



# Zajebiste odpowiedzi typie makaki są sztos

**Sieć Aptek**

**i inne mały**

**June 25, 2022**

# Spis treści

<b>I</b>	<b>Pytania - dr. hab. Bogdan Księżopolski</b>	<b>5</b>
1	Sieci i programowanie sieciowe . . . . .	5
1.1	Protokoły TCP i UDP - porównanie i zastosowanie. . . . .	5
1.2	Protokół IP. . . . .	7
1.3	Modele sieci komputerowych. . . . .	10
1.4	Porównanie protokołów IPv4 i IPv6. . . . .	11
1.5	Format pakietu IP (poszczególne pola, zastosowanie). . . . .	12
1.6	Ethernet. . . . .	13
1.7	Protokoły warstwy aplikacji. . . . .	14
1.8	Charakterystyka modelu OSI i TCP/IP. . . . .	15
1.9	Rodzaje i przykłady nagłówków HTTP. . . . .	17
1.10	Protokół WebSocket. . . . .	18
1.11	Serwer zdarzeniowy, a wielowątkowy. Charakterystyka i porównanie. . . . .	19
2	Bezpieka . . . . .	20
2.1	Infrastruktura klucza publicznego - charakterystyka. . . . .	20
2.2	Kryptografia symetryczna oraz asymetryczna - charakterystyka. . . . .	21
2.3	Bezpieczeństwo sieci w odniesieniu do warstw modelu TCP/IP. . . . .	23
2.4	Kacper S Metody kontroli dostępu w systemach IT. . . . .	24
2.5	Kacper S Atrybuty bezpieczeństwa informacji. . . . .	25
<b>II</b>	<b>Pytania - dr. hab. Grzegorz Wójcik</b>	<b>26</b>
1	Bazy danych . . . . .	26
1.1	Model relacyjny baz danych i języki zapytań. . . . .	26
1.2	Paweł Model obiektowo-relacyjny baz danych, inne modele danych. . . . .	29
1.3	Składnia podstawowych zapytań języka SQL. . . . .	30
1.4	Projektowanie baz danych oraz model związków encji. . . . .	32
1.5	Paweł Problemy indeksowania baz danych, rodzaje indeksów, indeksy typu B+ drzewo. . . . .	34
1.6	Przetwarzanie transakcyjne OLTP (On-Line Transaction Processing). . . . .	35
2	Paradygmaty . . . . .	36
2.1	Kacper S Założenia paradygmatu programowania obiektowego. . . . .	36
2.2	Kacper S Idea dziedziczenia i polimorfizmu w programowaniu. . . . .	37
2.3	Zasady programowania dynamicznego. . . . .	38
2.4	Główne paradygmaty programowania. . . . .	39
2.5	Kacper S Cechy programowania deklaratywnego. . . . .	41

<b>III Pytania - reszta</b>	<b>42</b>
1    Język C i C++ . . . . .	42
1.1    Instrukcje sterujące w języku C. . . . .	42
1.2    Zarządzanie pamięcią w języku C. . . . .	43
1.3    Budowa, obsługa i formatowanie łańcuchów znakowych w języku C. . . . .	44
1.4    Zasięg i czas życia obiektów w języku C++. . . . .	45
1.5    Obsługa wyjątków w języku C++. . . . .	46
1.6    Definicje obiektu, klasy i szablonu klasy w języku C++. . . . .	47
2    Algorytmy . . . . .	48
2.1    Algorytmy sortujące. . . . .	48
2.2    Algorytmy zachłanne. . . . .	49
2.3    Metoda „dziel i zwyciężaj” konstruowania algorytmów. . . . .	50
2.4    Struktura kopców binarnych. . . . .	51
2.5    Algorytmy wyszukiwania najkrótszej ścieżki w grafie. . . . .	52
2.6    Sposoby implementacji słownika. . . . .	53
2.7    Tablice mieszające. . . . .	54
2.8    Algorytmy Monte Carlo oraz algorytmy Las Vegas. . . . .	55
2.9    Metody rozwiązywania rekurencji. Rekurencje Flawiusza i wieża w Hanoi. . . . .	56
2.10    Algorytmy Euklidesa. Algorytmy faktoryzacji. . . . .	57
2.11    Metody reprezentacji grafów w komputerze. . . . .	58
2.12    Droga i cykl Eulera. Droga i cykl Hamiltona. . . . .	59
2.13    Drzewo spinające graf. . . . .	60
3    Teoria obliczalności czy coś . . . . .	61
3.1    Pojęcia P, NP, NP-zupełne. . . . .	61
4    Automaty i inne takie . . . . .	62
4.1    Deterministyczne i niedeterministyczne automaty skończone. . . . .	62
4.2    Automaty z epsilon przejściami, wyrażenia regularne. . . . .	63
4.3    Kompilacja: gramatyka bezkontekstowa, skaner, parser, błędy. . . . .	64
5    Podstawy komputera i systemy operacyjne . . . . .	65
5.1    Systemy liczbowe i konwersje pomiędzy nimi. . . . .	65
5.2    Sposoby cyfrowej reprezentacji liczby całkowitej i rzeczywistej. . . . .	66
5.3    Wielowarstwowa organizacja oprogramowania komputera. . . . .	69
5.4    Procesy, zasoby i wątki. . . . .	70
5.5    Planowanie przydziału procesora, priorytety, wywłaszczanie oraz planowanie. . . . .	71
5.6    Zarządzanie pamięcią operacyjną. . . . .	72
5.7    Problem zakleszczenia, algorytm Bankiera. . . . .	73
6    Inżynieria Oprogramowania . . . . .	74
6.1    Standardowe metodyki procesu twórczego oprogramowania. . . . .	74
6.2    Metodyki zwinne (agile). . . . .	75
6.3    Metody testowania oprogramowania. . . . .	76
6.4    Walidacja i weryfikacja oprogramowania. . . . .	77
6.5    Diagramy UML (przypadków użycia, klas, aktywności, sekwencji, stanów, obiektów, wdrożenia). . . . .	78
6.6    Wzorce projektowe programowania obiektowego. . . . .	79
6.7    Wzorce architektoniczne. . . . .	80
7    Systemy wbudowane i elektronika . . . . .	81

7.1	Różnice pomiędzy obsługą zdarzeń w przerwaniach sprzętowych a obsługą zdarzeń w pętli programowej. . . . .	81
7.2	Stosowalność systemów opartych o mikrokontrolery vs stosowalność typowych komputerów (stacjonarnych i laptopów). . . . .	82
7.3	Dekoder, multiplekser i demultiplekser: budowa, zasada, działania, przeznaczenie/zastosowanie. . . . .	83
7.4	Podstawowe układy budujące system mikroprocesorowy i sposób wymiany informacji pomiędzy nimi. . . . .	84

# Rozdział I

## Pytania - dr. hab. Bogdan Księżopolski

### 1 Sieci i programowanie sieciowe

#### 1.1 Protokoły TCP i UDP - porównanie i zastosowanie.

##### TCP

Protokół TCP lub Transmission Control Protocol jest protokołem zorientowanym na połączenie, znajdującym się w warstwie transportowej modelu TCP / IP. Nawiązuje połączenie między komputerem źródłowym a docelowym przed rozpoczęciem komunikacji.

