

# TEORETYCZNE PODSTAWY INFORMATYKI

6/11/2017

WFAiS UJ, Informatyka Stosowana  
I rok studiów, I stopień

# Wykład 5 – część I

2

## Modele danych

- **Abstrakcja, modele danych a struktury danych**
- **Modele danych**
  - języków programowania
  - w oprogramowaniu systemowym
  - w edytorach tekstów
  - układów komputerowych
  - Języka C
- **Bazy danych i bazy wiedzy**

# Abstrakcja

3

## □ Abstrakcja

- Oznacza **uproszczenie**, zastąpienie skomplikowanych i szczegółowych okoliczności występujących w świecie rzeczywistym zrozumiałym **modelem** umożliwiającym rozwiązanie naszego **problemu**.
- Oznacza to, że „**abstrahujemy**” od **szczeǳółów**, które nie mają wpływu lub mają minimalny wpływ na rozwiązanie problemu.
- Opracowanie odpowiedniego modelu umożliwia zajęcie się istotą problemu.

# Modele danych

4

- **Modele danych** są to abstrakcje wykorzystywane do opisywania problemów.
- W informatyce wyróżniamy zazwyczaj dwa aspekty:
  - **Wartości** które nasz obiekt może przyjmować.
    - Przykładowo wiele modeli danych zawiera obiekty przechowujące wartości całkowitoliczbowe. Ten aspekt modelu jest statyczny; określa bowiem wyłącznie grupę wartości przyjmowanych przez obiekt.
  - **Operacje na danych**.
    - Przykładowo stosujemy zazwyczaj operacje dodawania liczb całkowitych. Ten aspekt modelu nazywamy dynamicznym; określa bowiem metody wykorzystywane do operowania wartościami oraz tworzenia nowych wartości.
- **Badanie modeli danych, ich właściwości oraz sposobów właściwego ich wykorzystania stanowi jedno z podstawowych zagadnień informatyki.**

# Modele danych a struktury danych

5

- **Modele danych to abstrakcje wykorzystywane do opisywania problemów.**
- **Struktury danych to reprezentacja danego modelu danych, którą musimy skonstruować w sytuacji gdy język programowania nie ma wbudowanej tej reprezentacji.**
- **Konstruujemy strukturę danych za pomocą abstrakcji obsługiwanych przez ten język.**

# Modele danych języków programowania

6

- Każdy język programowania zawiera **własny model danych**, który zazwyczaj istotnie różni się od modeli oferowanych przez inne języki.
- **Podstawowa zasada** realizowana przez większość języków programowania w odniesieniu do modeli danych określa, że **każdy program ma dostęp do „pudełek”**, które traktuje jako obszary pamięci.
  - Każde „pudełko” ma swój **typ**, np. int, char.
  - Wartości przechowywane w pudełkach nazywamy często **obiektami danych**.
  - Możemy teraz nadawać nazwy wykorzystywanym pudełkom. W ogólności nazwa jest dowolnym wyrażeniem wskazującym na pudełko.

# Modele danych języków programowania

7

- Podstawowe **typy danych** w języku programowania **C** to:
  - liczby całkowite,
  - liczby zmiennoprzecinkowe,
  - znaki,
  - tablice,
  - struktury,
  - wskaźniki.
- Wszystkie te pojęcia to **statyczne elementy modelu danych**.

# Modele danych języków programowania

8

- **Dopuszczalne operacje** na tych danych to:
  - typowe operacje arytmetyczne na liczbach całkowitych i zmiennoprzecinkowych,
  - operacje dostępu do elementów tablic i struktur,
  - oraz wyłuskiwanie wskaźników czyli znajdowanie obiektów przez nie wskazywanych.
- Te operacje to **dynamiczne elementy** modelu danych.



# Modele danych języków programowania

9

- **Bardzo ważne są też modele danych, które nie są częścią języka programowania, takie jak listy, drzewa, grafy, zbiory.**
- **Np. w języku matematycznym, lista jest ciągiem  $n$  elementów, który zapisujemy jako  $(a_1, a_2, \dots, a_n)$ . Do zbioru operacji wykonywanych na listach należą:**
  - **tworzenie listy,**
  - **wstawianie nowego elementu do listy,**
  - **usuwanie elementu z listy,**
  - **łączenie list.**

# Model danych w oprogramowaniu systemowym

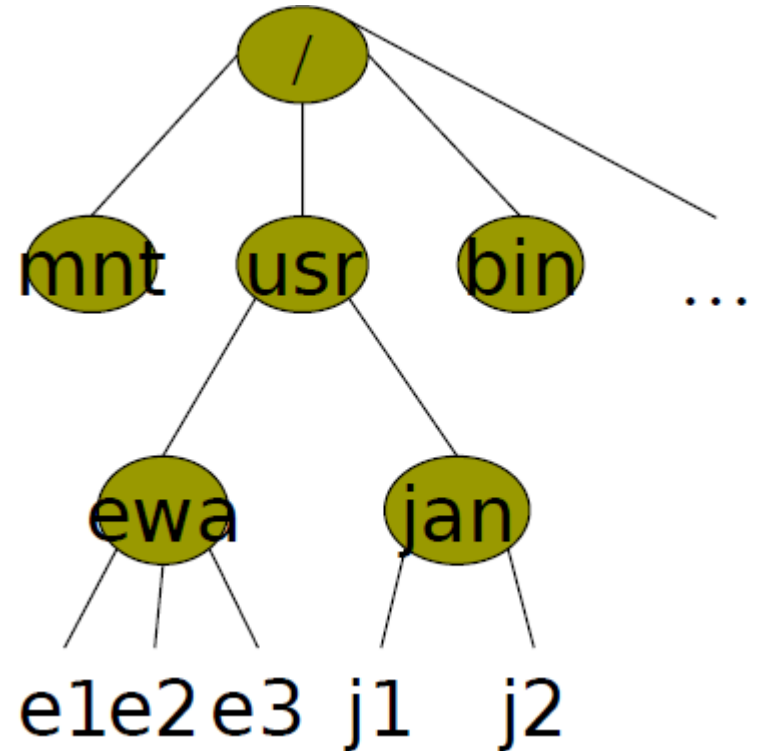
10

- **Modele danych możemy spotkać w systemach operacyjnych i w aplikacjach.**
- **Zadaniem systemu operacyjnego jest zarządzanie i szeregowanie zasobów komputera.**
- **Model danych systemów operacyjnych Unix składa się z takich pojęć jak: pliki, katalogi oraz procesy.**

# Model danych w oprogramowaniu systemowym

11

- Dane jako takie są przechowywane w plikach, które w systemie unix reprezentowane są przez ciągi znaków.
- **Pliki** są grupowane w ramach **katalogów** będących zbiorami plików i (lub) innych katalogów.
- **Katalogi i pliki tworzą drzewo** w którym pliki są liśćmi.



