Zajebiste odpowiedzi typie

makaki są sztos

Sieć Aptek

i inne małpy

June 25, 2022

Spis treści

Rozdział I

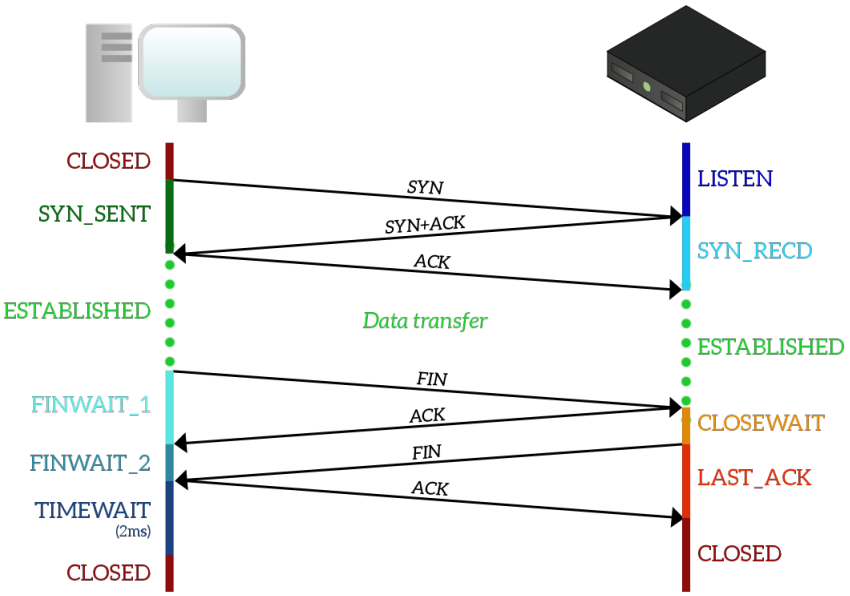
Pytania - dr. hab. Bogdan Księżopolski

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.

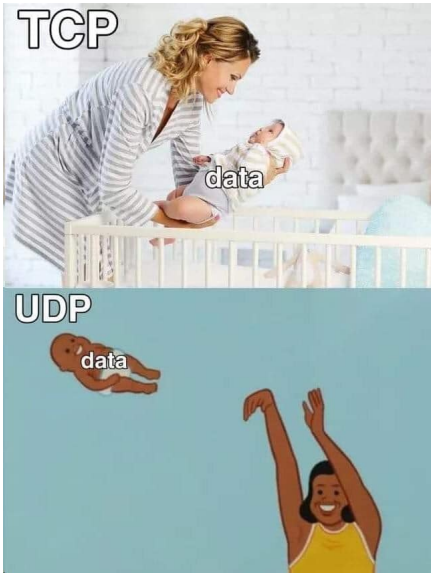


Rysunek I.1: TCP

Jest wysoce niezawodny, ponieważ wykorzystuje 3-drożną kontrolę uzgadniania, przepływu, błędu 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 zdążyć, ze 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 Rafał Modele sieci komputerowych.

a

1.4 Rafał Porównanie protokołów IPv4 i IPv6.

a

1.5 Format pakietu IP (poszczególne pola, zastosowanie).

<table>
<tr><th>0 – 3</th><th>4 - 7</th><th>8 - 15</th><th>16 - 18</th><th>19 - 23</th><th>24 - 31</th></tr>
<tr><td>Wersja</td><td>IHL</td><td>Typ usługi</td><td colspan="3">Długość całkowita</td></tr>
<tr><td colspan="3">Identyfikator</td><td>Flagi</td><td colspan="2">Przesunięcie fragmentu</td></tr>
<tr><td>Czas życia</td><td colspan="2">Protokół</td><td colspan="3">Suma kontrolna nagłówka</td></tr>
<tr><td colspan="6">Adres źródłowy</td></tr>
<tr><td colspan="6">Adres docelowy</td></tr>
<tr><td colspan="3">Opcje</td><td colspan="3">Dopełnienie</td></tr>
<tr><td colspan="6">Dane</td></tr>
</table>

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 unikatory 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 tą 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 sobie dane zapisane w formacie binarnym