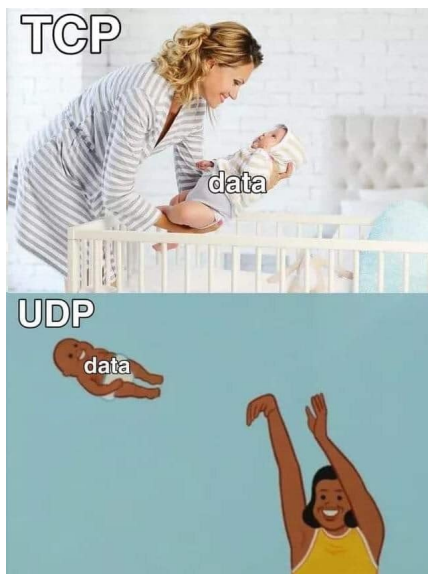


Rysunek I.1: TCP

Jest wysoce niezawodny, ponieważ wykorzystuje 3-drożną kontrolę uzgadniania, przepływu, błędów i przeciążenia. Zapewnia to, że dane wysyłane z komputera źródłowego są dokładnie odbierane przez komputer docelowy. Jeśli w przypadku, otrzymane dane nie są w odpowiednim formacie, to TCP ponownie przesyła dane. Poniższe protokoły używają TCP do transmisji danych:

- HTTP
- HTTPs
- FTP
- SMTP

## UDP



Rysunek I.2: UDP :D

Protokół UDP lub User Datagram Protocol to bezpołączeniowy protokół znajdujący się w warstwie transportowej modelu TCP / IP. Nie ustanawia połączenia ani nie sprawdza, czy komputer docelowy jest gotowy do odbioru, czy też nie, po prostu przesyła dane bezpośrednio. Protokół UDP służy do przesyłania danych z większą szybkością. Jest mniej niezawodny i dlatego jest używany do przesyłania danych, takich jak pliki audio i wideo. UDP nie gwarantuje ani dostarczenia danych, ani nie przesyła utraconych pakietów.

## 1.2 Protokół IP.

### IP (Internet Protocol)

Protokół internetowy jest protokołem komunikacyjnym warstwy Internet w modelu TCP/IP (odpowiada warstwie sieciowej modelu OSI). Protokół ten definiuje zasady i sposoby postępowania urządzeń sieciowych w celu nawiązania połączenia, utrzymania go i samej transmisji danych. Protokół IP stosowany jest w większości rodzajów sieci, w tym w sieci lokalnej i sieci Internet (każdy host, np. komputer, posiada swój własny, unikalny dla sieci adres IP).

Dane z użyciem protokołu IP transmitowane są w pakietach (paczkach danych). Nie gwarantuje on jednak dotarcia danych do celu czy utrzymania kolejności pakietów. Może się zdarzyć, że odbiorca otrzyma kilkukrotnie ten sam pakiet z całej paczki danych, pakiety dotrą w innej kolejności lub nie dotrą w ogóle. W celu zapewnienia prawidłowej transmisji stosuje się różne techniki w wyższej warstwie, np. z użyciem protokołu TCP.

Ponieważ każdy host w sieci posiada swój własny unikalny adres IP, obecnie wykorzystywana czwarta wersja protokołu (v4) okazała się niewystarczająca i brakuje wolnych adresów IP. W tym celu utworzona została wersja szósta (v6) znacznie zwiększająca ilość różnych adresów IP. Same adresy IP dzielone są na kilka grup z których 3 najważniejsze to adresy publiczne, adresy prywatne (do wykorzystania w sieciach domowych, np. 192.168.1.1), oraz adresy pętli zwrotnej (np. 127.0.0.1).

W skrócie:

- protokół komunikacyjny z warstwy trzeciej (sieci)
- jest to protokół bezpołączeniowy
- głównym zadaniem tego protokołu jest przypisywanie każdemu urządzeniu sieciowemu adresu IP i wybór trasy w celu przesłania pakietów z danymi (w przypadku problemów w przesyłaniu pakietów protokół wybierze trasy alternatywne do przesłania pakietów)
- nie zapewnia dostarczania pakietów (nie posiada mechanizmów retransmisji, lecz na szczęście za to odpowiadają protokoły z warstw wyższych)

### Klasy IP

Klasa	Zakres adresów pierwszego oktetu	Standardowa maska podsieci
A	0 – 127	255.0.0.0
B	128 – 191	255.255.0.0
C	192 – 223	255.255.255.0
D	224 – 239	–
E	240 – 255	–

Rysunek I.3: Klasy IP

**Adresy klasy A** przeznaczone są dla dużych sieci. Pierwszy bit oktetu w którym zawarty jest adres sieci jest równy 0. W związku z tym adresy sieci mogą przyjmować wartości od 0 do 127. Sieci 0 i 127 są zarezerwowane, więc do wykorzystania pozostają sieci od 1 do 126. W każdej sieci należącej do klasy A możemy wyodrębnić 16777216 adresów (liczba urządzeń będzie o 2 mniejsza, ale o tym w dalszej części artykułu). Klasa 127.0.0.0 wykorzystywana jest na potrzeby pętli zwrotnej, tj. umożliwia wysyłanie pakietów do samego siebie. Maska standardowa dla tej klasy to 255.0.0.0.

**Adresy klasy B** przeznaczone są do sieci średniej wielkości. Adres sieci zawarty jest w dwóch oktetach. Pierwsze dwa bity pierwszego oktetu wynoszą 10. W każdej sieci należącej do tego klasy można wyróżnić 65536 adresów (65534 urządzenia). Do tej klasy należą adresy sieci od 128 do 191 w ujęciu dziesiętnym. Maska standardowa dla tej klasy to 255.255.0.0.

**Adresy klasy C** przeznaczone są dla małych sieci, gdyż każda sieć może posiadać „jedynie” 256 adresów (254 urządzenia). Na adres sieci w sieciach należących do tej klasy przeznaczone są 3 oktety. Pierwsze trzy bity adresu wynoszą 110, w związku z tym do klasy tej należą adresy od 192 do 223 dziesiętne. Maska standardowa dla tej klasy to 255.255.255.0.

**Klasa D** została zarezerwowana na potrzeby rozsyłania grupowego przy użyciu adresów IP. Adres należący do tej klasy umożliwia przekierowanie pakietów do zdefiniowanej wcześniej grupy odbiorców. Dzięki temu możliwe jest przesłanie danych równocześnie do wielu odbiorców. Adresy tej klasy wykorzystywane są np. przez protokoły routingu. Pierwsze cztery bity adresu IP są równe 1110. Adresy należące do tej klasy zawierają się w przedziale od 224 do 239.

**Adresy należące do klasy E** zostały zarezerwowane przez Internet Engineering Task Force na potrzeby badawcze, wobec tego nie są dostępne publicznie. Pierwsze cztery bity adresu klasy E mają wartość 1111, w związku z tym adresy tej klasy zawierają się w przedziale od 240 do 255 dziesiętnie.

### **Prywatne adresy IP**

Adresy prywatne wg klas:

- Klasa A – 10.0.0.0 – 10.255.255.255 z maską 255.0.0.0
- Klasa B – 172.16.0.0 – 172.31.255.255 z maską 255.255.0.0
- Klasa C – 192.168.0.0 – 192.168.255.255 z maską 255.255.255.0

### **Rodzaje trasowania protokołu IP**

1. Anycast - dane są wysyłane do (topologicznie) najbliższego odbiorcy
2. Broadcast - dane wysyła do wszystkich możliwych hostów
3. Multicast - dane są wysyłane do wielu wybranych hostów (np. do hostów należących do jednej grupy)



4. Unicast - dane są wysyłane do jednego odbiorcy
5. Geocast - dane są wysyłane do wielu wybranych hostów należących do jednej strefy geograficznej

### **1.3 Modele sieci komputerowych.**

a

## **1.4 Porównanie protokołów IPv4 i IPv6.**

a

## 1.5 Format pakietu IP (poszczególne pola, zastosowanie).

0 - 3	4 - 7	8 - 15	16 - 18	19 - 23	24 - 31							
Wersja	IHL	Typ usługi	Długość całkowita									
Identyfikator			Flagi	Przesunięcie fragmentu								
Czas życia	Protokół		Suma kontrolna nagłówka									
Adres źródłowy												
Adres docelowy												
Opcje			Dopełnienie									
Dane												

Rysunek I.4: Datagram IPv4

**Wersja** - w tym polu nagłówka znajduje się wersja protokołu IP, w przypadku IPv4 znajduje się tam cyfra 4

**IHL** - długość nagłówka pakietu IP wyrażona w postaci liczby czterobajtowych części

**Typ usługi** – określa priorytet pakietu

**Długość całkowita** – pole zawiera całkowitą długość pakietu (nagłówek + dane), maksymalna długość pakietu to 65535 bajtów