# Model danych w oprogramowaniu systemowym

12

- **Procesy** są pojedynczymi wykonaniami programów. Procesy pobierają zero lub więcej strumieni wejściowych i produkują zero lub więcej strumieni wyjściowych.
- W systemach Unix procesy mogą składać się z **potoków** (ang. pipes), kiedy to wynik jednego procesu może zasilać wejście kolejnego procesu. Efekt takiego połączenia procesów można traktować jako jeden duży proces z własnym wejściem i wyjściem.
  - **Przykład:** `ls | grep file`

# Model danych w oprogramowaniu systemowym

13

- Istnieje wiele innych aspektów działania **systemu operacyjnego**, np. sposób zarządzania bezpieczeństwem danych oraz interakcja z użytkownikiem.
- Dość łatwo można zauważyć że **model danych systemu operacyjnego** różni się od **modeli danych języków programowania**.

# Model danych w edytorach tekstu

14

- Każdy **model danych wbudowany w edytor** wiąże się z pojęciami ciągów tekstowych oraz operacjami charakterystycznymi dla redagowania tekstu.
- Model zawiera więc zazwyczaj **pojęcie wierszy** (ang. lines), które podobnie jak większość plików są ciągami znaków. Jednak w przeciwieństwie do plików wiersze mogą się wiązać ze swoimi numerami. Mogą być także grupowane w większe jednostki zwane **akapitami**.
- **Operacje** na wierszach można zazwyczaj stosować dla wszystkich zawartych w nich elementów, nie tylko dla ich początku, jak w przypadku najbardziej powszechnych operacji na plikach.
- Typowy edytor wykorzystuje również **pojęcie wiersza bieżącego** oraz **bieżącej pozycji w danym wierszu**. Wykonywane przez edytor **operacje zawierają rozmaite modyfikacje wierszy**, takie jak usuwanie i wstawianie znaków, usuwanie lub tworzenie nowych wierszy, poszukiwanie określonych ciągów znaków, itd.

# Model danych układów komputerowych

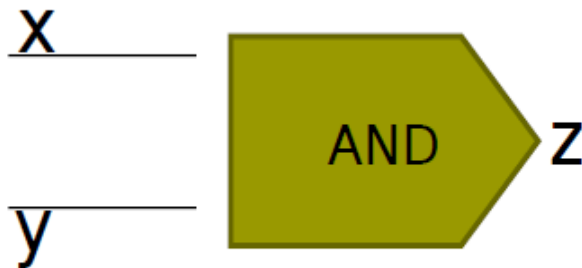
15

- Model danych opisujący **układy komputerowe** zwany jest logiką wnioskowania, jest najbardziej przydatnym narzędziem w projektowaniu komputerów.
- Komputery składają się z komponentów elementarnych zwanych **bramkami**. Każda bramka ma jedno lub więcej wejść i jedno wyjście; na wejściu i wyjściu dopuszczalne są tylko dwie wartości: 0 lub 1. **Bramka wykonuje prostą funkcję**, np. koniunkcję (AND).

# Model danych układów komputerowych

16

- Na pewnym poziomie abstrakcji projektowanie komputera jest procesem w którym decyduje się o sposobie połączenia bramek tak aby było możliwe wykonywanie **prostych operacji**.



x	y	z
0	0	0
0	1	0
1	0	0
1	1	1



# Sumator jednobitowy

17

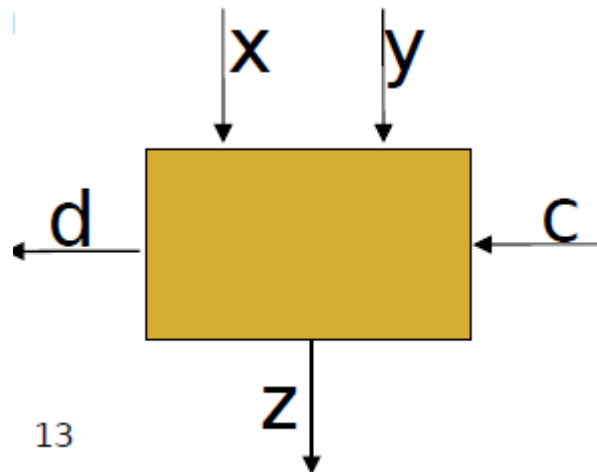
- Aby wykonać **instrukcję przypisania**  $a = b + c$  w języku C, komputer wykonuje dodawanie za pomocą układu zwanego sumatorem.
- W komputerze wszystkie liczby są zapisane w notacji binarnej wykorzystującej dwie cyfry: 0 i 1 (zwane cyframi binarnymi albo bitami). Mając kilka bramek możemy zbudować układ zwany **sumatorem jednobitowym**.
  - Dwa **bity wejściowe**  $x$ ,  $y$  oraz **wejściowy bit przeniesienia** są sumowane, wynikiem operacji **jest bit sumy oraz wyjściowy bit przeniesienia**  $d$ .

# Sumator jednobitowy

18

## □ Przykład:

- dz to łącznie **dwubitowa liczba binarna** wyrażająca łączną liczbę danych wejściowych ( $x, y, c$ ) mających wartość 1.
- $d$  = **bit przeniesienia sumy**
- $z$  = **bit sumy**



## Tabela prawdy:

x	y	c	d	z
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
1	0	0	0	1
0	1	1	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

# Modele danych języka C

19

- Statyczna część modelu danych w języku C to system typów opisujący wartości, które mogą być przyjmowane przez określone dane.
- System typów zawiera **typy proste**, np. liczby całkowite, oraz zbiór **zasad formowania typów**, dzięki którym możemy konstruować coraz bardziej skomplikowane typy na bazie typów już znanych.
- **Typy podstawowe**:
  - znaki (char, signed char, unsigned char),
  - liczby całkowite (int, short, long int, unsigned),
  - liczby zmiennoprzecinkowe (float, double, long double),
  - Wyliczenia (enum).
- Liczby całkowite i zmiennoprzecinkowe traktowane są jako **typy arytmetyczne**.

