Informatyka 2

Politechnika Białostocka - Wydział Elektryczny

Elektrotechnika, semestr III, studia stacjonarne I stopnia

Rok akademicki 2008/2009

Wykład nr 1 (08.10.2008)

dr inż. Jarosław Forenc

Dane podstawowe

- dr inż. Jarosław Forenc
- Politechnika Białostocka, Wydział Elektryczny,
  Katedra Elektrotechniki Teoretycznej i Metrologii
  ul. Wiejska 45D, 15-351 Białystok
  WE-204
- e-mail: jarekf@pb.edu.pl
- tel. (0-85) 746-93-97
- http://www.we.pb.edu.pl/~jforenc
  - Dydaktyka - slajdy prezentowane na wykładzie
- http://www.we.pb.edu.pl/~jforenc/archiwum.html
  - materiały z poprzedniego semestru

Przedmiot „Informatyka"

- Informatyka 1
  - semestr II
  - wykład: 15 h (J. Forenc)
  - pracownia specjalistyczna: 30 h (R. Bycul, J. Forenc, P. Myszkowski)
  - ECTS: 4 pkt.
  - kod przedmiotu: ES1A200 009
- Informatyka 2
  - semestr III
  - wykład: 15 h (J. Forenc)
  - pracownia specjalistyczna: 30 h (J. Forenc, W. Walendziuk)
  - ECTS: 4 pkt.
  - kod przedmiotu: ES1A300 016

Program wykładu (1/2)

1. Wskaźniki, operacje na wskaźnikach, wskaźniki i tablice. Dynamiczny przydział pamięci. Dynamiczne struktury danych: stos, kolejka, lista. Przykłady zastosowań.
2. Programowanie obiektowe w języku C++. Operacje wejścia/wyjścia. Klasy i obiekty. Składniki klasy: dane i funkcje. Prawa dostępu do składników klasy. Definiowanie funkcji składowych klasy. Konstruktory i destruktory. Przeładowanie operatorów. Dziedziczenie. Funkcje wirtualne.
3. Operacje na wektorach i macierzach. Mnożenie macierzy, normy wektorów i macierzy, wyznacznik macierzy.
4. Sortowanie. Klasyfikacje algorytmów sortowania. Algorytmy sortowania: przez proste wstawianie, przez proste wybieranie, bąbelkowe, szybkie (Quick-Sort).

Program wykładu (2/2)

5. System operacyjny. Funkcje i zadania systemu operacyjnego. Struktura i właściwości systemów Windows i Linux. Zarządzanie zadaniami, pamięcią i dyskami.
6. Sieci komputerowe. Podział sieci, topologie. Model ISO/OSI. Technologie, protokoły, urządzenia. Zasada działania sieci Internet.
7. Relacyjne bazy danych. Podstawowe pojęcia, organizacja i zasady wykorzystania.
8. Metody sztucznej inteligencji.

Literatura (1/2)

1. B.W. Kernighan, D.M. Ritchie: „Język ANSI C". WNT, Warszawa, 2007.
2. J. Grębosz: „Symfonia C++ standard". Tom 1 i 2. Edition 2000, Warszawa, 2006.
3. B. Stroustrup: „Język C++". WNT, Warszawa, 2004.
4. K. Barteczko: „Praktyczne wprowadzenie do programowania obiektowego w języku C++". Wydawnictwo Lupus, Warszawa, 1994.
5. B. Eckel: „Thinking in C++. Edycja polska". Helion, Gliwice, 2002.
6. D. Kincaid, W. Cheney: „Analiza numeryczna". WNT, Warszawa, 2006.
7. T.H. Cormen i in.: „Wprowadzenie do algorytmów". WNT, Warszawa, 2007.
8. P. Wróblewski: „Algorytmy, struktury danych i techniki programowania". Helion, Gliwice, 2003.

Literatura (2/2)

9. N. Wirth: „Algorytmy + struktury danych = programy”. WNT, Warszawa, 2004.
10. A. Silberschatz i in.: „Podstawy systemów operacyjnych”. WNT, Warszawa, 2006.
11. K. Krysiak: „Sieci komputerowe. Kompendium”. Wydanie II. Helion, Gliwice, 2005.
12. H. Garcia-Molina i in.: „Systemy baz danych. Pełny wykład”. WNT, Warszawa, 2006.
13. M. Whitehorn, B. Marklyn: „Relacyjne bazy danych”. Helion. Gliwice, 2003.
14. D.E. Goldberg: „Algorytmy genetyczne i ich zastosowania”. WNT, Warszawa, 2003.
15. Z. Michalewicz. D.B. Fogel: „Jak to rozwiązać, czyli nowoczesna heurystyka”. WNT, Warszawa, 2006.

Zaliczenie wykładu

- Kolokwium pisemne
  - Termin nr 1: zostanie ustalony w styczniu 2009
  - Termin nr 2: zostanie ustalony w styczniu 2009
- 4 pytania, za każde pytanie można otrzymać od 0 pkt. do 10 pkt.
- Przykładowe pytania zaliczeniowe na stronie www: połowa stycznia 2009
- Punktacja:
  - 37 pkt. - 40 pkt. - ocena: 5
  - 33 pkt. - 36 pkt. - ocena: 4+
  - 29 pkt. - 32 pkt. - ocena: 4
  - 25 pkt. - 28 pkt. - ocena: 3+
  - 21 pkt. - 24 pkt. - ocena: 3
  - poniżej 21 pkt. - ocena: 2

Plan wykładu nr 1

- Wskaźniki
  - co to jest wskaźnik?
  - deklaracja wskaźnika
  - operacje na wskaźnikach
- Wskaźniki a tablice
- Dynamiczny przydział pamięci
  - język C: funkcje calloc, malloc i free
  - język C++: operatory new i delete
- Dynamiczne struktury danych
  - stos

Wskaźniki - gdzie już występowały?