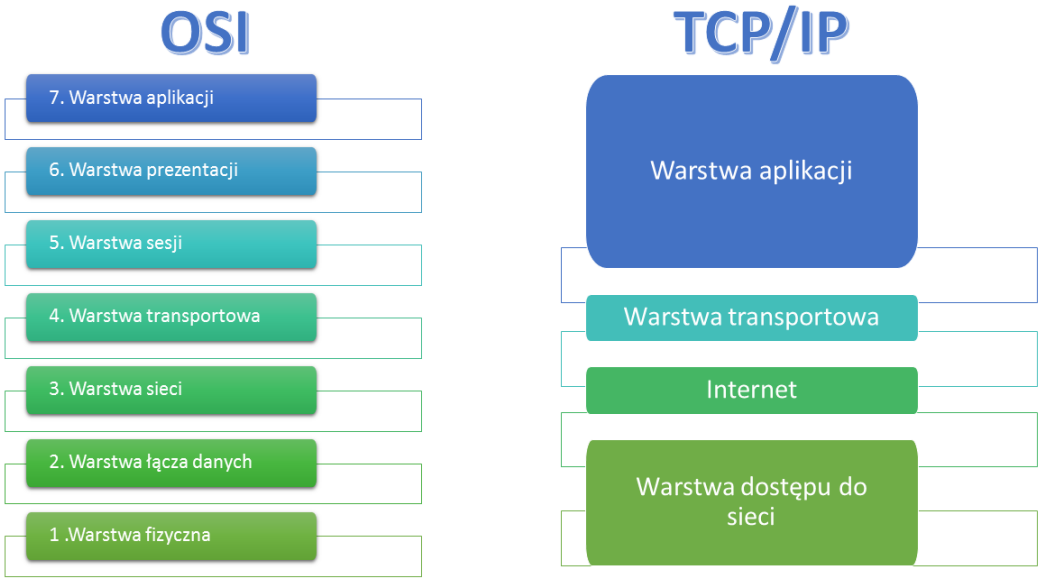
1.6 Rafał Ethernet.

a

1.7 Rafał 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 sieciową 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 łącza 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 Rafał 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**.

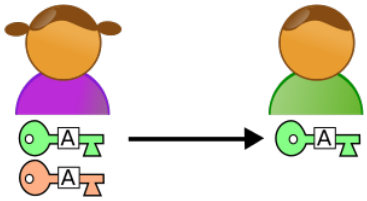
Kryptografia asymetryczna

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.

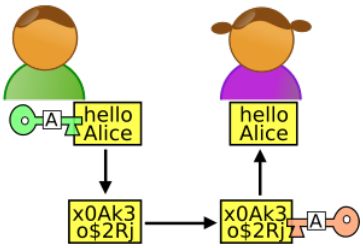
Kryptografia asymetryczna opiera się na funkcjach jednokierunkowych – takich, które da się łatwo wyliczyć w jedną stronę, ale bardzo trudno w drugą. Np. mnożenie jest łatwe, a rozkład na czynniki (z ang. faktoryzacja) trudny (na czym przykładowo opiera się **RSA**). Potęgowanie modulo jest łatwe, a logarytmowanie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



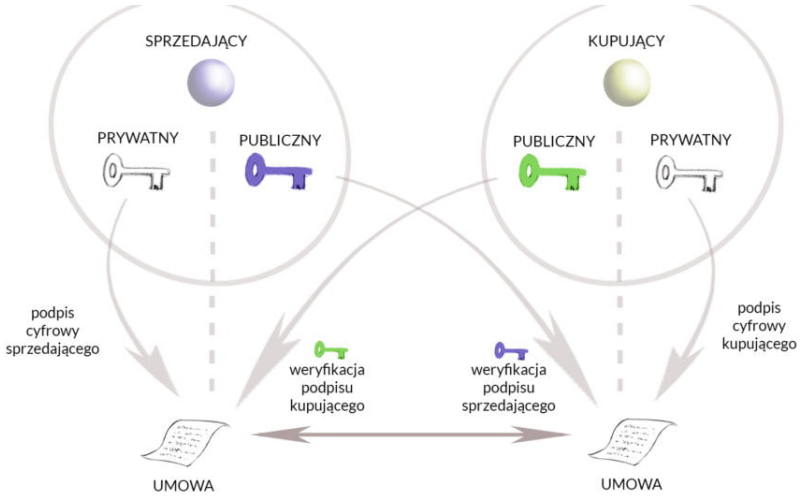
Rysunek I.7: Krok 1: Alice przesyła do Boba swój klucz publiczny



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

względu 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 Metody kontroli dostępu w systemach IT.

a

2.5 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 Codda 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/atrybutów/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 Codda – jest to zestaw 13 zasad stworzonych przez Edgara F. Codda – 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
- Projekcja - `SELECT` - pominięcie z wyniku pewnych kolumn
- Opracje 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 Model obiektowo-relacyjny baz danych, inne modele danych.