# Modele danych języka C

20

- **Reguły formowania typów** wymagają istnienia pewnych typów które mogą być albo typami podstawowymi; albo typami wcześniej skonstruowanymi za pomocą takich reguł.
- **Typy tablicowe:**
  - ▣ Możemy stworzyć tablice, której elementy są typu T:  $A[n]$ ;
  - ▣ Powyższa instrukcja deklaruje tablice n elementów, każdy typu T.
  - ▣ W języku C indeksy tablic rozpoczynają się od 0, zatem pierwszym elementem jest  $A[0]$ , ostatnim  $A[n-1]$ .
  - ▣ Tablice mogą być skonstruowane ze znaków, typów arytmetycznych, wskaźników, struktur, unii lub innych tablic.

# Modele danych języka C

21

## Struktury:

- Struktura jest **grupowaniem zmiennych** zwanych składnikami (ang. members) lub polami (ang. fields). Różne składniki struktur mogą być różnych typów, jednak każdy musi zawierać elementy jednego określonego typu.
- Jeśli  $T_1, T_2, \dots, T_n$  są typami oraz  $M_1, M_2, \dots, M_n$  są nazwami składników definiuje **strukturę** której wyróżnik (nazwa jej typu) to  $S$ , zaś  $n$  to liczba jej składników,  $i$ -ty składnik nosi nazwę  $M_i$  i jest typu  $T_i$ .

Deklaracja:

```
struct S {  
  
    T1 M1;  
  
    ....  
  
    Tn Mn;  
}
```

# Modele danych języka C

22

## Unie:

- Unia pozwala na przechowywanie zmiennych przyjmujących wartości różnych typów w różnych momentach wykonywania programu.
- definiuje zmienną  $x$ , która może przechowywać wartość dowolnego typu z grupy  $T_1, T_2, \dots, T_n$ .
- Nazwy składników  $M_1, M_2, \dots, M_n$  pomagają wyróżnić typ aktualnej wartości zmiennej. Oznacza to że  $x.M_i$  wskazuje na wartość zmiennej  $x$  traktowanej jako wartość typu  $T_i$ .

Deklaracja:

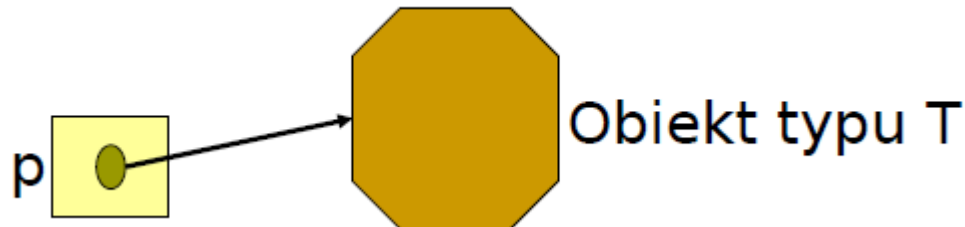
```
union {  
    T1 M1;  
    T2 M2;  
    ...  
    Tn Mn;  
}  
x;
```

# Modele danych języka C

23

## Wskaźniki:

- Język C wyróżnia się znaczeniem jaki mają w nim **wskaźniki**. Zmienna typu wskaźnikowego zawiera adres obszaru pamięci. Za pomocą wskaźnika możemy uzyskać dostęp do wartości innej zmiennej.
- Deklaracja: **T \*p;**  
definiuje zmienną p, jako wskaźnik do zmiennej typu T
- Zmienna p nazywa więc pudełko typu wskaźnikowego do T, wartością w pudełku p jest wskaźnik. Tym co „naprawdę” znajduje się w pudełku jest adres pod którym obiekt typu T jest przechowywany w komputerze.



# Modele danych języka C

24

## Typedef:

- Język C udostępnia instrukcje typedef, która umożliwia **tworzenie synonimów** dla nazw typów.
- **Deklaracja: typedef int Odległość;** pozwala na późniejsze używanie nazwy **Odległość** zamiast typu **int**.

## Funkcje:

- Funkcje także posiadają związane ze sobą typy, mimo że nie łączymy z nimi pudełek ani wartości.
- Dla dowolnej listy typów  $T_1, T_2, \dots, T_n$  możemy zdefiniować funkcje pobierającą odpowiednio  $n$  parametrów tych typów. Typ wartości zwracanych przez funkcje nazywamy typem funkcji. Jeżeli funkcja nie zwraca żadnej wartości wykorzystujemy typ **void**.



# Modele danych języka C

25

**W ogólności możemy budować typy dowolnie, stosując reguły ich konstrukcji, istnieje jednak kilka ograniczeń.**

- ▣ **Przykładowo nie możemy konstruować tablicy funkcji mimo że możemy zbudować tablice wskaźników do funkcji.**

# Operacje w modelu danych języka C

26

- **Przewidywane operacje na danych w modelu języka C możemy podzielić na trzy kategorie:**
  - **operacje tworzące i usuwające obiekt danych,**
  - **operacje dostępu i modyfikacji części obiektu danych,**
  - **operacje łączące części obiektu danych w celu sformowania nowej wartości obiektu danych.**

# Operacje w modelu danych języka C

27

## Tworzenie i usuwanie obiektu danych:

- **Język C udostępnia wiele elementarnych mechanizmów przeznaczonych do tworzenia danych.**
- **W momencie wywołania funkcji tworzone są pudełka dla wszystkich jej lokalnych argumentów (parametrów). Pozwala to na przechowywanie wartości tych parametrów.**
- **Innym mechanizmem jest procedura biblioteczna `malloc(n)`, która zwraca wskaźnik do  $n$  kolejnych pozycji znaków w niewykorzystanej pamięci. Obiekty danych mogą być wówczas utworzone właśnie w tych obszarach pamięci.**
- **Metody usuwania obiektów danych są analogiczne. Procedura biblioteczna `free` zwalnia pamięć zarezerwowaną przez `malloc`.**

# Operacje w modelu danych języka C

28

## Dostęp do danych i ich modyfikacja:

- **Język C zawiera mechanizm umożliwiający dostęp do komponentów składających się na obiekty. Wykorzystujemy:**
  - zapis `a[i]` do uzyskania dostępu do *i*-tego elementu tablicy *a*,
  - zapis `x.m` do uzyskania dostępu do składnika *m* struktury o nazwie *x*
  - zapis `*p` do uzyskania dostępu do obiektu wskazanego przez wskaźnik *p*.
- **Modyfikowanie (przypisywanie) wartości w języku C realizujemy za pomocą operatorów przypisania, które umożliwiają zmianę wartości obiektu. Np.:**
  - `a[0].(*pole[3]) = 99.`