- Funkcja scanf:
  - funkcja scanf wymaga podania adresów zmiennych pod którymi będą zapamiętane dane odczytane z klawiatury, np.

```
int a, b;
float c;
char txt[10];
scanf("%d %d %f %s", &a, &b, &c, txt);
```

- Struktury:
  - przy omawianiu struktur wspominane jest, że jeśli posługujemy się wskaźnikami na struktury, to do odwołania do pól struktury stosujemy operator pośredniego wyboru pola: ->

```
struct punkt { int x, y; } p1, *p2;
p2 = &p1;
p1.x = 10;     /* operator bezpośredniego wyboru pola */
p2->x = 10;    /* operator pośredniego wyboru pola */
```

Wskaźniki - gdzie już występowały?

- Przekazywanie parametrów do funkcji przez wskaźnik:
  - przekazywanie parametrów do funkcji przez wskaźnik polega na tym, że do funkcji przekazywane są adresy zmiennych
  - wszystkie operacje w funkcji wykonywane są zatem na zmiennych z funkcji wywołującej (poprzez adres tych zmiennych)

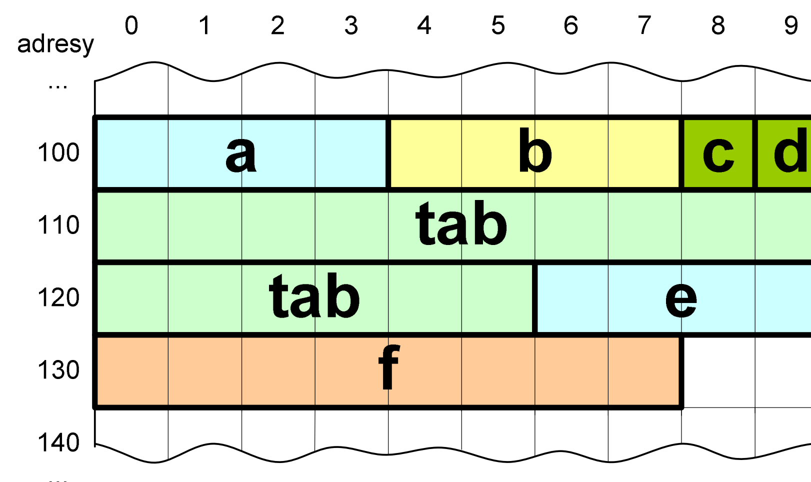
```
#include <stdio.h>

void fun(int *a, int *b)
{
   printf("fun1:  a =%3d, b =%3d \n",*a,*b);
   *a = 10; *b = 10;
   printf("fun2:  a =%3d, b =%3d \n",*a,*b);
}

int main()
{
   int a = 20, b = 20;

   printf("main1: a =%3d, b =%3d \n",a,b);
   fun(&a,&b);
   printf("main2: a =%3d, b =%3d \n",a,b);
   return 0;
}
```

```
main1: a = 20, b = 20
fun1:  a = 20, b = 20
fun2:  a = 10, b = 10
main2: a = 10, b = 10
```

Co to jest wskaźnik?

- Wskaźnik jest zmienną mogącą zawierać adres obszaru pamięci, a w szczególności adres innej zmiennej (obiektu)
- Rozpatrzmy następujące deklaracje zmiennych i sposób przechowywania ich w pamięci komputera:

```
int a;
float b;
char c, d;
int tab[4];
int e;
double f;
```

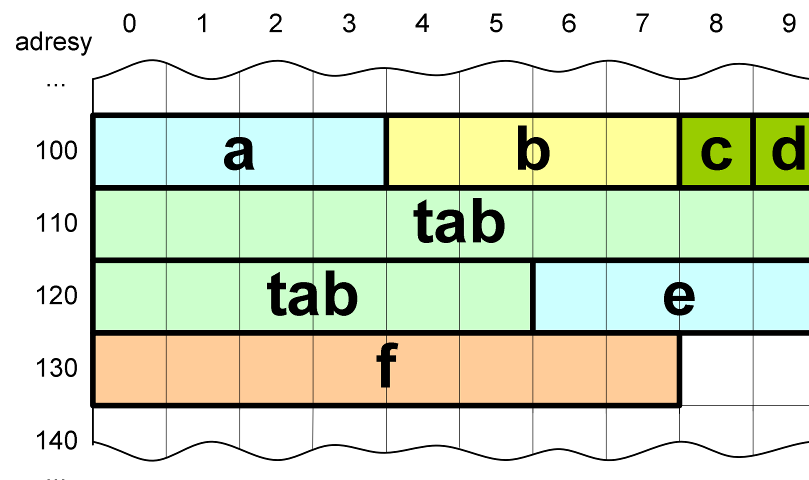


  - wszystkie zmienne przechowywane są w pamięci komputera i zależnie od typu zmiennej zajmują określoną liczbę bajtów
  - każda zmienna oprócz nazwy, której używa programista, ma także adres
  - komputer nie posługuje się nazwami zmiennych tylko ich adresami

Co to jest wskaźnik?