**Identyfikator** – pole zawiera unikatowy identyfikator dla każdego pakietu wykorzystywany do połączenia pakietów w strumień danych

**Flagi** – określa między innymi czy pakiet może być fragmentowany

**Przesunięcie fragmentu** – umożliwia złożenie pakietu w całość pakietu, określają miejsce danego fragmentu w całym pakiecie

**Czas życia (TTL – Time To Live)** – ilość przeskoków przez które może pakiet przejść zanim zostanie odrzucony (urządzenia przez które przechodzi dany pakiet zmniejszają tę wartość o 1)

**Protokół** – to pole zawiera informacje jaki protokół warstwy transportowej został wykorzystany (TCP, UDP, ICMP lub inne)

**Suma kontrolna nagłówka** – gdy odbiorca dostanie pakiet, sprawdza jego poprawność obliczając sumę kontrolną i porównując ją z sumą kontrolną zapisaną w nagłówku

**Adres źródłowy i adres docelowy** – zawierają adresy IP urządzeń które przesyłają między siebie dane zapisane w formacie binarnym

## 1.6 Ethernet.

a

## **1.7    Protokoły warstwy aplikacji.**

a

## 1.8 Charakterystyka modelu OSI i TCP/IP.



Rysunek I.5: OSI - TCP/IP

### OSI

**Model ISO/OSI** (International Organization for Standardization / Open Systems Interconnection) - to standard opisujący komunikację siecią oraz jej etapy. Jest on znany jako "model odniesienia" który służy do analizy (i zrozumienia) komunikacji.

Warstwy:

1. **Warstwa fizyczna** - zaliczają się do niej wszystkie media transmisji danych (np. kable, fale radiowe) oraz sposób przesyłania przez nich informacji (np. jaka częstotliwość lub amplituda).
2. **Warstwa łącza danych** - przeprowadza ramkowanie danych (dodawanie nagłówka do danych) i przesyła je przez warstwę fizyczną. W nagłówku zawarty jest adres MAC nadawcy oraz odbiorcy (host zna adresy MAC swoich najbliższych sąsiadów). Przykładowy protokół: PPP, Ethernet, STP
3. **Warstwa sieci** - w tej warstwie tworzone są pakiety (dane + nagłówek IP). Wykonywane jest tutaj adresowanie logiczne, routing oraz szukanie najlepszej ścieżki. Przykładowe protokoły: ARP, ICMP, IPv4, IPv6, BGP, RIP, OSPF
4. **Warstwa transportowa** - tworzy ona segmenty danych (nagłówek zawierający numer portów + dane) i wykorzystuje do nich przesyłu przez protokół UDP lub TCP. Mogą tutaj występować mechanizmy zapewniające dostarczanie danych jak np. retransmisja (TCP ma, a UDP tego nie ma). Przykładowe protokoły: TCP, UDP, SSL/TLS

5. **Warstwa sesji** - nie modyfikuje danych lecz zarządza ona sesją i synchronizacją danych. Mając jakieś dane wie ona do jakiej aplikacji przesłać te dane (umożliwia komunikację między aplikacjami [end-to-end]).  
Przykładowe protokoły: NetBIOS, NFS, PAP
6. **Warstwa prezentacji** - polega na normalizacji danych według ustalonych standardów poprzez konwersję (zamianę), kompresję, szyfrowanie.  
Przykładowe protokoły: SSL, TLS, MIME
7. **Warstwa aplikacji** - tutaj działają aplikacje które widzi użytkownik służące do przyjmowania i wyświetlania danych użytkownika oraz przy wykorzystaniu gniazd do przyjmowaniu danych z sieci i wysyłaniu danych do sieci  
Przykładowe protokoły: HTTP, FTP, POP3, SNMP

## TCP/IP

**Model TCP/IP** - model określany inaczej jako “model protokołów”, gdzie każda warstwa wykonuje konkretne zadania.

Warstwy:

1. **Warstwa dostępu do sieci** - służy ona do przekazywania danych między urządzeniami sieciowymi (karty sieciowe, modemy) za pośrednictwem medium fizycznym (np. kable)  
Odpowiada ona warstwie fizycznej i łączy danych z modelu OSI.
2. **Warstwa internetu** - w tej warstwie występują routery opierające się o adresy IP i dokonujące trasowania (wyszukiwanie najlepszej trasy do odbiorcy). Występują tutaj protokoły takie jak: IP, ARP, ICMP (do diagnostyki), IGRP (do transmisji grupowej).  
Odpowiada ona warstwie sieci z modelu OSI.
3. **Warstwa transportowa** - służy ona do obsługi komunikacji oraz jej zabezpieczenia (czyli wykorzystanie np. retransmisji danych). Ta warstwa wykorzystuje porty dzięki czemu wie z jakiej aplikacji przychodzą dane, a podczas odbierania do jakiej aplikacji przesłać dane. Występuje tutaj protokół TCP oraz UDP.  
Odpowiada ona warstwie transportowej z modelu OSI.
4. **Warstwa aplikacji** - w tej warstwie występują procesy oraz aplikacje z których korzystają użytkownicy (np. serwer WWW, przeglądarka internetowa). Działają tutaj protokoły tj. HTTP, FTP, Telnet.  
Odpowiada ona warstwie sesji, prezentacji i aplikacji z modelu OSI.



## **1.9 Rodzaje i przykłady nagłówków HTTP.**

a

## **1.10    Protokół WebSocket.**

a

**1.11 Serwer zdarzeniowy, a wielowątkowy. Charakterystyka i porównanie.**

a

## 2 Bezpieka

### 2.1 Infrastruktura klucza publicznego - charakterystyka.

a

## 2.2 Kryptografia symetryczna oraz asymetryczna - charakterystyka.

### Kryptografia symetryczna

W kryptografii symetrycznej szyfrowanie i deszyfrowanie wykonywane jest przy użyciu tego samego klucza. W niektórych algorytmach wykorzystywane są dwa klucze, jednak muszą one być od siebie zależne w taki sposób, że znając jeden z nich, można wygenerować drugi.



Rysunek I.6: Zasada działania kryptografii symetrycznej

W celu zapewnienia bezpiecznej komunikacji, algorytm szyfrowania musi być tak skonstruowany, żeby odtworzenie tekstu jawnego bez znajomości klucza było zadaniem trudnym obliczeniowo. Dodatkowym wymaganiem jest tajność klucza – przed rozpoczęciem wymiany wiadomości, należy opracować protokół uzgadniania lub przekazywania klucza.

Algorytmy szyfrowania symetrycznego możemy podzielić na algorytmy blokowe i strumieniowe. Pierwsze z nich przekształcają blok danych ustalonej długości, traktując go jako całość, na szyfrogram o tej samej liczbie bitów. Szyfry strumieniowe przyjmują natomiast ciąg (strumień) danych. Algorytmy kryptografii symetrycznej są szybkie, zwykle wymagają też mniejszej mocy obliczeniowej niż algorytmy asymetryczne. Powszechnie stosowanym szyfrem symetrycznych jest **AES**.

### Kryptografia asymetryczna

Kryptografia asymetryczna to rodzaj kryptografii, w którym jeden z używanych kluczy jest udostępniony publicznie. Każdy użytkownik może użyć tego klucza do zaszyfrowania wiadomości, ale tylko posiadacz drugiego, tajnego klucza może odszyfrować taką wiadomość.

Kryptografia asymetryczna opiera się na funkcjach jednokierunkowych – takich, które da się łatwo wyliczyć w jedną stronę, ale bardzo trudno w drugą. Np. mnożenie jest łatwe, a rozkład na czynniki (z ang. faktoryzacja) trudny (na czym przykładowo opiera się **RSA**). Potęgowanie modulo jest łatwe, a logarytmowanie dyskretnie jest trudne (na czym opierają się ElGamal, DSA i **ECC**).

