + 3;  
printf("Roznica: %d\n", ptr-tab);
```

```
Roznica: 3
```

- różnica wskaźników należących do dwóch różnych tablic może spowodować błąd w programie

Operacje na wskaźnikach (9)

- **Porównanie wskaźników**
 - porównanie może dotyczyć tylko wskaźników tego samego typu
 - w porównaniach stosowane są standardowe operatory: `<`, `>`, `<=`, `>=`, `==`, `!=`

```
int tab[5] = {0,1,2,3,4}, *ptr;  
ptr = tab + 2;  
ptr--;  
--ptr;  
if (tab == ptr)  
    printf("Ten sam wskaznik\n");  
else  
    printf("Inny wskaznik\n");
```

```
Ten sam wskaznik
```

Dynamiczny przydział pamięci w języku C

- Kiedy stosuje się dynamiczny przydział pamięci?
 - gdy rozmiar tablicy będzie znany dopiero podczas wykonania programu a nie podczas jego kompilacji
 - gdy rozmiar tablicy jest bardzo duży (np. największy rozmiar tablicy elementów typu `char` w języku C wynosi ok. 1 000 000)
- Do dynamicznego przydziału pamięci stosowane są funkcje:
 - `calloc()`
 - `malloc()`
- Przydział pamięci następuje w obszarze **sterty** (stosu zmiennych dynamicznych)
- Przydzieloną pamięć należy zwolnić wywołując funkcję:
 - `free()`

Dynamiczny przydział pamięci w języku C

```
CALLOC stdlib.h  
void *calloc(size_t num, size_t size);
```

- Przydziela blok pamięci o rozmiarze `num*size` (mogący pomieścić tablicę `num`-elementów, każdy rozmiaru `size`)
- Zwraca wskaźnik do przydzielonego bloku pamięci
- Jeśli pamięci nie można przydzielić, to zwraca wartość `NULL`
- Przydzielona pamięć jest inicjowana zerami (bitowo)
- Zwracaną wartość wskaźnika należy rzutować na właściwy typ

```
int *tab;  
tab = (int *) calloc(10, sizeof(int));
```

Dynamiczny przydział pamięci w języku C

```
MALLOC stdlib.h  
void *malloc(size_t size);
```

- Przydziela blok pamięci o rozmiarze określonym parametrem `size`
- Zwraca wskaźnik do przydzielonego bloku pamięci
- Jeśli pamięci nie można przydzielić, to zwraca wartość `NULL`
- Przydzielona pamięć nie jest inicjowana
- Zwracaną wartość wskaźnika należy rzutować na właściwy typ

```
int *tab;  
tab = (int *) malloc(10*sizeof(int));
```

Dynamiczny przydział pamięci w języku C

```
FREE stdlib.h  
void *free(void *ptr);
```

- Zwalnia blok pamięci wskazywany parametrem `ptr`
- Wartość `ptr` musi być wynikiem wywołania funkcji `calloc()` lub `malloc()`

```
int *tab;  
tab = (int *) calloc(10, sizeof(int));  
/* ... */  
free(tab);
```

Dynamiczny przydział pamięci na wektor

```
#include <stdio.h>  
#include <stdlib.h>  
  
int main(void)  
{  
    int *tab, i, n, x;  
    float suma = 0.0;  
  
    printf("Podaj ilosc liczb: ");  
    scanf("%d", &n);  
  
    tab = (int *) calloc(n, sizeof(int));  
    if (tab == NULL)  
    {  
        printf("Nie mozna przydzielic pamieci.\n");  
        exit(-1);  
    }  
}
```

Dynamiczny przydział pamięci na wektor

```
for (i=0; i<n; i++) /* wczytanie liczb */
{
    printf("Podaj liczbe nr %d: ",i+1);
    scanf("%d",&x);
    tab[i] = x;
}

for (i=0; i<n; i++)
    suma = suma + tab[i];

printf("Srednia %d liczb wynosi %f\n",n,suma/n);

free(tab);

return 0;
}
```

Dynamiczny przydział pamięci na wektor

```
for (i=0; i<n; i++)
{
    printf("Podaj lic
    scanf("%d",&x);
    tab[i] = x;
}

for (i=0; i<n; i++)
    suma = suma + tab[i];