Model relacyjny

W relacyjnym modelu baz danych informacja jest zapisywana w tabeli w formie wierszy. Tabele tworzą między sobą powiązania zwane relacjami. Dużo dużo więcej jest napisane podrozdział wyżej

Model obiektowy

Model obiektowy łączy cechy programów komputerowych tworzonych w językach programowania obiektowego z cechami aplikacji bazodanowych. Obiekt w bazie reprezentuje obiekt w święcie rzeczywistym. Występują tutaj pojęcia klasa, obiekt, dziedziczenie, hermetyzacja.

- oprócz danych dostępne są także operacje na danych
- użytkownik może definiować własne typy danych
- aplikacja oraz BD może być tworzona przy pomocy jednego języka
- aplikacja korzystająca z bazy nie wymaga zmiany modelu relacyjnego na obiektowy
- nie występuje problem z optymalizacją zapytań
- nie występuje ujednolicony standard modelu obiektowego

Model obiektowo-relacyjny

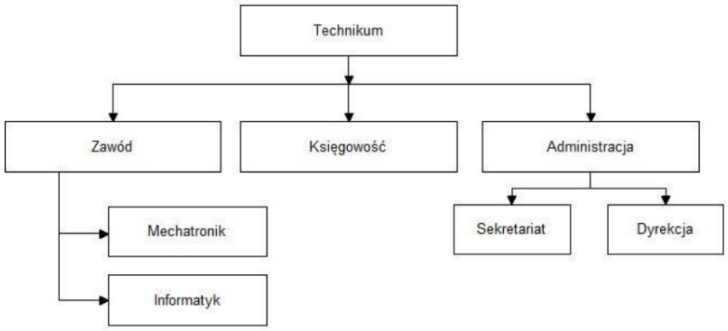
Model ten łączy koncepcję modelu relacyjnego oraz obiektowego. Jego ideą jest połączenie najlepszych cech obydwu modeli.

- występują tutaj tabele które mogą mieć pola o typie ATD (Abstract Data Type). Umożliwiają one przechowywanie w bazie np. list, kolejek, drzew - obiektów którym możemy przypisać pewne cechy i zachowanie - mają swoje własne implementacje sortowania, wyszukiwania itp
- przystosowanie do przechowywania multimediów
- ten model zwraca dane jako tabele krotek (jak w bazie relacyjnej), a nie jako kolekcję obiektów
- programista musi zamienić odpowiedź z bazy danych będącą w postaci krotek (jak w relacyjnej) na postać obiektową

Model hierarchiczny

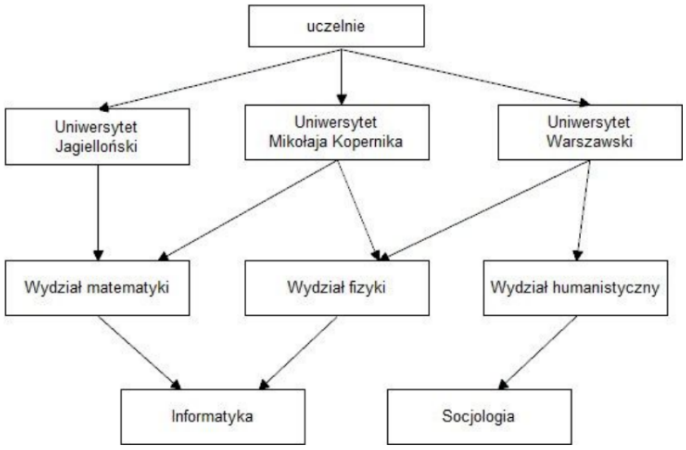
W tym modelu przechowywane dane są zorganizowane w postaci drzewa. Informacja jest zawarta w dokumentach oraz w strukturze drzewa (podobnej do drzewa folderów na dysku komputera).