# Operacje w modelu danych języka C

29

## Łączenie obiektów danych:

- Język C zawiera bogaty zbiór operatorów umożliwiających manipulowanie danymi i łączenie ich wartości. Oto podstawowe operatory:
  - Operatory arytmetyczne:  
dwuargumentowe  $+$ ,  $-$ ,  $\cdot$ ,  $/$ ,  $\%$ ,
    - $3 + 2 = 5$ ;
    - $3 - 2 = 1$ ;
    - $3 \cdot 2 = 6$ ;
    - $3 / 2 = 1.5$ ;
    - $3 \% 2 = 1$  (modulo – reszta z dzielenia);
  - jednoargumentowe  $+$ ,  $-$ ,  $++$ ,  $--$ ,
    - $n = -k$  (– jako zmiana znaku liczby);
    - $n++$ ,  $++n$  (inkrementacja, zwiększanie wartości o 1);
    - $n--$ ,  $--n$  (dekrementacja, zmniejszenie wartości o 1);

# Operacje w modelu danych języka C

30

## Operatory logiczne:

- Język C nie zawiera typu Boolean, wykorzystuje 0 do reprezentowania wartości logicznej fałszu oraz liczby różnej od zera do reprezentowania prawdy.
- Język C udostępnia:
  - ▣ koniunkcję && (dwuargumentowy)
  - ▣ alternatywę || (dwuargumentowy)
  - ▣ negację ! (jednoargumentowy)
  - ▣ operator warunkowy warunek ? y : z (trzyargumentowy), znaczący:

```
if (warunek)
    then return y;
else return x;
```

# Operacje w modelu danych języka C

31

## Operatory logiczne:

- **Operatory porównania:** (`==`, `!=`, `<`, `>`, `<=`, `>=` )
  - Dla liczb całkowitych i zmiennoprzecinkowych. Wynikiem jest prawda lub fałsz.
- **Operatory działań na poziomie bitowym**
- **Operatory przypisania**
- **Operatory koercji (konwersja, rzutowanie):**
  - przekształcenie wartości jednego typu na odpowiadającą jej wartość innego typu.
- **Często uzupełnia się podstawowe typy przez identyfikatory zdefiniowane w pliku nagłówkowym `stdio.h`: `NULL`, `TRUE`, `FALSE`, `BOOLEAN`, `EOF`.**

# Bazy danych

32

- W wielu zastosowaniach komputerów **same struktury danych nie wystarczają**. Nie zawsze jest to bowiem tylko kwestia rozważenia zadania algorytmicznego i zdefiniowania dobrych i użytecznych do jego rozwiązania struktur danych. Czasem potrzeba **bardzo obszernych zasobów danych, stanowiących dla wielu algorytmów potencjalne dane wejściowe**, a więc mające ustaloną strukturę i nadające się do odszukiwania i manipulowania nimi.
- Przykładami takich danych mogą być finansowe i osobowe dane przedsiębiorstwa, rezerwacje miejsc i informacje o lotach towarzystwa lotniczego, dane katalogowe biblioteki, itd....



# Bazy danych

33

- **Bazy danych** są zazwyczaj bardzo obszerne i zawierają wiele różnych rodzajów danych, począwszy od nazwisk i adresów po specjalne kody i symbole, a czasami nawet zwykły tekst.
- Zgromadzone dane są zazwyczaj przedmiotem **licznych rodzajów operacji** wstawiania, usuwania i wyszukiwania, wykorzystywanych w różnych celach przez różnych ludzi.
- O ile dodanie nowej informacji do bazy danych lub usunięcie już istniejącej są zadaniami stosunkowo łatwymi, o tyle **zapytanie bazy danych z zamiarem wydobywania z niej informacji** zazwyczaj jest dużo bardziej skomplikowane.

# Bazy danych

34

- **Ogromne znaczenie ma dobra organizacja bazy danych.**
- **Dobry projekt bazy danych** – to projekt przejrzysty, łatwy do zapisania, najważniejsza zaleta to duża sprawność działania i wykonalność opartego na tym projekcie systemu zarządzania bazą danych który potrafi odpowiedzieć na zapytania w krótkim czasie.
- **Stosuje się różne modele organizacji baz danych.** Modele są zaprojektowane do obsługi dużych ilości danych, jednocześnie wiernie i sprawnie wychwytyją związki zachodzące między obiektami danych.
- **Istnieje wiele metod i języków manipulacji danymi i zapytań baz danych.**

# Bazy danych

35

- Jeden z najpopularniejszych, **model relacyjny**, zaspokaja potrzeby związane z układami **danych w postaci ogromnych tabel**, przypominających tablicowe struktury danych.
- Inny model, **model hierarchiczny**, wymaga pewnych rodzajów **układów drzewiastych albo sieciowych**. Ten model organizuje dane w drzewiastej formie o wielu poziomach.
- Na niektóre rodzaje danych lepiej patrzeć jak na **fragmenty wiedzy** niż tylko jako na liczby, nazwiska czy kody.
- Oprócz dużej bazy danych opisującej wyposażenie przedsiębiorstwa produkcyjnego moglibyśmy chcieć mieć dużą bazę informacji dotyczących prowadzenia tego przedsiębiorstwa. Tego rodzaju fragmenty wiedzy wymagają organizacji bardziej złożonej niż obiekty danych o mniej więcej ustalonym formacie, zwłaszcza wówczas gdy zależy nam na sprawnym wyszukiwaniu.

# Bazy wiedzy

36

- **Bazy wiedzy** stają się następnym naturalnym stopniem po bazach danych, są bogatym źródłem ciekawych pytań związanych z reprezentowaniem, organizacją i wyszukiwaniem algorytmicznym.
- Problem **reprezentacji wiedzy** jest faktycznie jednym z podstawowych **zagadnień sztucznej inteligencji**.
- Trudność wynika z tego, że wiedza składa się nie tylko z wielkiego **zbioru faktów, ale także wielu zawiłych związków między nimi**. Te związki implikują inne, wyższego poziomu związki z innymi elementami wiedzy.