printf("Srednia %d liczb wynosi %f\n",n,suma/n);

free(tab);

return 0;
}
```

```
Podaj ilosc liczb: 5
Podaj liczbe nr 1: 1
Podaj liczbe nr 2: 2
Podaj liczbe nr 3: 3
Podaj liczbe nr 4: 4
Podaj liczbe nr 5: 5
Srednia 5 liczb wynosi 3.000000
```

Dynamiczny przydział pamięci na wektor

- Wczytanie liczb bezpośrednio do wektora `tab`

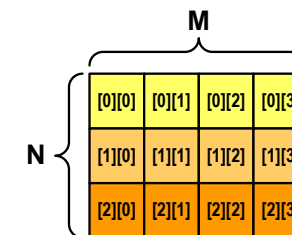
```
for (i=0; i<n; i++) /* wczytanie liczb */
{
    printf("Podaj liczbe nr %d: ",i+1);
    scanf("%d",&tab[i]);
}
```

- Inny sposób odwołania do elementów wektora `tab`

```
for (i=0; i<n; i++) /* wczytanie liczb */
{
    printf("Podaj liczbe nr %d: ",i+1);
    scanf("%d", (tab+i));
}
```

Dynamiczny przydział pamięci na macierz

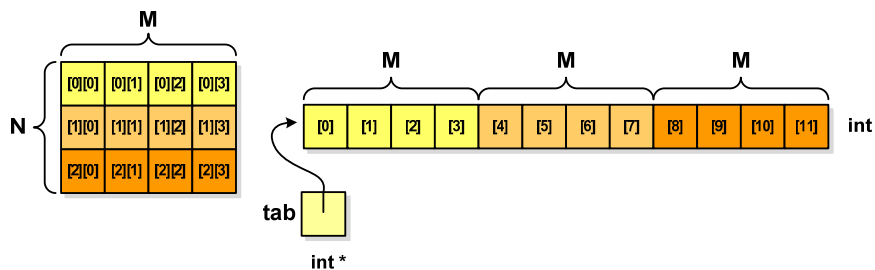
- Funkcje `calloc()` i `malloc()` umożliwiają bezpośrednio przydział pamięci tylko na wektor elementów
- Dynamiczny przydział pamięci na macierz wymaga zastosowania specjalnych metod
- Przydzielamy pamięć na macierz zawierającą **N-wierszy** i **M-kolumn**



Dynamiczny przydział pamięci na macierz (1)

- Wektor N×M-elementowy
- Przydział pamięci:

```
int *tab = (int *) calloc(N*M, sizeof(int));
```



Dynamiczny przydział pamięci na macierz (1)

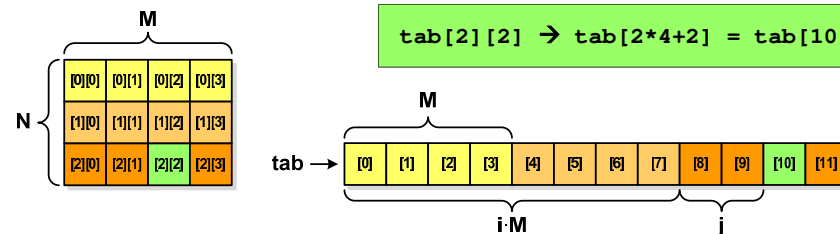
- Odwwołanie do elementów macierzy:

`tab[i*M+j]`

lub

`*(tab+i*M+j)`

`tab[2][2] → tab[2*4+2] = tab[10]`



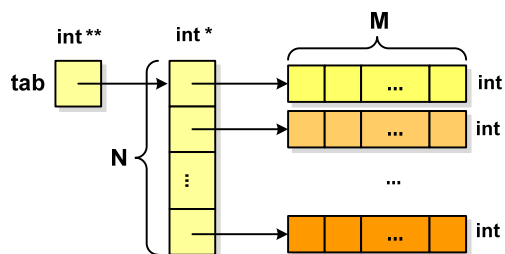
- Zwolnienie pamięci:

```
free(tab);
```

Dynamiczny przydział pamięci na macierz (2)

- N-elementowy wektor wskaźników + N-wektorów M-elementowych
- Przydział pamięci:

```
int **tab = (int **) calloc(N, sizeof(int *));  
for (i=0; i<N; i++)  
    tab[i] = (int *) calloc(M, sizeof(int));
```



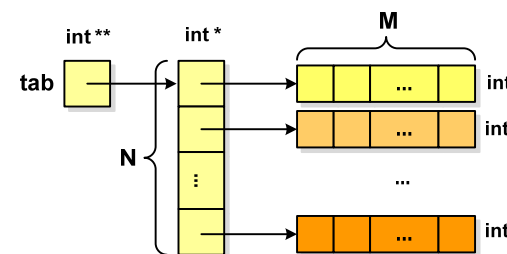
Dynamiczny przydział pamięci na macierz (2)

- Odwwołania do elementów macierzy:

`tab[i][j]`

- Zwolnienie pamięci:

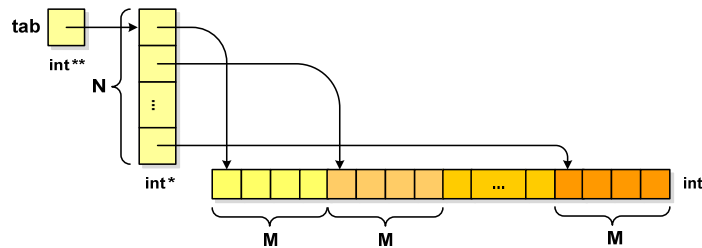
```
for (i=0; i<N; i++)  
    free(tab[i]);  
free(tab);
```



Dynamiczny przydział pamięci na macierz (3)

- N-elementowy wektor wskaźników + wektor N×M-elementowy
- Przydział pamięci:

```
int **tab = (int **) malloc(N*sizeof(int *));
tab[0] = (int *) malloc(N*M*sizeof(int));
for (i=1; i<N; i++)
    tab[i] = tab[0]+i*M;
```

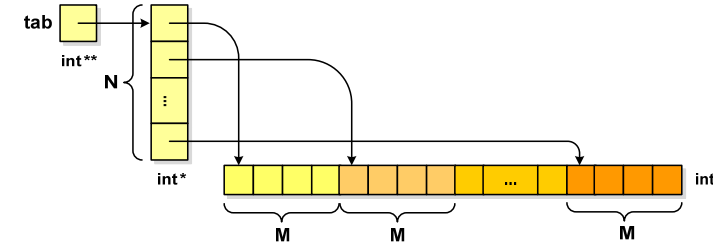


Dynamiczny przydział pamięci na macierz (3)

- Odwołania do elementów macierzy:
- Zwolnienie pamięci:

`tab[i][j]`

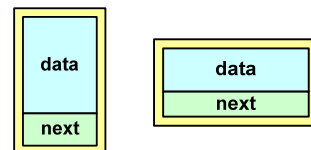
```
free(tab[0]);
free(tab);
```



Dynamiczne struktury danych

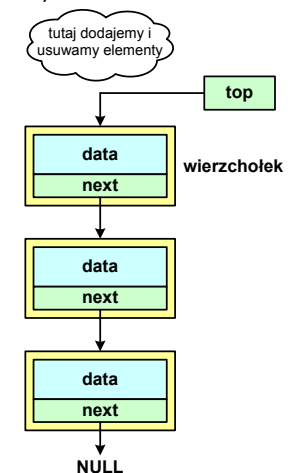
- **Dynamiczne struktury danych** - struktury danych, którym pamięć jest przydzielana i zwalniana w trakcie wykonywania programu
 - stos, kolejka
 - lista (jednokierunkowa, dwukierunkowa, cykliczna)
 - drzewo
- Elementy w dynamicznych strukturach danych są strukturami składającymi się z „użytecznych” danych (**data**) oraz z jednego lub kilku wskaźników (**next**) zawierających adresy innych elementów