Klucz publiczny używany jest do zaszyfrowania informacji, klucz prywatny do jej odczytu. Ponieważ klucz prywatny jest w wyłącznym posiadaniu adresata informacji, tylko on może ją odczytać. Natomiast klucz publiczny jest udostępniony każdemu, kto zechce zaszyfrować wiadomość.

Ponieważ kryptografia asymetryczna jest o wiele wolniejsza od symetrycznej, prawie nigdy nie szyfruje się wiadomości za pomocą kryptosystemów asymetrycznych (również ze



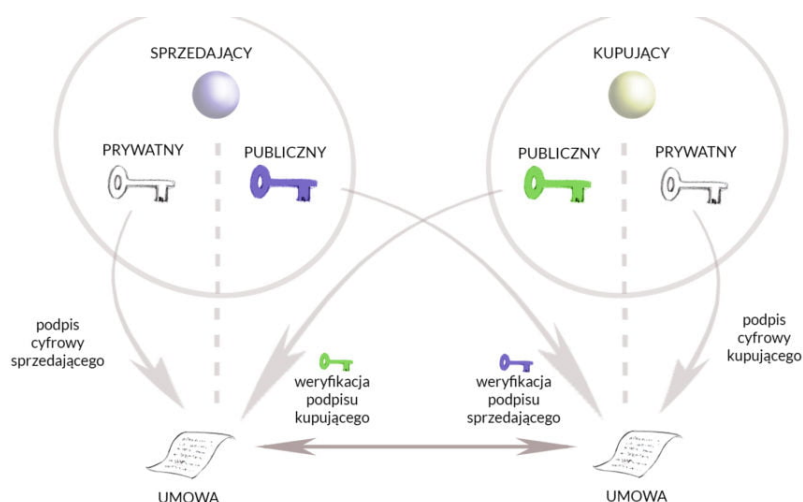
Rysunek I.7: Krok 1: Alice przesyła do Boba swój klucz publiczny



Rysunek I.8: Kroki 2 i 3: Bob szyfruje wiadomość kluczem publicznym Alice, która to następnie otrzymuje zaszyfrowaną wiadomość i rozszyfrowuje ją kluczem prywatnym

względem na ograniczenie wielkości szyfrowanej wiadomości). Zamiast tego szyfruje się jedynie klucz jakiegoś szyfru symetrycznego, takiego jak np. AES. Takie protokoły, łączące elementy kryptografii symetrycznej i asymetrycznej, nazywa się hybrydowymi.

Nadawcy mogą także używać kluczy prywatnych do cyfrowego podpisywania wiadomości. Te podpisy cyfrowe pozwalają odbiorcom uwierzytelnić tożsamość nadawcy i spać spokojnie, wiedząc, że wiadomości nie zostały zmienione od momentu podpisania. W takim przypadku przesyłane informacje mogą być publiczne, a odbiorca może użyć certyfikatu, który towarzyszy tej informacji, aby zweryfikować integralność i autentyczność podpisanej wiadomości.



Rysunek I.9: Jak działa podpis

## **2.3 Bezpieczeństwo sieci w odniesieniu do warstw modelu TCP/IP.**

a

## 2.4 Kacper S Metody kontroli dostępu w systemach IT.

a



## 2.5 Kacper S Atrybuty bezpieczeństwa informacji.

a

# Rozdział II

## Pytania - dr. hab. Grzegorz Wójcik

### 1 Bazy danych

#### 1.1 Model relacyjny baz danych i języki zapytań.

##### Model relacyjny baz danych

Relacyjny model danych pojawił się po raz pierwszy w artykule naukowym Edgara Codd'a w 1970 roku. W terminologii matematycznej - baza danych jest zbiorem relacji. Stąd historycznie pochodzi nazwa relacyjny model danych i relacyjna baza danych. W matematyce definiuje się relację jako podzbiór iloczynu kartezjańskiego zbiorów wartości. Reprezentacją relacji jest dwuwymiarowa tabela złożona z kolumn i wierszy.

Założenia modelu relacyjnego:

- Liczba **kolumn/attributów/pól (synonimy)** jest z góry ustalona.
- Z każdą kolumną jest związana jej nazwa (np. FirstName) oraz dziedzina (np. TEXT(20)), określająca zbiór wartości, jakie mogą wystąpić w kolumnie.
- Na przecięciu **wiersza/krotki/rekordu (synonimy)** i kolumny znajduje się pojedyncza (atomowa) wartość należąca do dziedziny kolumny.
- Wiersz reprezentuje jeden rekord informacji np. osobę.
- W modelu relacyjnym abstrahujemy od kolejności wierszy (rekordów) i kolumn (pól w rekordzie).

**Klucz główny:** dla każdej tabeli musi być określony klucz główny, będący jednoznacznym identyfikatorem. Może to być jedna lub więcej kolumn, w których wartości jednoznacznie identyfikują cały wiersz. Klucz główny w tabeli może być tylko jeden.

**Klucz jednoznaczny** ma te same właściwości co klucz główny, ale ich może być w tabeli więcej niż jeden.

**Klucz obcy** - jedna lub więcej kolumn, których wartości występują również jako klucz główny/jednoznaczny w tej samej/innej tabeli i są interpretowane jako wskaźniki do wierszy w tej drugiej tabeli

**Dwanaście postulatów Codd'a** – jest to zestaw 13 zasad stworzonych przez Edgara F. Codd'a – pioniera relacyjnych baz danych. Każda relacyjna baza danych musi je spełniać:

- System musi być kwalifikowany jako relacyjny, jako baza danych i jako system zarządzania.
- **Postulat informacyjny** – dane są reprezentowane jedynie przez wartości atrybutów w wierszach tabel (w krotkach).
- **Postulat dostępu** – każda wartość w bazie danych jest dostępna poprzez podanie nazwy tabeli, atrybutu i wartości klucza podstawowego (głównego).
- **Postulat dotyczący wartości NULL** – dostępna jest specjalna wartość NULL dla reprezentacji zarówno wartości nieokreślonej, jak i nieadekwatnej, inna od wszystkich i podlegająca przetwarzaniu.
- **Postulat dotyczący katalogu** – wymaga się, aby system obsługiwał wbudowany katalog relacyjny z bieżącym dostępem dla uprawnionych użytkowników używających języka zapytań.
- **Postulat języka danych** – system musi dostarczać pełny język przetwarzania danych, który może być używany zarówno w trybie interaktywnym, jak i w obrębie programów, obsługuje operacje definiowania danych, operacje manipulowania danymi, ograniczenia związane z bezpieczeństwem i integralnością oraz operacje zarządzania transakcji.
- **Postulat modyfikowalności perspektyw** – system musi umożliwiać modyfikowanie perspektyw, o ile jest ono semantycznie realizowalne.
- **Postulat modyfikowalności danych** – system musi umożliwiać operacje modyfikacji danych, musi obsługiwać operacje INSERT, UPDATE oraz DELETE.
- **Postulat fizycznej niezależności danych** – zmiany fizycznej reprezentacji danych i organizacji dostępu nie wpływają na aplikacje.
- **Postulat logicznej niezależności danych** – zmiany wartości w tabelach nie wpływają na aplikacje.
- **Postulat niezależności więzów spójności** – więzy spójności są definiowane w bazie i nie zależą od aplikacji.
- **Postulat niezależności dystrybucyjnej** – działanie aplikacji nie zależy od modyfikacji i dystrybucji bazy.
- **Postulat bezpieczeństwa względem operacji niskiego poziomu** – operacje niskiego poziomu nie mogą naruszać modelu relacyjnego i więzów spójności.

Operacje modelu relacyjnego:

- Selekcja - WHERE - selekcja podzbioru wierszy które spełniają określone warunki
- Projektcja - SELECT - pominięcie z wyniku pewnych kolumn
- Operacje na zbiorach: iloczyn kartezjański - FROM (złączenie tabel), suma zbiorów - UNION, różnica zbiorów - EXCEPT
- Agregacja - funkcje agregujące SUM, MIN itd. oraz klauzula GROUP\_BY

## Języki zapytań

**SQL** - Structured Query Language - jest to język strukturalny (i deklaratywny) służący do zarządzania bazą danych (CRUD). SQL jest najbardziej znanym językiem zapytań, ale istnieje także xBase.