# Bazy wiedzy

37

- Zaproponowano wiele **modeli wiedzy** które można by wykorzystać w inteligentnych programach. Niektóre opierają się na pojęciach czysto informatycznych, takich jak relacyjne czy hierarchiczne bazy danych. Inne na logicznych formalizmach takich jak rachunek predykatów czy logika modalna.
- Pewne języki programowania, jak **Lisp** czy **Prolog**, łatwiej nadają się do manipulowania wiedzą niż inne.
  - ▣ Np. Prolog wydaje się trafnie dobrany jeżeli chodzi o fragmenty wiedzy dotyczące prostych relacji.
- Wymagane związki stają się coraz bardziej zagnatwane gdy wyjdziemy poza małą dobrze określoną dziedzinę dyskusji. Sięganie do wiedzy wiążącej się z pewną decyzją, którą program musi podjąć, staje się ogromnym zadaniem.

# Bazy wiedzy

38

**Efektywny” model algorytmicznej reprezentacji wiedzy wciąż czeka na odkrycie...**

# Wykład 5 – część II

39

## Listy

- **Podstawowa terminologia**
- **Lista jednokierunkowa**
- **Słownik**
- **Lista dwukierunkowa, cykliczna, lista z duplikatami**
- **Lista oparta na tablicy**
- **Stos**
- **Kolejka**

# Listy

40

- **Listy** należą do najbardziej podstawowych modeli danych wykorzystywanych w programach komputerowych.



# Podstawowa terminologia

41

## Lista

- Jest to skończona sekwencja zera lub większej ilości elementów.
- Jeśli wszystkie te elementy należą do typu  $T$ , to w odniesieniu do takiej struktury używamy sformułowania „lista elementów  $T$ ”.
- Możemy więc mieć listę liczb całkowitych, listę liczb rzeczywistych, listę struktur, listę list liczb całkowitych, itd. Oczekujemy że elementy listy należą do jednego typu, ale ponieważ może być on unią różnych typów to to ograniczenie może być łatwo pominięte.
- Często przedstawiamy listę jako  $(a_1, a_2, \dots, a_n)$  gdzie symbole  $a_i$  reprezentują kolejne elementy listy.
- Listą może być też ciąg znaków.

# Podstawowa terminologia

42

## Podlista

- Jeśli  $L = (a_1, a_2, \dots, a_n)$  jest listą, to dla dowolnych  $i$  oraz  $j$ , takich że  $1 \leq i < j \leq n$ , lista  $(a_i, a_{i+1}, \dots, a_j)$  jest podlistą (ang. sublist) listy  $L$ . Oznacza to, że podlista jest tworzona od pewnej pozycji  $i$ , oraz że zawiera wszystkie elementy aż do pozycji  $j$ . Lista pusta jest podlistą dowolnej listy.

## Części listy

- Jeżeli lista nie jest pusta to składa się z pierwszego elementu zwanego nagłówkiem (ang. head) oraz reszty listy, zwanej stopką (ang. tail). Istotne jest że nagłówek listy jest elementem, natomiast stopka listy jest listą .

# Podstawowa terminologia

43

## Przedrostek (ang. prefix)

- **Przedrostkiem listy jest dowolna podlista rozpoczynająca się na początku tej listy (czyli  $i=1$ ).**

## Przyrostek (ang. suffix)

- **Przyrostek jest dowolna podlista kończąca się wraz z końcem listy (czyli  $j=n$ ).**
- **Lista pusta jest zarówno przedrostkiem jak i przyrostkiem.**

## Długość listy

- **Jest to liczba wystąpień elementów należących do listy. Jeżeli liczba tych elementów wynosi 0 to mówimy że lista jest pusta.**

# Podstawowa terminologia

44

## Podciąg

- Jeśli  $L = (a_1, a_2, \dots, a_n)$  jest listą, to lista utworzona przez wyciągnięcie zera lub większej liczby elementów z listy  $L$  jest podciągiem listy  $L$ .
- Pozostałe elementy, które także tworzą podciąg, muszą występować w tej samej kolejności, w której występowały na oryginalnej liście  $L$ .

## Pozycja elementu na liście

- Każdy element na liście jest związany z określoną pozycją. Jeśli  $(a_1, a_2, \dots, a_n)$  jest listą oraz  $n \geq 1$ , to o elemencie  $a_1$  mówimy, że jest pierwszym elementem, o  $a_2$  że jest drugim elementem, itd. aż dochodzimy do elementu  $a_n$  o którym mówimy że jest ostatnim elementem listy.

# Podstawowa terminologia

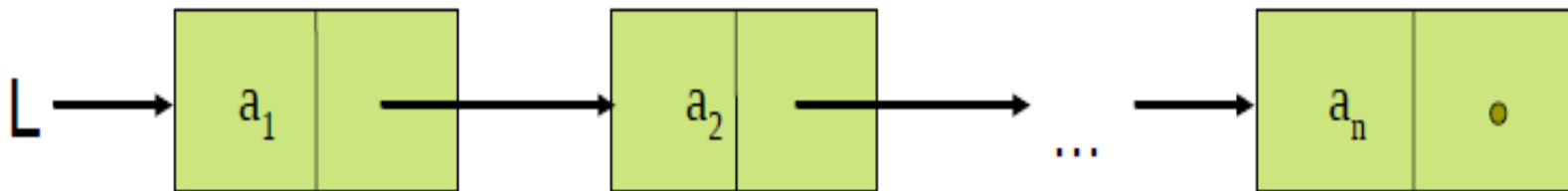
45

- **Operacje na listach, możemy:**
  - **sortować listę** czyli formalnie zastępować daną listę inną listą która powstaje przez wykonanie permutacji na liście oryginalnej,
  - **dzielić listę** na podlisty,
  - **scalać podlisty,**
  - **dodawać element** do listy,
  - **usuwać element** z listy,
  - **wyszukać element** w liście.