```
struct element
{
    typ data;
    struct element *next;
};
```



Stos

- **stos** (ang. stack) - struktur składająca się z elementów, z których każdy posiada tylko adres następnika
- dostęp do danych przechowywanych na stosie jest możliwy tylko w miejscu określonym mianem **wierzchołek** stosu (ang. top)
- wierzchołek stosu jest jedynym miejscem, do którego można dołączać lub z którego można usuwać elementy
- każdy składnik stosu posiada wyróżniony element (**next**) zawierający adres następnego elementu
- wskaźnik ostatniego elementu stosu wskazuje na adres pusty (**NULL**)
- podstawowe operacje na stosie to:
 - dodanie elementu do stosu - funkcja **push()**
 - zdjęcie elementu ze stosu - funkcja **pop()**



Notacja polska

- **Notacja polska** (zapis przedrostkowy, Notacja Łukasiewicza) jest to sposób zapisu wyrażeń arytmetycznych, podający najpierw operator, a następnie argumenty
- Wyrażenie arytmetyczne:

$4 / (1 + 3)$

ma w notacji polskiej postać:

$/ 4 + 1 3$

- Wyrażenie powyższe nie wymaga nawiasów, ponieważ przypisanie argumentów do operatorów wynika wprost z ich kolejności w zapisie
- Notacja ta była podstawą opracowania tzw. **odwrotnej notacji polskiej**

Odwrotna notacja polska

- **Odwrotna Notacja Polska** - ONP (ang. Reverse Polish Notation, RPN) jest sposobem zapisu wyrażeń arytmetycznych, w którym operator umieszczany jest **po** argumentach
- Wyrażenie arytmetyczne:

$(1 + 3) / 2$

ma w odwrotnej notacji polskiej postać:

$1 3 + 2 /$

- Odwrotna notacja polska została opracowana przez australijskiego naukowca **Charlesa Hamblina**

Odwrotna notacja polska

- Obliczenie wartości wyrażenia przy zastosowaniu ONP wymaga:
 - zamiany notacji konwencjonalnej (nawiasowej) na ONP (algorytm Dijkstry nazywany stacją rozrządową)
 - obliczenia wartości wyrażenia arytmetycznego zapisanego w ONP
- W obu powyższych algorytmach wykorzystywany jest stos
- Przykład:

- wyrażenie arytmetyczne:

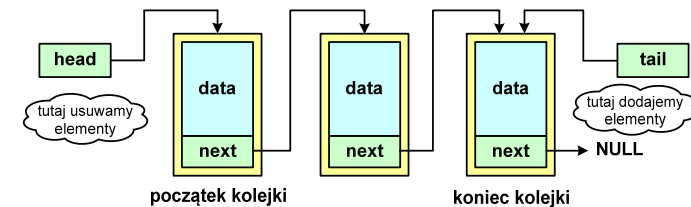
$(2 + 1) * 3 - 4 * (7 + 4)$

- ma w odwrotnej notacji polskiej postać:

$2 1 + 3 * 4 7 4 + * -$

Kolejka

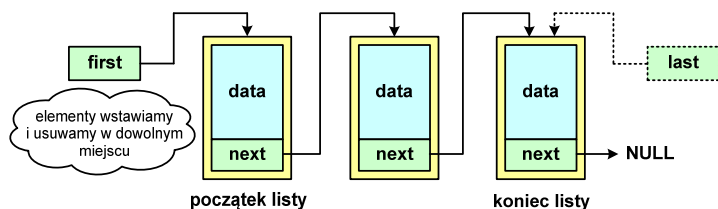
- **Kolejka** - składa się z liniowo uporządkowanych elementów