- **DML (Data Manipulation Language)** służy do wykonywania operacji na danych – do ich umieszczania w bazie, kasowania, przeglądania oraz dokonywania zmian. Najważniejsze polecenia z tego zbioru to:

INSERT – umieszczenie danych w bazie,

UPDATE – zmiana danych,

DELETE – usunięcie danych z bazy.

- **DDL (Data Definition Language)** - operacje na strukturach, w których dane są przechowywane – czyli np. dodawanie, zmienianie i kasowanie tabel lub baz. Najważniejsze polecenia tej grupy to:

CREATE – utworzenie struktury (bazy, tabeli, indeksu itp.)

DROP – usunięcie struktury

ALTER – zmiana struktury

- **DCL (Data Control Language)** ma zastosowanie do nadawania uprawnień do obiektów bazodanowych. Najważniejsze polecenia w tej grupie to:

GRANT – nadawanie uprawnień do pojedynczych obiektów lub globalnie konkretnemu użytkownikowi

REVOKE – odbieranie wskazanych uprawnień konkretnemu użytkownikowi

DENY – zabranianie wykonywania operacji

- **DQL (Data Query Language)** to język formułowania zapytań do bazy danych. W zakres tego języka wchodzi jedno polecenie – **SELECT**. Często SELECT traktuje się jako część języka DML, ale to podejście nie wydaje się właściwe, ponieważ DML z definicji służy do manipulowania danymi – ich tworzenia, usuwania i uaktualniania. Na pograniczu obu języków znajduje się polecenie SELECT INTO, które dodatkowo modyfikuje (przepisuje, tworzy) dane.

## 1.2 **Paweł** Model obiektowo-relacyjny baz danych, inne modele danych.

a

## 1.3 Składnia podstawowych zapytań języka SQL.

### DDL - Data Definition Language

Jak zapamiętać skrót:

Definition = DEFINIOWANIE - TWORZENIE STRUKTUR

W skład DDL wchodzi DROP, CREATE oraz ALTER.

- DROP - usunięcie struktury

```
DROP TABLE tabela;
```

- CREATE - stworzenie struktury

```
CREATE TABLE tabela(  
  kolumna1 typ (rozmiar),  
  kolumna2 typ (rozmiar),  
  ...  
);
```

- ALTER - modyfikacja struktury. Obejmuje operacje takie jak np. dodanie kolumny do tabeli, zmiana typu danych w kolumnie, usunięcie kolumny.

```
ALTER TABLE table  
ADD kolumna typ(dlugosc);
```

### DML - Data Manipulation Language

Jak zapamiętać skrót:

Manipulation = MANIPULACJA - EDYCJA LUB TWORZENIE REKORDÓW

W skład DML wchodzi INSERT, UPDATE oraz DELETE.

- INSERT - dodawanie wierszy

```
INSERT INTO tabela (kolumna1, kolumna2, ..., kolumna_n)  
VALUES (wartosc1, wartosc2, ..., wartosc_n)
```

Nie trzeba podawać kolumn po nazwie tabeli gdy podamy po pierwsze wszystkie wartości, a po drugie w dobrej kolejności

- UPDATE - aktualizowanie danych, zmiana

```
UPDATE tabela  
SET kolumna1 = wartosc, kolumna2 = wartosc2, ...  
WHERE warunek
```

WHERE nie jest konieczny, możemy go użyć jak chcemy doprecyzować które rekordy mają się zaktualizować

- DELETE - usuwanie wierszy

```
DELETE FROM tabela  
WHERE warunek
```

WHERE nie jest konieczny, możemy go użyć jak chcemy doprecyzować które rekordy mają się skasować

## DCL - Data Control Language

Jak zapamiętać skrót:

Control = KONTROLA = UPRAWNIENIA

W skład DCL wchodzi GRANT, REVOKE oraz DENY.

- GRANT - nadawanie uprawnień do pojedynczych obiektów lub globalnie konkretnemu userowi
- REVOKE - odbieranie uprawnień konkretnemu userowi
- DENY - zabranianie wykonywania operacji
- Składnia jest taka sama dla w/w poleceń:

```
[GRANT/REVOKE/DENY] operacja1, operacja2, ...  
ON tabela  
TO user
```

Przykład:

```
GRANT SELECT, INSERT  
ON Fragment  
TO glazik
```

## DQL - Data Query Language

Jak zapamiętać skrót:

Query = ZAPYTANIA = SELECTY

W skład DQL wchodzi jedno polecenie: SELECT. Pozwala wybierać wiersze z bazy danych. Składnia:

```
SELECT kolumny  
FROM tabele  
WHERE warunek  
GROUP BY kolumna  
HAVING warunek  
ORDER BY ... DESC/ASC;
```

Dodatkowe klauzule SELECT:

- ORDER BY - sortowanie wyników względem np. kolumny
- ASC oraz DESC - dodawane po sortowaniu, wybieramy czy ma być ascending czy descending
- GROUP BY - grupowanie wyników względem danej kolumny
- HAVING - filtrowanie grup, działa podobnie do WHERE. WHERE oraz HAVING mogą występować jednocześnie w zapytaniu.

## 1.4 Projektowanie baz danych oraz model związków encji.

### Projektowanie baz danych

Etapy projektowania systemów bazodanowych:

- **Sformułowanie problemu**
- **Analiza wycinka rzeczywistości** - wywiad z ekspertem dziedzinowym, analiza wymagań funkcjonalnych (co dodawać, co usuwać itd.) i нефunkcjonalnych (tryb graficzny, platforma sprzętowa itd.)
- **Opracowanie conceptualnego modelu danych**, czyli wyodrębnienie i zdefiniowanie encji, związków między nimi oraz atrybutów (taka jakby luźna forma)  
Projektowanie modelu conceptualnego bazy danych składa się z następujących etapów:
  - **określenie zbiorów encji** - jakie encje (tabele) będzie zawierać baza?
  - **określenie atrybutów encji** - jakie cechy (kolumny) będą miały poszczególne encje?
  - **określenie dziedziny atrybutów** - jaki zakres wartości będzie miał każdy atrybut?
  - **ustalenie kluczy podstawowych** - a'la klucze główne (?)
  - **określenie związków między encjami** - a'la klucze obce (?)

**Encja** to każdy element rzeczywistości, który można scharakteryzować i odróżnić od innych obiektów, np. książka (przedmiot), klient (obiekt), zamówienie (zjawisko), stan cywilny (stan).

**Atrybut** to cecha encji. Zestaw atrybutów, które określamy dla encji, zależy od potrzeb bazy danych (np. książka ma atrybuty: tytuł, autor, ...)

**Dziedzina** opisuje jakie wartości może przyjąć dany atrybut, np. tytuł książki to napis o długości do x znaków.

- **Opracowanie modelu logicznego**, czyli wyrażenie świata rzeczywistego za pomocą reguł modelu danych (np. relacyjnego). Transformujemy model conceptualny do logicznego. Zamiast encji są tabele, zamiast związków są wiązania poprzez klucze główne i obce itd. Następuje normalizacja relacji.
- **Opracowanie modelu fizycznego**, czyli konstruowanie modelu świata rzeczywistego za pomocą struktur danych i mechanizmów istniejących w wybranym SZBD (systemie zarządzania bazami danych), czyli np naklepanie tego w MySQL
- **Tworzenie aplikacji**
- **Testowanie systemu**



### Model związków encji

Model związków encji (model ER lub ERD) to jedna z najpopularniejszych metod modelowania danych. Służy do graficznego przedstawiania koncepcji projektowanej bazy danych. Jest bardzo podobny do relacyjnego modelu. Diagram taki pokazuje obiekty (encje), ich cechy (atrybuty) oraz relacje między nimi. Elementy takie, jak encja, atrybut oraz związek zostały opisane stroną wyżej.