# Lista jednokierunkowa

46

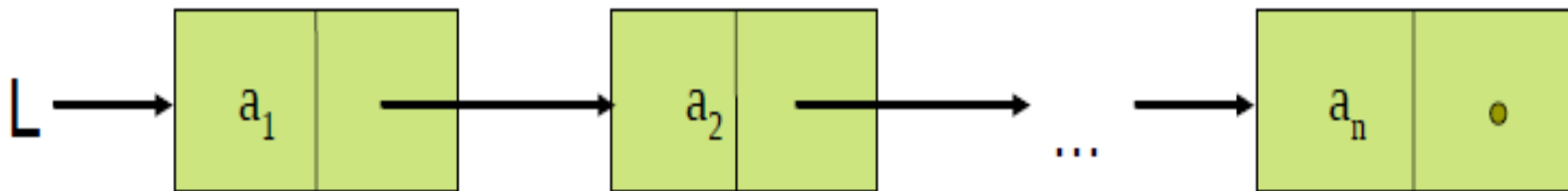
- Najprostszym sposobem **implementacji** listy jest wykorzystanie jednokierunkowej listy komórek.
  - Każda z komórek składa się z dwóch pól, jedno zawiera element listy, drugie zawiera wskaźnik do następnej komórki listy jednokierunkowej.
- Jeżeli mówimy o **konkretnej implementacji**, to oznacza że dyskutujemy „**strukturę danych**”
- Lista jednokierunkowa  $L = (a_1, a_2, a_3, \dots, a_n)$



# Lista jednokierunkowa

47

- Dla każdego elementu istnieje dokładnie jedna komórka; element  $a_i$  znajduje się w polu  $i$ -tej komórki
- Wskaźnik w  $i$ -tej komórce wskazuje na  $i+1$  komórkę dla  $i=1,2,\dots,n-1$ .
- Wskaźnik w ostatniej komórce jest równy NULL i oznacza koniec listy.
- Poza listą wykorzystujemy wskaźnik  $L$ , który wskazuje na pierwszą komórkę listy. Gdyby lista była pusta  $L = \text{NULL}$ .
- Dla każdej komórki znamy wskaźnik następnej.



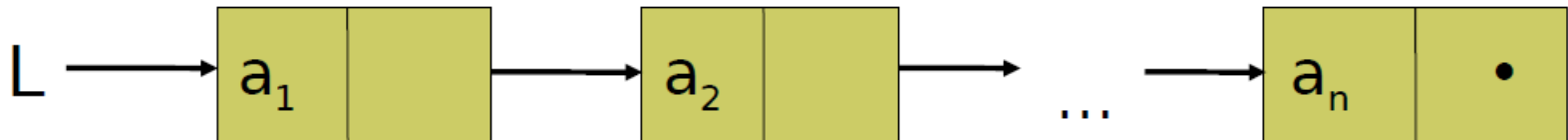
# Model danych a struktury danych

48

- **Lista to jest model danych (abstrakcja matematyczna)**
- **Lista jednokierunkowa to struktura danych**

```
typedef struct CELL *LIST;  
struct CELL{  
    int element;  
    LIST next;  
}
```

W niektórych językach (Lisp, Prolog) nie ma potrzeby stosowania (konstruowania) struktur danych do reprezentowania abstrakcyjnych list.





# Słownik

49

- Często wykorzystywanym **typem danych** jest **słownik**. Słownik zawiera zbiór elementów  $\{a_1, a_2, a_3, \dots, a_n\}$ . Uporządkowanie elementów nie ma znaczenia.
- Dla słownika chcemy wykonywać następujące operacje:
  - ▣ Wstawianie nowych elementów do słownika.
  - ▣ Usuwanie elementów ze słownika.
  - ▣ Wyszukiwanie elementu w celu sprawdzenia czy znajduje się w słowniku.

# Lista jednokierunkowa: struktura danych dla słownika

50

## Wyszukiwanie:

- Aby zrealizować tą operację musimy przeanalizować każdą komórkę listy reprezentującą słownik  $D$ , by przekonać się czy zawiera on szukany element  $x$ .
  - ▣ Jeśli tak odpowiedź jest „prawda”.
  - ▣ Jeśli dojdziemy do końca listy i nie znajdziemy elementu odpowiedź jest „fałsz”.
- Wyszukiwanie może być zaimplementowane rekurencyjnie.
- Średni czas wykonywania operacji wyszukiwania jest  $O(n/2)$ .

Dla listy  $L$  o długości  $n$  operacja wyszukiwania  $T(n) = O(n)$ .

**Podstawa:**  $T(0) = O(1)$ , ponieważ jeśli lista  $L = \text{NULL}$ , nie wykonujemy żadnego wywołania rekurencyjnego.

**Indukcja:**  $T(n) = T(n-1) + O(1)$ .

# Lista jednokierunkowa: struktura danych dla słownika

51

## Usuwanie:

- Aby zrealizować tę operację musimy przeanalizować każdą komórkę listy reprezentującą słownik  $D$ , by przekonać się czy zawiera on szukany element  $x$ . Jeśli tak, następuje usunięcie elementu  $x$  z listy.
- Ta operacja może być zaimplementowana rekurencyjnie, czas wykonania jest  $O(n)$ , średni czas wykonania jest  $O(n/2)$ .

Słownik jako abstrakcyjny typ danych nie dopuszcza duplikatów (z definicji) ale struktura danych która go implementuje (lista jednokierunkowa) **może te duplikaty dopuszczać.**

# Lista jednokierunkowa: struktura danych dla słownika

52

## Wstawianie:

- Aby wstawić  $x$  musimy sprawdzić, czy takiego elementu nie ma już na liście (jeśli jest nie wykonujemy żadnej operacji).
- Jeśli lista nie zawiera elementu  $x$  dodajemy go do listy. Miejsce w którym go dodajemy nie ma znaczenia, np. dodajemy go na końcu listy po dojściu do wskaźnika NULL.
- Podobnie jak w przypadku operacji wyszukiwania i usuwania, jeśli nie znajdziemy elementu  $x$  na liście dochodzimy do jej końca co wymaga czasu  $O(n)$ .
- Średni czas wykonywania operacji wstawiania jest  $O(n)$ .

# Lista jednokierunkowa: struktura danych dla słownika

53

- **Słownik jako abstrakcyjny typ danych nie wymaga uporządkowania ale struktura danych która go implementuje (lista jednokierunkowa) to uporządkowanie wprowadza. Możemy to wykorzystać.**
- **Wstawianie, wyszukiwanie, usuwanie: musimy znaleźć właściwe miejsce na wstawienie, dla wszystkich  $T(n) = O(n/2)$ .**

# Lista jednokierunkowa z duplikatami

54

## Wstawianie:

- tworzymy tylko nową komórkę:  $T(n) = O(1)$ .

## Wyszukiwanie:

- wygląda tak samo, możemy tylko musieć przeszukać dłuższą listę:  $T(n) = O(n)$ , średni czas jest  $O(n/2)$ .

## Usuwanie:

- Wygląda tak samo ale zawsze musimy przejrzeć całą listę:  $T(n) = O(n)$ , średni czas również  $O(n)$ .