- Elementy dołączane są tylko na końcu kolejki (wskaźnik **tail**)
- Elementy usuwane są tylko z początku kolejki (wskaźnik **head**)



- Powiązanie między elementami kolejki jest takie samo, jak w stosie
- Kolejka nazywana jest stosem **FIFO** (ang. **First In First Out**)

Lista jednokierunkowa

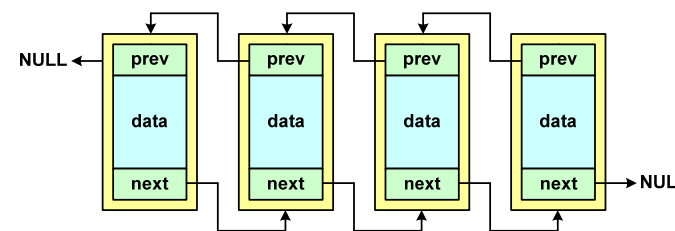
- Organizacja listy jednokierunkowej podobna jest do organizacji stosu i kolejki
- Dla każdego składnika (poza ostatnim) jest określony następny składnik (lub poprzedni - zależnie od implementacji)



- Zapamiętywany jest wskaźnik tylko na pierwszy element listy (**first**) lub wskaźniki na pierwszy (**first**) i ostatni element listy (**last**)
- Elementy listy można dołączać/usuwać w dowolnym miejscu listy

Lista dwukierunkowa

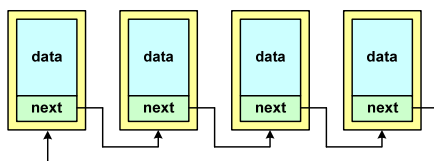
- Każdy węzeł posiada adres następnika, jak i poprzednika
- W strukturze tego typu wygodne jest przechodzenie pomiędzy elementami w obu kierunkach (od początku do końca i odwrotnie)



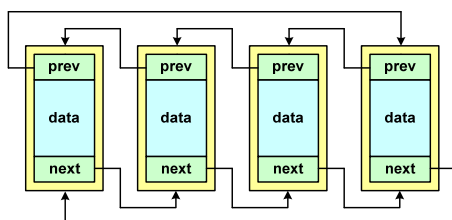
Lista cykliczna

- Powstaje z listy jednokierunkowej lub dwukierunkowej, poprzez połączenie ostatniego element z pierwszym

Jednokierunkowa:

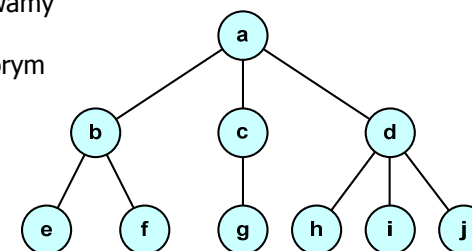


Dwukierunkowa:



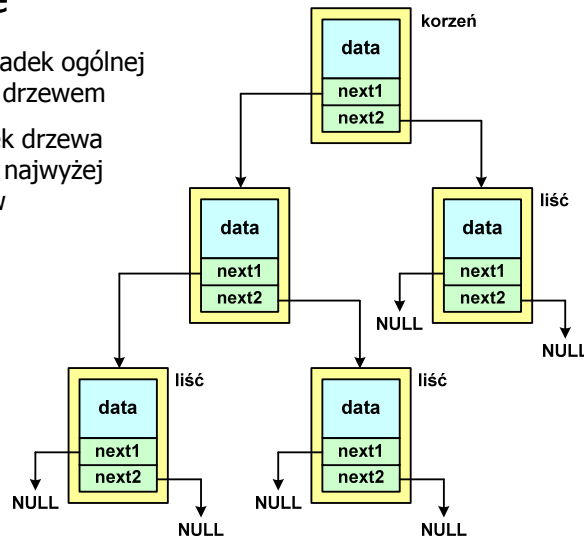
Drzewo

- Najbardziej ogólna dynamiczna struktura danych, może być reprezentowane graficznie na różne sposoby
- Na górze znajduje się **korzeń drzewa** (a)
- Skojarzone z korzeniem poddrzewa połączone są z nim liniami zwanymi **gałęziami drzewa**
- Potomkiem węzła **w** nazywamy każdy, różny od **w**, węzeł należący do drzewa, w którym **w** jest korzeniem
- Węzeł, który nie ma potomków, to **liść drzewa**



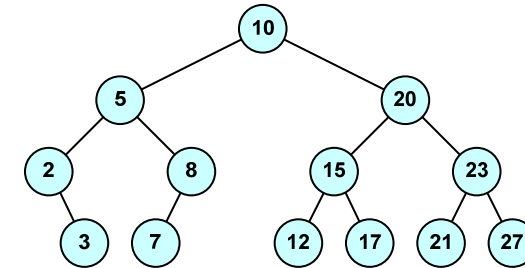
Drzewo binarne

- Szczególny przypadek ogólnej struktury zwanej drzewem
- Każdy wierzchołek drzewa binarnego ma co najwyżej dwóch potomków



Binarne drzewo wyszukiwawcze

- Drzewo binarne, w którym dla każdego węzła w_i :
 - wszystkie klucze w lewym poddrzewie węzła w_i są mniejsze od klucza w węźle w_i
 - wszystkie klucze w prawym poddrzewie węzła w_i są większe od klucza w węźle w_i



- Zaleta: szybkość wyszukiwania informacji

Koniec wykładu nr 4

Dziękuję za uwagę!