Typy związków encji:

- **jeden do jednego (1:1)**

Związek ten określa, iż każde wystąpienie danego obiektu (encji) związane jest wyłącznie z pojedynczym wystąpieniem innego bytu. Na przykład: Pracownik w organizacji ma przypisane konkretne Miejsce, a to Miejsce jest przypisane do konkretnego Pracownika. Kierownik zarządza jednym Działem, a Dział zarządzany jest przez jednego Kierownika.

- **jeden do wielu (1:N)**

Związek ten określa, iż jedno wystąpienie danego obiektu (encji) związane jest z większą ilością wystąpień innego bytu. Na przykład: Klient składa wiele Zamówień. Konkretnie Zamówienie jest składane przez danego Klienta. W Firmie istnieje wiele Działów. Konkretny Dział istnieje w danej Firmie.

- **wiele do wielu (M:N)**

Związek ten określa, iż jedno wystąpienie danego obiektu (encji) związane jest z większą ilością wystąpień innego bytu, tak samo jak jedno wystąpienie innego bytu może być powiązane z większą ilością wystąpień tego pierwszego obiektu. Przykładem takiego związku może być zależność jaka występuje pomiędzy Studentem oraz Wykładem. Student zapisany może być na wiele Wykładów, tak i na dany Wykład zapisanych może być wielu Studentów.

**1.5** **Paweł** Problemy indeksowania baz danych, rodzaje indeksów, indeksy typu B+ drzewo.

a

## 1.6 Przetwarzanie transakcyjne OLTP (On-Line Transaction Processing).

**OLTP** (Online Transaction Processing) to kategoria aplikacji klient-serwer dotyczących baz danych w ramach bieżącego przetwarzania transakcji obejmujących takie zastosowania jak systemy rezerwacji, obsługa punktów sprzedaży, systemy śledzące itp. W systemach tych klient współpracuje z serwerem transakcji, zamiast z serwerem bazy danych. Zastosowanie - np. w bankowości internetowej, w handlu detalicznym, podczas składania zamówień lub przy wysyłaniu wiadomości tekstowych. Transakcje te (tradycyjnie określane jako transakcje gospodarcze lub finansowe) są rejestrowane i zabezpieczane w taki sposób, aby przedsiębiorstwo mogło w każdej chwili uzyskać do nich dostęp w celach księgowych lub sprawozdawczych.

System OLTP:

- Umożliwia wykonywanie w czasie rzeczywistym dużej liczby transakcji bazodanowych przez dużą liczbę osób
- Wymaga błyskawicznych czasów reakcji
- Często modyfikuje małe ilości danych i zazwyczaj dokonuje tyle samo odczytów, co zapisów danych
- Używa indeksowanych danych w celu skrócenia czasu odpowiedzi
- Wymaga częstego lub równoczesnego wykonywania kopii zapasowych bazy danych
- Wymaga stosunkowo niewiele miejsca do przechowywania danych
- Zazwyczaj obsługuje proste zapytania dotyczące tylko jednego lub kilku rekordów

**Transakcja** to zbiór operacji na bazie danych, które stanowią w istocie pewną całość i powinny być wykonane wszystkie lub żadna z nich. Składa się zawsze z trzech etapów: **rozpoczęcia, wykonania i zamknięcia**. W systemach bazodanowych istotne jest, aby trwała jak najkrócej, ponieważ równolegle może być wykonywane wiele transakcji i część operacji musi zostać wykonana w pewnej kolejności. Każdy etap transakcji jest logowany, dzięki czemu w razie awarii można odtworzyć stan bazy sprzed transakcji.

Cechy transakcji: **ACID**

- **Atomicity** - atomowość - wykona się wszystko albo nic (zbiór operacji = całość);
- **Consistency** - spójność - baza przed i po zmianach jest spójna;
- **Isolation** - izolacja - równolegle wykonywane transakcje nie mają na siebie wpływu;
- **Durability** - trwałość - zmiany prowadzone przez transakcje są trwałe (nawet w przypadku awarii);

W SQL wyróżniamy następujące polecenia dotyczące transakcji:

- **BEGIN** lub **BEGIN\_WORK** – rozpoczęcie transakcji
- **COMMIT** – zatwierdzenie zmian wykonanych w obrębie transakcji
- **ROLLBACK** – odrzucenie zmian wykonanych w obrębie transakcji
- **SAVEPOINT\_nazwa** – zdefiniowanie punktu pośredniego o określonej nazwie
- **RELEASE\_SAVEPOINT\_nazwa** – skasowanie punktu pośredniego (nie wpływa w żaden sposób na stan transakcji)
- **ROLLBACK\_TO\_SAVEPOINT\_nazwa** – wycofanie transakcji do stanu zapamiętanego w podanym punkcie pośrednim.

## 2 Paradygmaty

### 2.1 Kacper S Założenia paradygmatu programowania obiektowego.

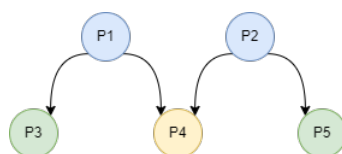
a

## 2.2 Kacper S Idea dziedziczenia i polimorfizmu w programowaniu.

a

## 2.3 Zasady programowania dynamicznego.

**Programowanie dynamiczne** - jest to rozszerzenie strategii "dziel i zwyciężaj". „Programowanie” oznacza w tym kontekście tabelaryczną metodę rozwiązywania problemów, a nie pisanie programów komputerowych. Wiemy, że w algorytmach typu „dziel i zwyciężaj” dzieli się problem na niezależne podproblemy, rozwiązuje je rekurencyjnie, a następnie łączy się rozwiązania wszystkich podproblemów w celu utworzenia rozwiązania pierwotnego problemu. Programowanie dynamiczne można stosować wtedy, kiedy podproblemy nie są niezależne, tzn. kiedy podproblemy mogą zawierać te same podpodproblemy. Wtedy algorytm typu „dziel i zwyciężaj” wykonuje więcej pracy niż to w istocie jest konieczne, wielokrotnie bowiem rozwiązuje ten sam podproblem.



Rysunek II.1: Przykładowy problem złożony

Jak widać na obrazku wyżej chcąc obliczyć rozwiązania problemów P1 i P2 rozwiązaniem typu "dziel i zwyciężaj", problem P4 zostałby policzony 2 razy jako problem zależny. Programowanie dynamiczne ma na celu optymalizację takich przypadków i pisząc program zapamiętalibyśmy rozwiązanie P4 gdy liczymy P1. Następnie obliczając drugi główny problem - P2, znamy już rozwiązanie P4 więc nie marnujemy czasu i mocy obliczeniowej.

**Proces projektowania algorytmu** opartego na programowaniu dynamicznym można podzielić na cztery etapy:

1. Scharakteryzowanie struktury optymalnego rozwiązania.
2. Rekurencyjne zdefiniowanie kosztu optymalnego rozwiązania.
3. Obliczenie optymalnego kosztu metodą wstępującą (ang. bottom-up) – czyli rozpoczynając od najmniejszych podproblemów, rozwiązywać coraz większe, wykorzystując zapamiętane rozwiązania mniejszych.
4. Konstruowanie optymalnego rozwiązania na podstawie wyników wcześniejszych obliczeń.

**Zasady** paradygmatu programowania dynamicznego (pretty much to samo co proces projektowania) to:

- Scharakteryzować strukturę rozwiązania optymalnego.
- Zdefiniować rekurencyjnie wartość rozwiązania optymalnego, jako funkcję rozwiązań optymalnych dla podproblemów.
- Skonstruować optymalne rozwiązanie problemu na bazie wyliczonych wcześniej wielkości.

## 2.4 Główne paradygmaty programowania.

**Paradygmat programowania** – jest to wzorzec sposobu programowania, patrzenia na dane, przepływ danych oraz sposób wykonywania programu.