- Wskaźnik jest zmienną mogącą zawierać adres obszaru pamięci, a w szczególności adres innej zmiennej (obiektu)
- Rozpatrzmy następujące deklaracje zmiennych i sposób przechowywania ich w pamięci komputera:

```
int a;
float b;
char c, d;
int tab[4];
int e;
double f;
```



- w powyższym przykładzie poszczególne zmienne mają następujące adresy:

a → 100 (int, 4 bajty) b → 104 (float, 4 bajty)
c → 108 (char, 1 bajt) d → 109 (char, 1 bajt)
e → 126 (int, 4 bajty) f → 130 (double, 8 bajtów)
tab → 110 (tablica, int, 4 × 4 bajty)

Wskaźniki

Deklaracja wskaźnika:

- adres określa miejsce w pamięci, nie mówi natomiast nic na temat rozmiaru wskazywanego obiektu
- deklarując wskaźnik (zmienną wskazującą) musimy podać typ obiektu na jaki on wskazuje
- deklaracja zmiennej wskazującej wygląda tak samo jak deklaracja każdej innej zmiennej, tylko że jej nazwa poprzedzona jest symbolem gwiazdki (*****):

  `typ *nazwa_zmiennej;` lub `typ* nazwa_zmiennej;`

Przykład:

```
int *ptr;
```

- powyższy zapis oznacza deklarację zmiennej wskaźnikowej na typ int
- mówimy, że zmienna ptr jest typu: wskaźnik na zmienną typu int, a zatem może przechowywać adresy zmiennych a i e typu int z wcześniejszego przykładu

Wskaźniki

- do przechowywania adresu zmiennej f typu double trzeba zadeklarować zmienną typu: wskaźnik na zmienną typu double, czyli np.

```
double *ptrd;
```

- można konstruować wskaźniki na dane dowolnego typu łącznie z typami wskaźnik na... lub wskaźnik na wskaźnik na...

```
char **wsk;
```

- w powyższym przykładzie wsk jest typu wskaźnik na wskaźnik na typ char
- można deklarować tablice wskaźników

Uwaga:

- deklaracja wskaźnika wydziela tylko obszar pamięci dla przechowywania wskaźnika, tj. adresu (dla komputerów PC są to 2 lub 4 bajty, zależnie od tego czy adresy są „bliskie" - near, czy „dalekie" - far), nie jest natomiast przydzielany obszar dla danych, na które ma wskazywać wskaźnik

Wskaźniki

Przypisywanie wskaźnikom adresów innych zmiennych:

- adresy zmiennych, które można przypisać wskaźnikom, tworzy się za pomocą jednoargumentowego operatora pobierania adresu & (znanego z funkcji scanf):

`int a = 10;`	- deklaracja zmiennej typu int i nadanie jej wartości 10
`int *ptr;`	- deklaracja wskaźnika na zmienną typu int
`ptr = &a;`	- przypisanie zmiennej ptr adresu zmiennej a (od tej chwili zmienna ptr wskazuje na zmienną a; a - zmienna typu int, &a - adres zmiennej a)

Odwołanie do zmiennej poprzez jej adres:

- mając adres zmiennej można „dostać się" do jej wartości używając tzw. operatora wyłuskania (odwołania pośredniego) oznaczanego przez gwiazdkę (*)

`*ptr = 20;`	- jest równoważne z tym, że zmiennej a z powyższego przykładu przypisujemy wartość 20

Wskaźniki

Odwołanie do zmiennej poprzez jej adres:

- jeśli ptr jest wskaźnikiem na jakiś obiekt, to *ptr jest obiektem (zawartością pamięci pod adresem zawartym w ptr, interpretowaną zgodnie z typem obiektu, na który wskazuje ptr)

Przykład:

```
int a = 10;              /* a - zmienna typu int */
int *ptr = &a;           /* ptr - wskaźnik na zmienną typu int */
int **ptr1 = &ptr;       /* ptr1 - wskaźnik na wskaźnik na zmienną typu int */
a = 20;                  /* przypisanie zmiennej a wartości 20 */
*ptr = 20;               /* j.w. */
*(*ptr1) = 20;           /* j.w. */
```

- efekt końcowy wykonania trzech ostatnich instrukcji jest taki sam

Wskaźniki

Uwaga:

- zmiennej wskaźnikowej można nadać wartość zerową, co oznacza, że wskaźnik na nic nie wskazuje - należy stosować wtedy stałą o nazwie NULL

```
int *ptr;
ptr = NULL;
```

Wskaźniki - przykład

```
Wartosc zmiennej a        a    = 10
Adres zmiennej a          &a   = 2293644
Adres zmiennej a          ptr  = 2293644
Wartosci zmiennej a       *ptr = 10
Wartosci zmiennej a       *ptr = 20
Wartosc zmiennej a        a    = 20
```

```
#include <stdio.h>
int main()
{
   int a = 10;          /* deklar
   int *ptr;            /* deklar

   printf("Wartosc zmiennej a        a    = %d\n",a);
   printf("Adres zmiennej a          &a   = %d\n",&a);

   ptr = &a;            /* zmiennej ptr przypisujemy adres zmiennej a */

   printf("Adres zmiennej a          ptr  = %d\n",ptr);
   printf("Wartosci zmiennej a       *ptr = %d\n",*ptr);

   *ptr = 20;           /* zmiana wartosci zmiennej a */

   printf("Wartosci zmiennej a       *ptr = %d\n",*ptr);
   printf("Wartosc zmiennej a        a    = %d\n",a);

   return 0;
}
```