- dane są przedstawione w postaci drzewa (najłatwiej to porównać do XML)
- informacje są przechowywane w strukturze drzewa
- tabele muszą być zawsze ze sobą powiązane, czyli podrzędne z nadrzędnym (np. uczeń musi być podpięty pod jakąś szkołę)
- usunięcie tabeli nadrzędnej powoduje usunięcie wszystkich tabel podrzędnych



Model sieciowy

Połączenia między dokumentami tworzą sieć. Informacja jest zawarta w dokumentach oraz w przebiegu połączeń sieci.



- jest to rozwinięcie modelu hierarchicznego gdzie tabele podrzędne mają wiele tabel nadrzędnych (skutkiem czego jest stworzenie grafu/sieci)
- można bezpośrednio odczytywać każdą tabele (nie trzeba przechodzić od korzenia do liścia jak w hierarchicznym)
- relacje/zależności między tabelami są nazywane strukturami
- w strukturze można wyłonić jedną tabelę główną z której to można przejść do tabel podrzędnych
- struktura jest relacja 1:N
- zmiany w bazie są bardzo trudne do wykonania

Model semistrukturalny

- polega na przedstawieniu danych w postaci modelu XML
- przedstawia on dane w postaci drzewiastej (czyli jest hierarchiczny)
- zapytania są w postaci ścieżki
- atrybuty znacznika przechowywują dodatkowe informacje (cechy)
- dane w postaci XML można łatwo umieścić na stronie WWW (tylko trzeba określić sposób wyświetlania)

- baza w postaci /xML może być łatwo przekonwertowana na obiekt

Model jednorodny

Wszystkie dane upchane w jednej tabeli, łatwo szukać danych po unikalnej wartości (ale ciężej po czymś nieunikalnym np nazwisko), taka forma posiada dużo duplikatów

1.3 Składnia podstawowych zapytań języka SQL.

DDL - Data Definition Language

Jak zapamiętać skrót:
Definition = DEFINIOWANIE - TWORZENIE STRUKTUR
W skład DDL wchodzą `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 wchodzą `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 wchodzą `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 niefunkcjonalnych (tryb graficzny, platforma sprzętowa itd.)
- **Opracowanie konceptualnego modelu danych**, czyli wyodrębnienie i zdefiniowanie encji, związków między nimi oraz atrybutów (taka jakby luźna forma)
  Projektowanie modelu konceptualnego 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 konceptualny 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 stronę 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. Konkretne 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 Problemy indeksowania baz danych, rodzaje indeksów, indeksy typu B+ drzewo.

Problemy indeksowania baz danych

Problem z przeszukiwaniem baz danych polega na tym, że... tabele w MySQL (a także w innych RDBMS) **nie są posortowane według kolumn**, dzięki którym wyciągamy odpowiednie dane. Dane ulegają aktualizacji, usuwaniu i ciągle dodawane są nowe. Jedyna „kolejność" to często klucz główny, który jest de facto indeksem. Nie jest to przydatna kolejność kiedy szukamy danych nie po kluczu a po jakimś innym polu np:

```
SELECT nazwa_produktu FROM produkty WHERE cena = 128;
```

Jeżeli mielibyśmy te nasze produkty posortowane według ceny, to znalezienie było by banalne. W przypadku kiedy dane są posortowane po kluczu głównym trzeba „sprawdzić" wszystkie rekordy i nie mamy możliwości ułatwienia sobie zadania bo produkty mogą mieć przecież różną cenę. Mówi wtedy że dokonujemy **pełnego skanu tabeli** (ang. full table scan), który działa niekorzystnie na wydajność – zabiera po prostu za wiele czasu. Sztuka optymalizacji za pomocą indeksowania polega na unikaniu tego pełnego skanu. Działanie indeksu polega na tym, że zawiera w przystępnej postaci informację, w których komórkach pamięci znajdują się produkty o cenie 128. Z tego korzysta RDBMS – najpierw uderza do indeksu a kiedy już wie skąd ma pobrać dane to od razu przechodzi do odpowiednich miejsc bez przeszukiwania całej tabeli i zwraca wynik użytkownikowi.