### Programowanie imperatywne

Chcąc najkrócej scharakteryzować programowanie imperatywne, moglibyśmy napisać „zrób najpierw to, a potem tamto”. Mamy zatem sekwencję poleceń zmieniających krok po kroku stan maszyny, aż do uzyskania oczekiwanego wyniku — czyli mamy stan będący funkcją czasu. Ten sposób patrzenia na programy związany jest ściśle z budową sprzętu komputerowego o architekturze von Neumanna, w którym poszczególne instrukcje to właśnie polecenia zmieniające ów globalny stan.

Imperatywne języki programowania posługują się abstrakcjami bliskimi architekturze von Neumanna, np. zmienne są abstrakcją komórek pamięci. Naturalną abstrakcją są tu też procedury (będące, notabene, zasadniczym elementem „podparadygmatu” proceduralnego). Najważniejsze — w swoim czasie — języki imperatywne to Fortran, Cobol, Pascal i C.

### Programowanie proceduralne

Programowanie proceduralne – paradygmat programowania zalecający dzielenie kodu na procedury, czyli fragmenty wykonujące ściśle określone operacje. Procedury nie powinny korzystać ze zmiennych globalnych (w miarę możliwości), lecz pobierać i przekazywać wszystkie dane (czy też wskaźniki do nich) jako parametry wywołania.

### Programowanie strukturalne

O programowaniu strukturalnym wspominamy z kronikarskiego obowiązku, jest to bowiem bardzo dobrze znany i powszechnie stosowany „podparadygmat” programowania imperatywnego, czy ściślej — proceduralnego. Chodzi w nim o tworzenie programów z kilku dobrze zdefiniowanych konstrukcji takich jak instrukcja warunkowa if-then-else i pętla while, za to bez skoków (go to). Powinno to sprzyjać pisaniu programów przejrzystych, łatwych w rozumieniu i utrzymaniu.

Ściślej, Dijkstra proponował użycie tylko trzech rodzajów struktur sterujących:

- Sekwencja (lub konkatenacja) — czyli po prostu wykonanie instrukcji w określonej kolejności. W wielu językach rolę „operatora konkatenacji instrukcji” spełnia niepozorny średnik...
- Wybór — czyli wykonanie jednej z kilku instrukcji zależnie od stanu programu. Przykładem jest if-then-else i switch/case.
- Iteracja, czyli powtarzanie instrukcji tak długo, jak długo spełniony (lub niespełniony) jest dany warunek. Chodzi oczywiście o pętle, np. while, repeat-until, for itp.

### Programowanie obiektowe

W programowaniu obiektowym program to zbiór porozumiewających się ze sobą obiektów, czyli jednostek zawierających określone dane i umiejących wykonywać na nich

określone operacje. Najważniejsze są tu dwie cechy: po pierwsze, powiązanie danych (czyli stanu) z operacjami na nich (czyli poleceniami) w całość, stanowiącą odrębną jednostkę — obiekt; po drugie, mechanizm dziedziczenia, czyli możliwość definiowania nowych, bardziej złożonych obiektów, na bazie obiektów już istniejących.

Z tych dwóch cech bierze się zapewne wielki sukces paradygmatu obiektowego. Umożliwia on bowiem modelowanie zjawisk rzeczywistego świata w uporządkowany, hierarchiczny sposób — od idei do szczegółów technicznych. Wśród najważniejszych języków należy wymienić C++ (choć nie jest to język czysto obiektowy) i Javę; ze względów historycznych i poznawczych należałoby dodać Simulę 67 i Smalltalk.

## Programowanie funkcyjne

Tu po prostu składamy i obliczamy funkcje, w sensie podobnym do funkcji znanych z matematyki. Nie ma stanu maszyny — nie ma zmiennych mogących zmieniać wartość. Nie ma zatem „samodzielnie biegnącego” czasu, a jedynie zależności między danymi. Nie ma efektów ubocznych. Można by wręcz sądzić, że programowanie funkcyjne zdefiniowane zostało jako poważne ograniczenie paradygmatu imperatywnego... Rzeczywiście, do pewnego stopnia zwykle języki imperatywne zawierają w sobie „podjęzyk” funkcyjny (można przecież tworzyć, składać i wywoływać funkcje). Prawdziwe języki funkcyjne oferują jednak znacznie więcej, m.in. rekurencyjne struktury danych i możliwość operowania funkcjami wyższego rzędu.

Programowanie funkcyjne nie doczekało się takiej popularności jak dwa poprzednie paradygmaty, choć są dziedziny, gdzie jego pozycja jest przyzwoita (np. język Erlang używany w telekomunikacji oraz język K używany do obliczeń finansowych). Najśłynniejsze języki funkcyjne żyły lub żyją przede wszystkim w środowiskach akademickich: Lisp, Scheme (potomek Lispu), ML, Ocaml (potomek ML-a), Miranda, Haskell.

## Programowanie w logice

Podobnie jak w programowaniu funkcyjnym, nie „wydajemy rozkazów”, a jedynie opisujemy, co wiemy i co chcemy uzyskać (z tego powodu języki funkcyjne i logiczne nazywa się łącznie językami deklaratywnymi). Na program składa się zbiór zależności (przesłanki) i pewne stwierdzenie (cel). Wykonanie programu to próba udowodnienia celu w oparciu o podane przesłanki, a więc pewien rodzaj automatycznego wnioskowania; obliczenia wykonywane są niejako „przy okazji” dowodzenia celu.

Język programowania w logice (a ściślej — jego interpreter) to właściwie system automatycznego dowodzenia twierdzeń, działający w oparciu o nieco uproszczony rachunek predykatów. Kluczowym pojęciem jest tu rezolucja, realizowana m.in. przy pomocy unifikacji i nawrotów. Zastosowania programowania w logice obejmują przede wszystkim sztuczną inteligencję (np. systemy ekspertowe, rozpoznawanie obrazów) i przetwarzanie języka naturalnego. Językiem, który uzyskał największą popularność, jest Prolog. Historycznie ważny jest również Planner — wcześniejszy i bardziej złożony od Prologu.



## 2.5 Kacper S Cechy programowania deklaratywnego.

a

# Rozdział III

## Pytania - reszta

### 1    Jezyk C i C++

#### 1.1    Instrukcje sterujące w języku C.

a

## 1.2 Zarządzanie pamięcią w języku C.

a

### 1.3 Budowa, obsługa i formatowanie łańcuchów znakowych w języku C.

a

## 1.4 Zasięg i czas życia obiektów w języku C++.

a

## 1.5 Obsługa wyjątków w języku C++.

a

## **1.6    Definicje obiektu, klasy i szablonu klasy w języku C++.**

a

## 2 Algosy

### 2.1 Algorytmy sortujące.

a



## 2.2 Algorytmy zachłanne.

a

## **2.3    Metoda „dziel i zwyciężaj” konstruowania algorytmów.**

a

## 2.4 Struktura kopców binarnych.

a

## 2.5 Algorytmy wyszukiwania najkrótszej ścieżki w grafie.

a

## 2.6 Sposoby implementacji słownika.

a

## 2.7 **Tablice mieszające.**

a

## 2.8 Algorytmy Monte Carlo oraz algorytmy Las Vegas.

a

## **2.9 Metody rozwiązywania rekurencji. Rekurencje Flawiusza i wieża w Hanoi.**

a



## **2.10 Algorytmy Euklidesa. Algorytmy faktoryzacji.**

a

## **2.11    Metody reprezentacji grafów w komputerze.**

a

## **2.12 Droga i cykl Eulera. Droga i cykl Hamiltona.**

a

### **2.13**   **Drzewo spinające graf.**

a

### **3 Teoria obliczalności czy coś**

#### **3.1 Pojęcia P, NP, NP-zupełne.**

a

## 4 Automaty i inne takie

### 4.1 **Deterministyczne i niedeterministyczne automaty skończone.**

a

## 4.2 Automaty z epsilon przejściami, wyrażenia regularne.

a

### 4.3 **Kompilacja: gramatyka bezkontekstowa, skaner, parser, błędy.**

a



## 5 Podstawy komputera i systemy operacyjne

### 5.1 Systemy liczbowe i konwersje pomiędzy nimi.

a

## 5.2 Sposoby cyfrowej reprezentacji liczby całkowitej i rzeczywistej.

### Liczby całkowite

**Kod ZM (kod znak-moduł)** Sprawa w kodzie ZM jest w miarę prosta i klarowna. Najstarszy bit  $b_{n-1}$  dla  $n$ -bitowej liczby jest bitem znaku i określa czy liczba jest dodatnia czy ujemna:

- 0 - liczba dodatnia,
- 1 - liczba ujemna.