# Lista jednokierunkowa: struktura danych dla słownika

55

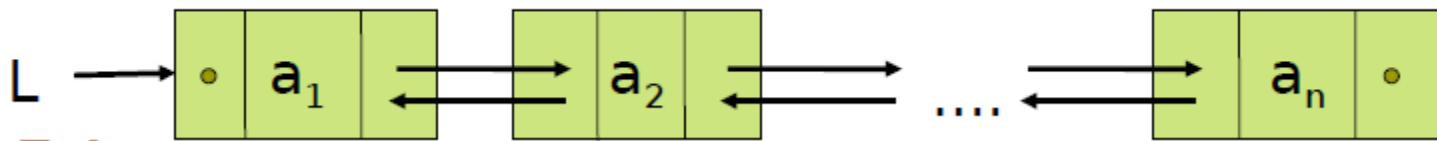
Lista	Wstawianie	Usuwanie	Wyszukanie
Brak Duplikatów	$n/2 \rightarrow n$	$n/2 \rightarrow n$	$n/2 \rightarrow n$
Duplikaty	0	m	$m/2 \rightarrow m$
Lista posortowana	$n/2$	$n/2$	$n/2$

- $n$  - ilość elementów w słowniku (oraz liście bez powtórzeń)
- $m$  – ilość elementów w liście z duplikatami
- $n/2 \rightarrow m$  oznacza że średnio przeszukujemy  $n/2$  razy przy pomyślnym wyniku oraz  $m$  przy niepomyślnym.

# Lista dwukierunkowa

56

- W przypadku listy jednokierunkowej nie ma mechanizmu na przejście od dowolnej komórki do początku listy.
- Lista dwukierunkowa umożliwia przemieszczanie się w obydwu kierunkach. Komórki listy dwukierunkowej składają się z dwóch pól.
- Dodatkowe pola zawiera wskaźnik do poprzedniej komórki na liście. Dla każdej komórki mamy wskaźnik do poprzedniej i następnej na liście.

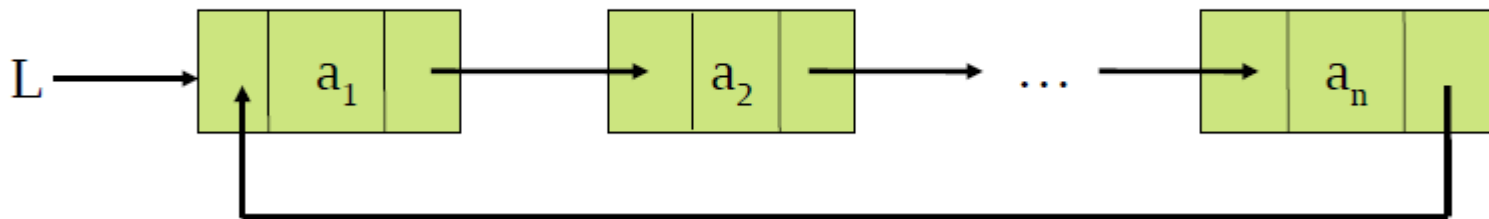




# Lista cykliczna

57

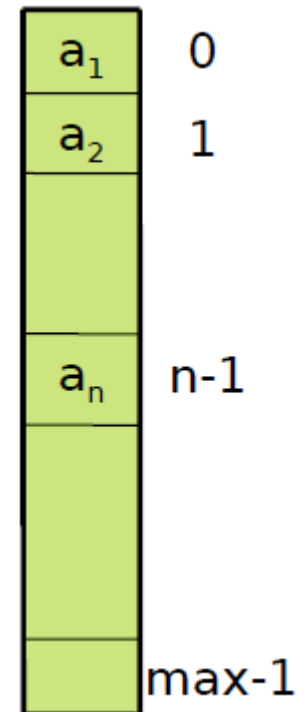
- Czasem nie potrzebujemy przechodzić listy wstecz, a jedynie mieć dostęp z każdego elementu do wszystkich innych.
- Prostym rozwiązaniem jest użycie listy cyklicznej. Na takiej liście, dla ostatniego elementu  $\text{next} \neq \text{NULL}$ , zamiast tego  $\text{next}$  wskazuje na początek listy.
- Nie ma więc pierwszego i ostatniego elementu, ponieważ elementy tworzą cykl. Potrzebny jest co najmniej jeden zewnętrzny wskaźnik do jakiegoś elementu listy.
- Jeśli lista zawiera tylko jeden element to musi on wskazywać sam na siebie. Jeżeli lista jest pusta to wartością każdego zewnętrznego wskaźnika do niej jest **NULL**.



# Lista oparta na tablicy

58

- Innym powszechnie znanym sposobem implementowania listy jest tworzenie struktury złożonej z dwóch komponentów:
  - **tablicy** przechowującej elementy listy  $L$ , musimy zadeklarować maksymalny wymiar tablicy.
  - **zmiennej** przechowującej liczbę elementów znajdujących się aktualnie na liście (oznaczona przez  $n$ ).
- **Implementacja listy oparta na tablicy** jest z wielu powodów bardziej wygodna niż oparta na liście jednokierunkowej.
  - **Wada:** konieczność deklarowania maksymalnej liczby elementów.
  - **Zaleta:** np. możliwość przeszukiwania binarnego jeżeli lista była posortowana.



# Lista oparta na tablicy

59

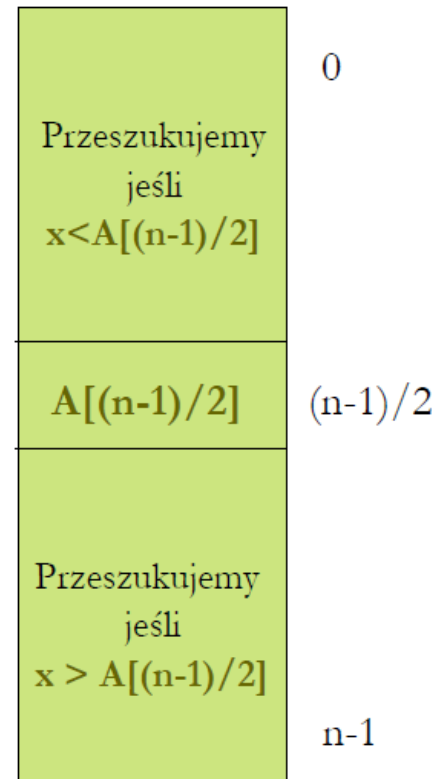
## Operacja przeszukiwania liniowego:

- ❑ Przeglądamy wszystkie elementy występujące w liście  $L$  (a więc w implementującej ją macierzy),
- ❑ Operacja jest  $T(n) = O(n)$ .