Przykładowe operacje na wskaźnikach

Wskaźniki

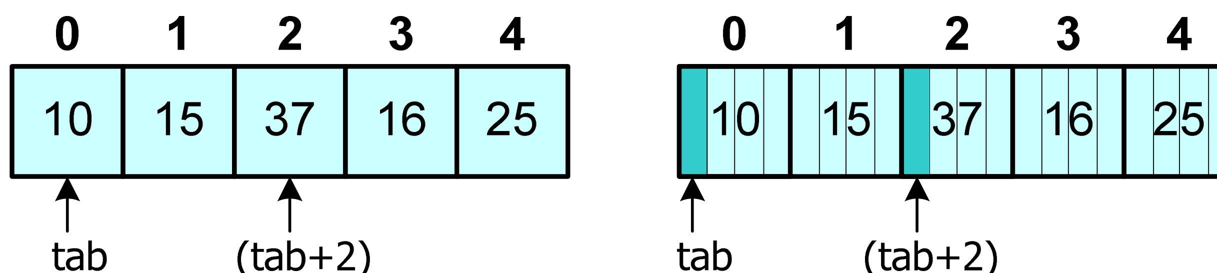
Operacje na wskaźnikach:

- na wskaźnikach dopuszczalne są następujące operacje:
  - przypisywanie wartości innych wskaźników oraz zera (NULL)
  - dodawanie do wartości wskaźnika liczb całkowitych
  - obliczanie różnicy wskaźników
  - porównywanie wartości wskaźnika z zerem (wartością stałej NULL)
  - porównywanie wskaźników między sobą
  - przekształcanie wskaźników (konwersje wskaźnikowe)
- dodanie do wskaźnika lub odjęcie wartości całkowitej 1 oznacza zwiększenie lub zmniejszenie adresu o wartość zależną od tego na jaki typ wskazuje wskaźnik
- jeśli mamy zadeklarowany wskaźnik: typ *ptr; to wyrażenie (ptr + n) wskazuje na obszar odległy o n * sizeof(typ) bajtów od adresu zawartego w ptr
- do operacji na wskaźnikach można używać operatorów inkrementacji (++) lub dekrementacji (--)
- odejmować można wskaźniki wskazujące na ten sam typ danych, różnica wskaźników jest liczbą elementów zawartych pomiędzy odejmowanymi elementami

Wskaźniki a tablice

- nazwa tablicy jest adresem zerowego elementu tablicy

```
int tab[5] = {10,15,37,16,25};
```



- **tab** - nazwa tablicy będąca adresem jej zerowego elementu
- ***tab** - wartość zerowego elementu tablicy (10) - równoważny zapis: tab[0]
- ***(tab+1)** - wartość pierwszego elementu tablicy (15) - równoważny zapis: tab[1]
- ***(tab+2)** - wartość drugiego elementu tablicy (37) - równoważny zapis: tab[2]
- ...
- ogólnie: tab[i] jest równoważne: *(tab+i)

Wskaźniki a tablice

- w zapisie *(tab+i) nawiasy są konieczne, gdyż operator * ma bardzo wysoki priorytet

Przykład:

```
int tab[5] = {10,15,37,16,25},x;

x = *(tab+2);
printf("x = %d",x);              /*  x = 37 */

x = *tab+2;
printf("x = %d",x);              /*  x = 12 */
```

x = *(tab+2); jest równoważne x = tab[2];

x = *tab+2; jest równoważne x = tab[0] + 2;

Dynamiczny przydział pamięci

- deklaracja tablicy w języku C wymaga, aby jej rozmiar był znany już na etapie kompilacji programu
- w przypadku, gdy rozmiar tablicy będzie znany dopiero podczas wykonania programu, pamięć należy przydzielić dynamicznie
- do tego celu służą funkcje dynamicznego przydziału pamięci:
    - calloc
    - malloc

  i zwalniania pamięci:
    - free
- działanie powyższych funkcji polega na przydzielaniu i zwalnianiu bloków pamięci w obszarze stosu zmiennych dynamicznych

Dynamiczny przydział pamięci

CALLOC stdlib.h

```
void *calloc(size_t n, size_t size);
```

- funkcja calloc przydziela blok pamięci o rozmiarze n*size (czyli blok pamięci mogący pomieścić tablicę n-elementów podanego rozmiaru size) i zwraca wskaźnik do tego bloku
- jeśli pamięci nie można przydzielić, to funkcja zwraca wartość NULL
- przydzielona pamięć jest inicjowana zerami

MALLOC stdlib.h

```
void *malloc(size_t size);
```

- funkcja malloc przydziela blok pamięci o rozmiarze określonym parametrem size i zwraca wskaźnik do tego bloku
- jeśli pamięci nie można przydzielić, to funkcja zwraca wartość NULL
- przydzielona pamięć nie jest inicjowana

Dynamiczny przydział pamięci

- zwracaną przez funkcje calloc i malloc wartość wskaźnika należy rzutować na właściwy typ, np. przydział pamięci na 10 elementową tablicę elementów typu int:

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

- do elementów takiej tablicy odwołujemy się tak samo jak do elementów zwykłej tablicy (ale <u>tylko w przypadku tablic jednowymiarowych</u> !!!)

- przydzieloną pamięć należy po wykorzystaniu zwolnić funkcją free

FREE — stdlib.h

```
void *free(void *ptr);
```

- funkcja free zwalnia blok pamięci wskazywany parametrem ptr
- wartość ptr musi być wynikiem wcześniejszego wywołania funkcji calloc lub malloc

Dynamiczny przydział pamięci - przykład

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int   *tab, i, n, x;
   float suma = 0.0;

   printf("Podaj ilosc liczb: "); scanf("%d"

   tab = (int *) calloc(n,sizeof(int));
   if (tab == NULL)
   {
      printf("Nie mozna przydzielic pamieci.\n"); exit(-1);
   }