Bit y od  $b_{n-1}$  do  $b_0$  odpowiadają za kodowanie wartości samej liczby. Wzór na obliczenie wartości liczby zakodowanej w **ZM**:

$$L_{ZM} = (-1)^{b_{n-1}} \cdot (b_{n-2}2^{n-2} + \dots + b_22^2 + b_12^1 + b_02^0)$$

Przykładowe kodowanie liczby na ośmiu bitach w kodzie **ZM**:

$$\begin{aligned} 26 &\longrightarrow 00011010 \\ -26 &\longrightarrow 10011010 \end{aligned}$$

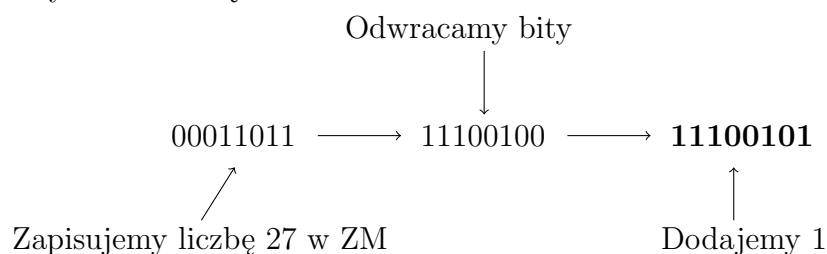
Proste, logiczne, fajne. Pytania, problemy? To jedziemy dalej.

**Kod U2 (kod uzupełnień do 2)** Tutaj sprawa się nieco komplikuje z zapisem liczb ujemnych. Bit  $b_{n-1}$  ma wagę  $-2^{n-1}$  co sprawia, że musimy bitowo tak jakby zapisać odwrotność liczby, którą chcemy reprezentować jako ujemna (i dodać 1, żeby się wszystko zgadzało). W zapisie liczb dodatnich zapis jest identyczny jak w **ZM** - na najstarszym bicie musimy tylko zachować 0.

Istnieje prosty algorytm konwersji na U2 z wykorzystaniem ZM:

1. Zapisać moduł liczby w ZM,
2. Dokonać inwersji bitów (0 na 1 i 1 na 0),
3. Zwiększ wynik dodając 1.

Przykład z liczbą -27 na 8 bitach:



**Liczby rzeczywiste**

**Zapis stałopozycyjny** Do zapisu liczby stałoprzecinkowej przeznaczona jest z góry określona liczba bitów, a pozycję przecinka ustala się arbitralnie, w zależności od wymaganej dokładności, wolne bity uzupełniając zerami. Do reprezentacji liczb ze znakiem stosuje także kod U2.

Liczba  $6,25 = 110,01_{(2)}$  zapisana na 8 bitach gdy część ułamkowa zajmuje 3 najmłodsze bity, ma postać:

$$\begin{array}{c} \text{część ułamkowa} \\ \underbrace{00110}_{\text{część całkowita}}010 \end{array}$$

A w reprezentacji U2 będzie miała postać:

$$11001110$$

Część całkowita liczby zachowuje się identycznie jak w przypadku zwykłych liczb całkowitych, natomiast bity w części ułamkowej posiadają wagi  $2^{-1}$ ,  $2^{-2}$ , itd. - czyli  $\frac{1}{2}$ ,  $\frac{1}{4}$ , ..., więc ilość bitów w części ułamkowej wpływa na precyzję zapisu.

**Zapis zmiennopozycyjny** Liczba zmiennoprzecinkowa jest komputerową reprezentacją liczb rzeczywistych zapisanych w postaci wykładniczej o podstawie 2. Przykładowa notacja:

$$(-1)^Z \cdot M \cdot 2^C = (-1)^Z \cdot (1 + m) \cdot 2^{c-BIAS}$$

gdzie:

$(-1)^Z$  - znak liczby

$M = 1 + m$  - znormalizowana mantysa (liczba spełniająca warunek:  $1 \leq M \leq 2$ ). Ponieważ przed przecinkiem stoi zawsze 1, więc można ją przedstawić w postaci  $1 + m$ , gdzie  $m$  jest liczbą ułamkową:  $0 \leq m \leq 1$ )

$C = c - BIAS$  - cecha (liczba całkowita), która dzięki zastosowaniu stałej BIAS pozwoli przedstawić cechę w postaci różnicy  $c - BIAS$  ( $c$  jest liczbą całkowitą dodatnią, tzw. spolaryzowana cechę)

$BIAS$  - stała (liczba całkowita BIAS zależna od danej implementacji – rozwiązuje problem znaku cechy)

Kodujemy wyłącznie:

**z** - bit znaku

**m** - mantysę pomniejszoną o 1

**c** - cechę przesuniętą o BIAS

Założmy, że operujemy następującym zmiennopozycyjnym formatem zapisu liczby rzeczywistej:

- na zapis przeznaczamy 16 bitów
- najstarszy bit ( $b_{15}$ ) to bit znaku (będziemy stosować kod ZM)
- kolejne 6 bitów ( $b_9-b_{14}$ ) to mantysa
- pozostałe bity ( $b_0-b_8$ ) są przeznaczone na zapis cechy i przyjmijmy, że  $BIAS=9$

Przedstawimy liczbę  $+0,0224609375$  w powyższym formacie. Naszą liczbę zapisujemy w systemie binarnym w postaci wykładniczej o podstawie 2, przesuwamy przecinek zapisując ją w notacji wykładniczej:

$$0,0224609375 = 0,0000010111_{(2)} = 1,0111_{(2)} \cdot 2^{-6}$$

Z tego wynika, że:

- Znak:  $(-1)^0$
- Mantysa:  $1.\underline{0111}_2$
- Cecha:  $-6 = 3 - 9 = 11_2 - BIAS$

Oto liczba  $0,0224609375$  zapisana w zadanym formacie:

$$\begin{array}{c} \text{mantysa} \\ \underbrace{0 \overbrace{011100}^{\text{mantysa}} \underbrace{000000011}_{\text{cecha}}}_{\text{bit znaku}} \end{array}$$

### **5.3 Wielowarstwowa organizacja oprogramowania komputera.**

a

## 5.4 Procesy, zasoby i wątki.

a

## 5.5 Planowanie przydziału procesora, priorytety, wywłaszczanie oraz planowanie.

a

## 5.6 Zarządzanie pamięcią operacyjną.

a



## 5.7 Problem zakleszczenia, algorytm Bankiera.

a

## 6 Inżynieria Oprogramowania

### 6.1 Standardowe metodyki procesu wytwórczego oprogramowania.

a

## **6.2    Metodyki zwinne (agile).**

a

### **6.3    Metody testowania oprogramowania.**

a

## **6.4 Walidacja i weryfikacja oprogramowania.**

a

## 6.5 Diagramy UML (przypadków użycia, klas, aktywności, sekwencji, stanów, obiektów, wdrożenia).

a

## **6.6**   **Wzorce projektowe programowania obiektowego.**

a

## **6.7    Wzorce architektoniczne.**

a



## 7 Systemy wbudowane i elektronika

### 7.1 Różnice pomiędzy obsługą zdarzeń w przerwaniach sprzętowych a obsługą zdarzeń w pętli programowej.

a

## 7.2 Stosowalność systemów opartych o mikrokontrolery vs stosowalność typowych komputerów (stacjonarnych i laptopów).

a

**7.3 Dekoder, multiplekser i demultiplekser: budowa, zasada, działania, przeznaczenie/zastosowanie.**

a

**7.4 Podstawowe układy budujące system mikroprocesorowy i sposób wymiany informacji pomiędzy nimi.**

a