## Operacja przeszukiwania binarnego: (Możliwa jeśli lista była posortowana)

- ❑ Znajdujemy indeks środkowego elementu, czyli  $m = (n-1)/2$ .
- ❑ Porównujemy element  $x$  z elementem  $A[m]$ .
- ❑ Jeśli  $x = A[m]$  kończymy, jeśli  $x < A[m]$  przeszukujemy podlistę  $A[0, m-1]$ , jeśli  $x > A[m]$  przeszukujemy podlistę  $A[m+1, n-1]$ .
- ❑ Powtarzamy operację rekurencyjnie.
- ❑ Operacja jest  $T(n) = O(\log n)$ .

## Tablica A



# Stos

60

- **Stos:** Sekwencja elementów  $a_1, a_2, \dots, a_n$  należących do pewnego typu.
- **Operacje wykonywane na stosie:**
  - ▣ kładziemy element na szczycie stosu (ang. push)
  - ▣ zdejmujemy element ze szczytu stosu (ang. pop)
  - ▣ czyszczenie stosu – sprawienie że stanie się pusty (ang. clear)
  - ▣ sprawdzenie czy stos jest pusty (ang. empty)
  - ▣ sprawdzenie czy stos jest pełny

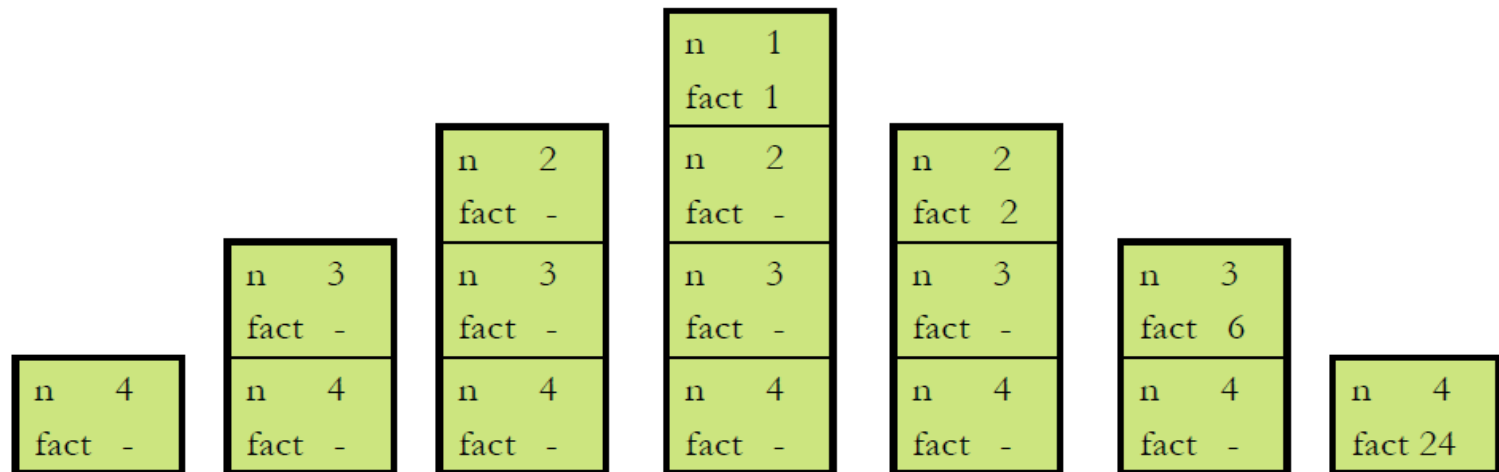
Każda z operacji jest  $T(n) = O(1)$ .

Stos jest wykorzystywany „w tle” do implementowania funkcji rekurencyjnych.

# Implementacja wywołań funkcji

61

- **Stos czasu wykonania przechowuje rekordy aktywacji dla wszystkich istniejących w danej chwili aktywacji. Wywołując funkcje kładziemy rekord aktywacji „na stosie”.**
- **Kiedy funkcja kończy swoje działanie, zdejmujemy jej rekord aktywacji ze szczytu stosu, odslaniając tym samym rekord aktywacji funkcji która ją wywołała.**



# Kolejka

62

- **Kolejka**: Sekwencja elementów  $a_1, a_2, \dots, a_n$  należących do pewnego typu.
- **Operacje wykonywane na kolejce**:
  - ▣ dołączenie elementu do końca kolejki (ang. push)
  - ▣ usunięcie elementu z początku kolejki (ang. pop)
  - ▣ czyszczenie kolejki – sprawienie że stanie się pusta (ang. clear)
  - ▣ sprawdzenie czy kolejka jest pusta (ang. empty)
- **Każda z operacji jest  $T(n) = O(1)$ .**

# Więcej abstrakcyjnych typów danych

63

Abstrakcyjny typ danych	Abstrakcyjna implementacja	Struktura danych
<b>Słownik</b>	Drzewa przeszukiwania binarnego	Struktura lewe dziecko - prawe dziecko
<b>Kolejka priorytetowa</b>	Zrównoważone drzewo częściowo uporządkowane	Kopiec
<b>Słownik</b>	Lista	1. Lista jednokierunkowa 2. Tablica mieszająca
<b>Stos</b>	Lista	1. Lista jednokierunkowa 2. Tablica
<b>Kolejka</b>	Lista	1. Lista jednokierunkowa 2. Tablica cykliczna

# Podsumowanie

64

- Ważnym modelem danych reprezentującym sekwencje elementów są listy.
- Do implementowania list możemy wykorzystać dwie struktury danych – **listy jednokierunkowe** i **tablice**.
- **Listy umożliwiają prostą implementacją słowników**, jednak efektywność takiego rozwiązania jest znacznie gorsza niż efektywność implementacji opartej na drzewach przeszukiwania binarnego. (Jest też gorsza od implementacji przy użyciu tablic mieszających, patrz następny wykład).
- Ważnymi **specyficznymi odmianami list są stosy i kolejki**.
- **Stos jest wykorzystywany w tle do implementowania funkcji rekurencyjnych**.



# Pytania do wykładu

65

- 1) **Co to są dynamiczne i statystyczne elementy modelu danych?**
- 2) **Wymień statyczne i dynamiczne elementy języka C**
- 3) **Co to jest lista jednokierunkowa?**
- 4) **Co to jest słownik, stos, kolejka?**
- 5) **Jakie operacje wykonujemy na modelach danych: lista, stos, kolejka, słownik. Jaką mają one złożoność obliczeniową?**