   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;
}
```

```
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
```

Przydział pamięci na n-liczb typu int, wczytanie liczb, obliczenie średniej

Dynamiczny przydział pamięci

- w programie z poprzedniego slajdu można zrezygnować ze zmiennej x i liczby wczytywać bezpośrednio do tablicy tab - pętla wczytująca liczby będzie miała wtedy postać:

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

- wczytywanie liczb można zapisać jeszcze w inny sposób ((tab+i) jest adresem i-tego elementu tablicy):

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

Dynamiczny przydział pamięci

Dynamiczny przydział pamięci na tablicę dwuwymiarową (macierz):

- przydzielamy dynamicznie pamięć na tablicę zawierającą N-wierszy i M-kolumn

```
int tab[N][M];
```

Metoda 1 (wektor N×M-elementowy):

- przydzielamy pamięć na wektor N×M-elementowy:

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

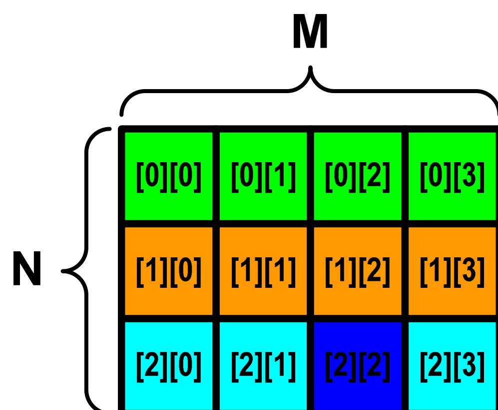
- po wykorzystaniu wektora, przydzieloną pamięć zwalniamy funkcją free

Dynamiczny przydział pamięci

Dynamiczny przydział pamięci na tablicę dwuwymiarową (macierz):

- zamiast standardowego odwołania do elementów macierzy: `tab[i][j]` stosujemy odwołania do odpowiednich elementów wektora:

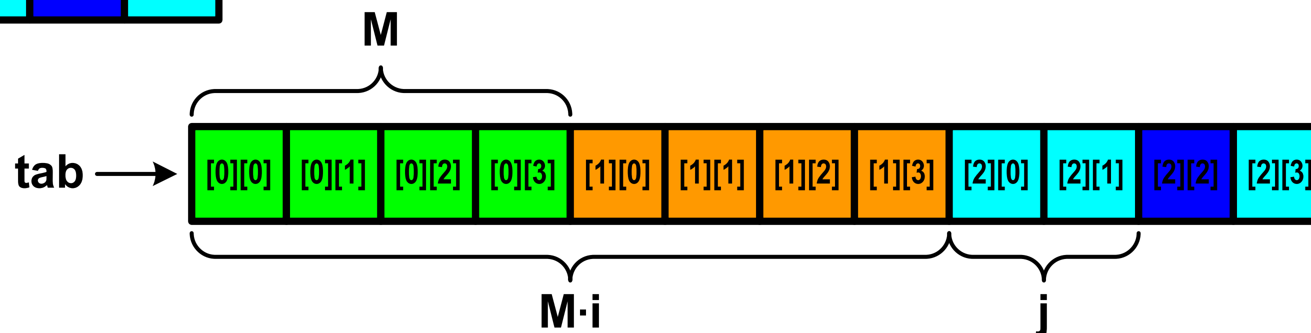
  `*(tab+M*i+j)` lub `tab[M*i+j]`



Przykład:

- tablica tab, dla której N=3, M=4
- odwołujemy się do elementu tab[2][2]:

  *(tab+4*2+2)



Dynamiczny przydział pamięci - macierz

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 4
#define M 6

int main()
{
    int i,j,*tab;

    tab = (int *) calloc(N*M,sizeof(int));
    srand(time(NULL));
    for (i=0;i<N;i++)
        for (j=0;j<M;j++)
            *(tab+M*i+j) = rand()%100;

    for (i=0;i<N;i++)
    {
        for (j=0;j<M;j++)
            printf("%4d",tab[M*i+j]);
        printf("\n");
    }

    free(tab);
    return 0;
}
```

```
  77  59  49  41  82  62
  89  80  86  52   7  64
  51  43   4  29  23  98
  52  10   2  43  37  95
```

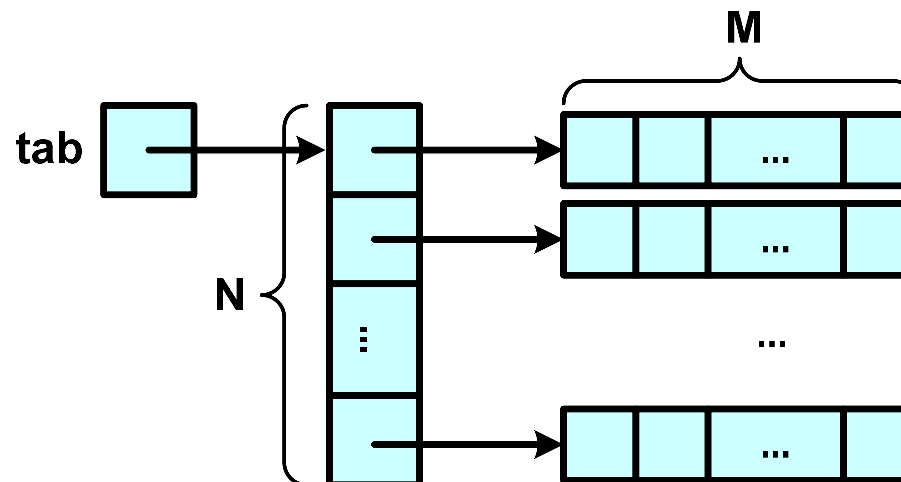
Przydział pamięci na tablicę NxM, wygenerowanie i wyświetlenie liczb

Dynamiczny przydział pamięci

Metoda 2 (wskaźnik na tablicę wskaźników):

- przydzielamy pamięć na N-elementowy wektor wskaźników na typ int, a następnie do kolejnych elementów tego wektora zapisujemy adresy M-elementowych wektorów liczb typu int (pamięć na wektory jest także przydzielana dynamicznie)

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



Dynamiczny przydział pamięci

Metoda 2 (wskaźnik na tablicę wskaźników):

- odwołania do elementów takiej tablicy mają taką samą postać jak w przypadku „zwykłych" tablic: `tab[i][j]`
- po wykorzystaniu tablicy należy zwolnić przydzieloną pamięć

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

- w pierwszej kolejności zwalniamy pamięć przydzieloną na N wektorów, każdy o rozmiarze M, a następnie zwalniamy pamięć przydzieloną na N-elementowy wektor wskaźników na typ int

Dynamiczny przydział pamięci - macierz (1/2)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 4
#define M 6

int main()
{
    int i, j, **tab;

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

    srand(time(NULL));
    for (i=0;i<N;i++)
        for (j=0;j<M;j++)
            tab[i][j] = rand()%100;

    for (i=0;i<N;i++)
    {
        for (j=0;j<M;j++)
            printf("%4d",tab[i][j]);
        printf("\n");
    }
```

Przydział pamięci na tablicę NxM, wygenerowanie i wyświetlenie liczb

Dynamiczny przydział pamięci - macierz (2/2)

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

    return 0;
}
```

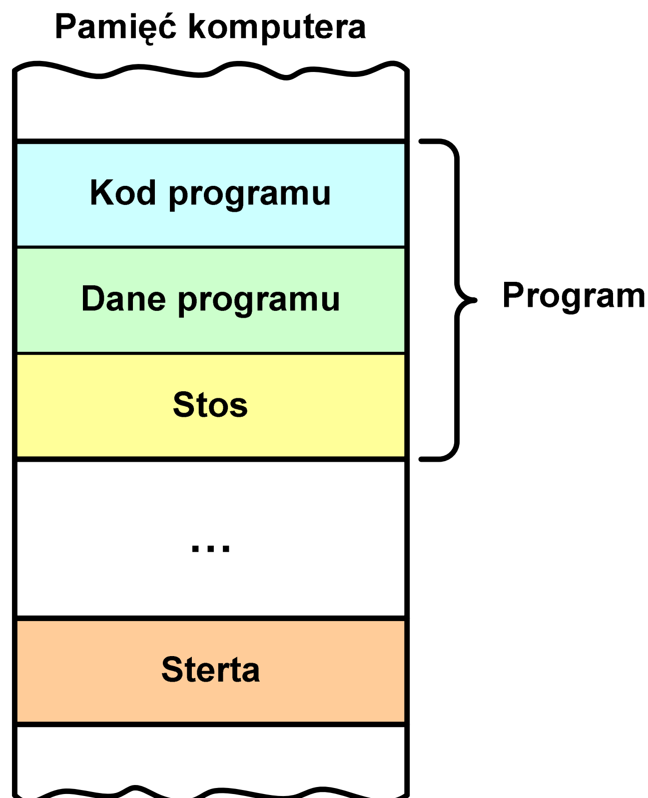
```
77  59  49  41  82  62
89  80  86  52   7  64
51  43   4  29  23  98
52  10   2  43  37  95
```

Pamięć a zmienne w programie

- Ze względu na czas życia wyróżnia się:
  - obiekty statyczne - istniejące od chwili rozpoczęcia działania programu aż do jego zakończenia
  - obiekty dynamiczne - tworzone i usuwane z pamięci w trakcie wykonania programu:
    - automatycznie, czyli bez udziału programisty
    - kontrolowane przez programistę
- O tym czy obiekt jest statyczny lub dynamiczny decyduje miejsce deklaracji oraz klasa pamięci (przypisywana jawnie w deklaracji lub przyjmowana domyślnie)
  - klasę pamięci określają słowa kluczowe: auto, static, register, umieszczane przed deklaracją zmiennej, np. static int x;
  - wszystkie zmienne globalne są statyczne
  - wszystkie zmienne lokalne zadeklarowane bez jawnego specyfikowania klasy pamięci są automatyczne (auto)
  - zmienną lokalną można uczynić statyczną dodając przed deklaracją static
  - zmienne klasy register są automatyczne

Pamięć a zmienne w programie

- Po uruchomieniu programu, jego struktura w pamięci operacyjnej jest następująca:



- Kod programu:
  - instrukcje programu
- Dane programu:
  - zmienne globalne
  - zmienne statyczne (static)
- Stos:
  - zmienne lokalne (automatyczne)
  - adresy powrotu z funkcji
- Sterta (wspólna dla wszystkich programów):
  - zmienne dynamiczne

Operatory new i delete (C++)

Operator new:

- w najprostszym przypadku operator new ma postać:

```
new typ
```

- operator ten alokuje obszar pamięci niezbędny dla przechowywania obiektu typu typ i zwraca wskaźnik na początek tego obszaru
- jeśli alokacja pamięci nie jest możliwa, to zwracana jest wartość NULL
- typ wskaźnika jest odpowiednio dopasowany do typu obiektu, więc nie są konieczne żadne konwersje wskaźnikowe
- pamięć przydzielona przez operator new istnieje do zakończenia programu lub do chwili zwolnienia tej pamięci za pomocą operatora delete

Operator delete:

- w najprostszym przypadku operator delete ma postać:

```
delete ptr
```

gdzie ptr jest wskaźnikiem wskazującym na obiekt stworzony przez new

Operatory new i delete (C++) - przykład

```
#include <stdio.h>

int main()
{
    int *ptr;

    ptr = new int;
    if (ptr == NULL)
    {
        printf("Blad przydzialu pamieci.\n");
        return 1;
    }

    *ptr = 10;

    printf("*ptr = %d\n",*ptr);

    delete ptr;

    return 0;
}
```

```
*ptr = 10
```

Przydział pamięci operatorem new i zwolnienie operatorem delete

Operatory new i delete (C++)

- operatorów new i delete nie stosuje się zazwyczaj do alokacji danych skalarnych, prostych typów
- znajdują one zastosowanie głównie przy tworzeniu dynamicznych tablic oraz przy tworzeniu dynamicznych struktur danych (stos, kolejka, lista)
- utworzenie tablicy n elementów typu typ ma postać:

```
new typ[n]
```

- wartością powyższego wyrażenia jest wskaźnik na typ (a nie na tablicę elementów typu typ)
- jako rozmiar tablicy n można podać dowolne wyrażenie, a nie wyrażenie stałe
- niestety nie można w ten sposób tworzyć dynamicznych tablic wielowymiarowych
- usuwając tablicę, trzeba podać, że chodzi o tablicę, a nie o pojedynczy element:

```
delete [] ptr;
```

gdzie ptr jest wskaźnikiem zwróconym przez operator new podczas tworzenia dynamicznej tablicy

Operatory new i delete (C++) - przykład

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int   *tab, i, n;
   float suma = 0.0;

   printf("Podaj ilosc liczb: "); scanf("%d"

   tab = new int[n];
   if (tab == NULL)
   {
      printf("Nie mozna przydzielic pamieci.\n"); exit(-1);
   }

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

   for (i=0; i<n; i++) suma = suma + tab[i];
   printf("Srednia %d liczb wynosi %f\n",n,suma/n);

   delete [] 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
```

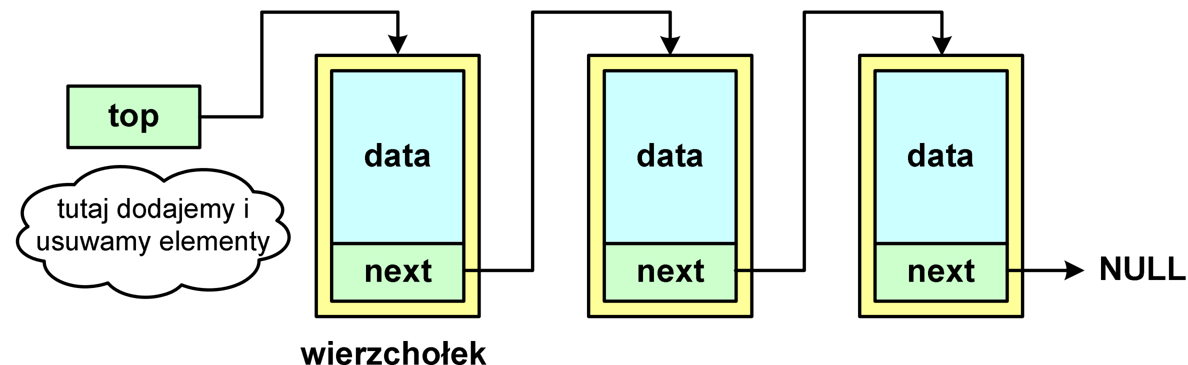
Przydział pamięci na n-liczb typu int, wczytanie liczb, obliczenie średniej

Dynamiczne struktury danych

- Dynamiczne struktury danych są to proste i złożone struktury danych, którym pamięć może być przydzielana i zwalniana w trakcie wykonywania programu
- Elementami takich struktur mogą być:
    - dane typów prostych, np. liczby, znaki
    - dane typów złożonych, np. struktury, tablice, obiekty
- Do podstawowych dynamicznych struktur danych należą:
    - stos
    - kolejka
    - listy:
        - jednokierunkowa
        - dwukierunkowa
        - cykliczna, jednokierunkowa
        - cykliczna, dwukierunkowa
    - drzewa

Stos

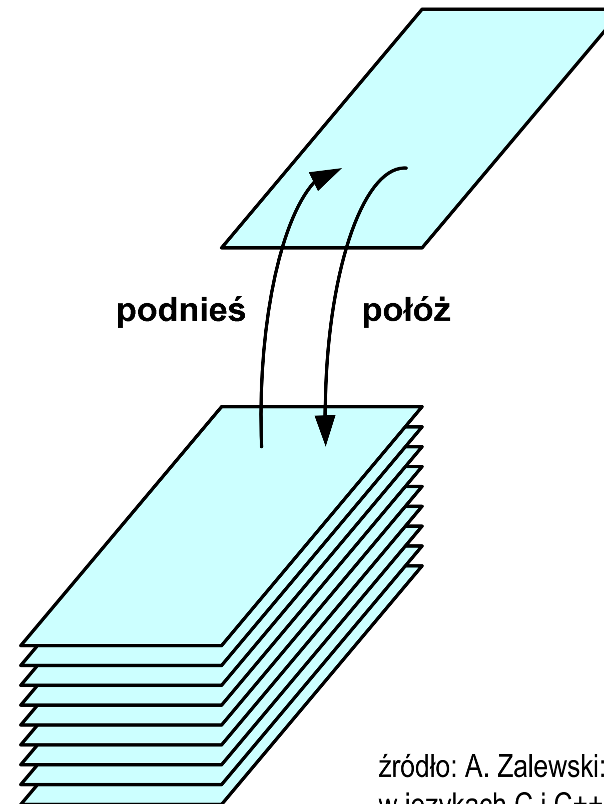
- stos (ang. stack) jest strukturą danych składającą się z elementów, z których każdy posiada tylko adres następnika
- dostęp do danych przechowywanych na stosie jest możliwy tylko w miejscu określanym mianem wierzchołka stosu (ang. top)
- wierzchołek stosu jest jedynym miejscem, do którego można dołączać lub z którego można usuwać elementy



- każdy składnik stosu posiada wyróżniony element (wskaźnik next) zawierający adres następnego elementu
- wskaźnik ostatniego elementu stosu wskazuje na adres pusty (NULL)
- dane (data) są umowną nazwą pewnych struktur

Stos

- nazwa stos odnosi się ściśle do funkcjonowania tej struktury - stos przypomina stertę kartek, na której wierzchołku można położyć kartkę lub ją zdjąć
- struktura stosu bywa nazywana stosem LIFO (ang. Last In First Out - ostatni wchodzi, pierwszy wychodzi)
- podstawowe operacje na stosie to:
  - dodanie elementu do stosu, czyli położenie elementu na stosie - funkcja push()
  - zdjęcie elementu ze stosu, czyli pobranie elementu ze stosu - funkcja pop()



źródło: A. Zalewski: „Programowanie w językach C i C++ z wykorzystaniem pakietu Borland C++”

Stos

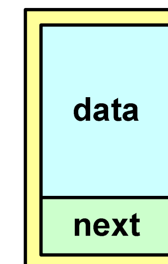
Implementacja w języku C:

- dane przechowywane na stosie są najczęściej pewną strukturą - dla uproszczenia przyjmijmy, że struktura skład się tylko z jednego pola typu int

```
struct element
{
   int x;
};
```

- każdy składnik stosu skład się z „użytecznych” danych przechowywanych na stosie (data) oraz ze wskaźnika (next) zawierającego adres następnego elementu

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



Stos

Implementacja w języku C - położenie elementu na stosie:

- funkcja dodająca element do stosu ma postać:

```
struct stos *push(struct stos *top, struct element data)
{
   struct stos *wsk;

   wsk = (struct stos*) malloc(sizeof(struct stos));
   wsk->next = top;
   wsk->data = data;

   return wsk;
}
```

- przykładowe wywołanie funkcji:

```
top = push(top,data);
```

- przed pierwszym wywołaniem funkcji push(), wskaźnik top musi mieć wartość NULL

Stos

Implementacja w języku C - zdjęcie elementu ze stosu:

- funkcja usuwająca element ze stosu ma postać:

```
struct stos *pop(struct stos *top, struct element *data)
{
   struct stos *wsk;

   if (top!=NULL)
   {
       wsk = top->next;
       *data = top->data;
       free(top);
       return wsk;
   }
   else return NULL;
}
```

- przykładowe wywołanie funkcji:

```
top = pop(top,&data);
```

Stos - przykład (1/3)

```
#include <stdio.h>
#include <stdlib.h>

struct element
{
   int x;
};

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

Implementacja stosu w języku C

Stos - przykład (2/3)

```
struct stos *push(struct stos *top, struct element data)
{
   struct stos *wsk;

   wsk = (struct stos*) malloc(sizeof(struct stos));
   wsk->next = top;
   wsk->data = data;

   return wsk;
}

struct stos *pop(struct stos *top, struct element *data)
{
   struct stos *wsk;

   if (top!=NULL)
   {
       wsk = top->next;
       *data = top->data;
       free(top);
       return wsk;
   }
   else
       return NULL;
}
```

Implementacja stosu w języku C

Stos - przykład (3/3)

```
int main()
{
    int tab[5] = {1,2,3,4,5};
    struct stos *top = NULL;
    struct element data;
    int i;

    for (i=0;i<5;i++)
    {
        data.x = tab[i];
        top = push(top,data);
        printf("push() --> %d\n",data.x);
    }

    printf("\n");
    while (top!=NULL)
    {
        top=pop(top,&data);
        printf("pop() --> %d\n",data.x);
    }

    system("pause");
    return 0;
}
```

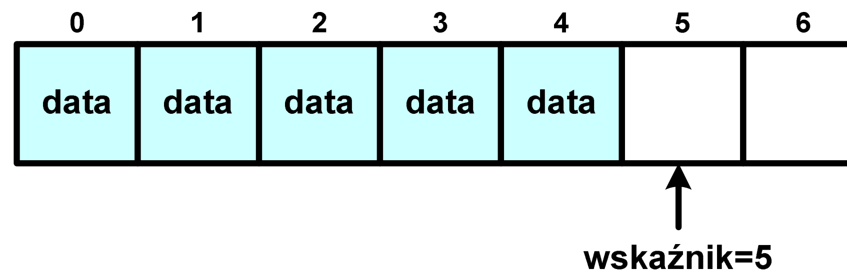
```
push() --> 1
push() --> 2
push() --> 3
push() --> 4
push() --> 5

pop() --> 5
pop() --> 4
pop() --> 3
pop() --> 2
pop() --> 1
```

Implementacja stosu w języku C

Stos

Tablicowa implementacja stosu:

- przedstawiona wcześniej struktura stosu jest to tzw. listowa implementacja stosu
- istniej także tablicowa implementacja stosu



- rozmiar tablicy jest stały i określany w momencie tworzenia stosu
- elementy stosu przechowywane są w tablicy, przy czym im później element został położony, tym dalej w tablicy się znajduje
- wskaźnik stosu określa indeks tablicy, pod który zapisana zostanie następna dana umieszczana na stosie
- przy umieszczaniu danej na stosie indeks jest zwiększany, zaś przy zdejmowaniu danej ze stosu - zmniejszany

Koniec wykładu nr 1

Dziękuję za uwagę!