Zalety indeksów: znacznie przyspieszają pewne SELECTy
Wady selectów: spowalniają dodawanie, modyfikowanie i usuwanie danych (w przypadku edycji tabeli należy również zaktualizować jej indeks)

Rodzaje indeksów

- **Ze względu na liczbę wskazań indeksu (ile rekordów wskazuje indeks)**:

  indeks rzadki - zawiera rekordy tylko dla wybranych rekordów indeksowanego pliku (indeksowana jest tylko część pliku)

  indeks gęsty - zawiera rekord indeksu dla każdego rekordu indeksowanego pliku (indeksowany jest cały plik)

- **Ze względu na liczbę poziomów indeksu (indeks może wskazywać na inny indeks)**:

  indeks jednopoziomowy - do pliku z danymi tworzony jest tylko jeden indeks na dany atrybut

  indeks wielopoziomowy - do pliku z danymi tworzonych jest wiele indeksów na dany atrybut (tworzenie indeksu do indeksu)

- **Ze względu na charakterystykę indeksowanego atrybutu**:

  indeks podstawowy - - jest on założony na atrybucie (unikalnym) który umożliwia porządkowanie pliku (np. klucz główny, PESEL, dane osobowe). Jest to indeks rzadki.

  indeks zgrupowany - jest on zakładany na atrybucie który nie jest unikalny (również umożliwia porządkowanie pliku) (np. imię, wartość liczbowa). Ten indeks

wskazuje na pierwszy blok w którym występuje dana wartość (patrz wartość 2, 3, 4). Jest to indeks rzadki.

indeks wtórny - jest on zakładany na atrybucie który nie jest atrybutem uporządkowanym (czyli taki według którego plik nie jest uporządkowany). Jest to indeks gęsty.

- **Ze względu na liczbę indeksowanych atrybutów**

  indeks z kluczem prostym - indeksowany jest tylko jeden atrybut/klucz

  indeks z kluczem złożonym - indeksowane jest wiele atrybutów jednocześnie (klucz)

- **Ze względu na zarządzanie strukturą**

  indeksowanie statyczne - indeks nie ma mechanizmów które modyfikują strukturę czyli np. usuwanie pozostawia puste rekordy w indeksie, a dodawanie polega na dodawaniu rekordów do specjalnego miejsca zwanego "blokiem przepełnienia"

  indeksowanie dynamiczne - indeks ma mechanizmy które modyfikują strukturę (i nie ma tych problemów co w indeksowaniu statycznym)

Indeksy typu B+ drzewo

- to rodzaj indeksowania dynamicznego wielowarstwowego które jest przedstawione w postaci drzewa
- każdy indeks posiada wskaźnik (który nie jest liściem) wskazuje na inny indeks
- wskaźnik po lewej stronie klucza (wartości atrybutu) zawiera wartości mniejsze, a po prawej większe
- węzły tego drzewa mogą pomieścić określona ilość rekordów indeksu
- gdy rekordów w węźle jest za dużo to następuje dzielenie węzła, a gdy w dwóch węzłach jest mniej niż połowa to następuje łączenie węzłów
- B+ drzewo rośnie do góry (tak samo jak B)
- jest to drzewo zrównoważone (odległość od korzenia do liścia wszędzie jest taka sama)
- węzły które nie są liśćmi nie posiadają wartości (w B-drzewie posiadały)
- wartości kluczy które są w węzłach wewnętrznych powtarzają się w liściach (np. (2,4) C (1,2,3,4,5)
- liście są ze sobą połączone (liść o mniejszych wartościach ma wskaźnik do liścia o większych wartościach)
- Wyszukiwanie informacji w drzewie wynosi O(logn)

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**
- **A**tomicity - atomowość - wykona się wszystko albo nic (zbiór operacji = całość);
- **C**onsistency - spójność - baza przed i po zmianach jest spójna;
- **I**solation - izolacja - równolegle wykonywane transakcje nie mają na siebie wpływu;
- **D**urability - 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 Założenia paradygmatu programowania obiektowego.

Paradygmat obiektowy zakłada modułowość, czyli tworzenie programu z wcześniej zdefiniowanych komponentów (np. klasy).

Główne założenia paradygmatu programowania obiektowego:

1. **Abstrakcja** - to przedstawienie obiektu ze świata rzeczywistego i jego cech jako uproszczony model/szablon (tutaj klasa i obiekty) wykorzystywany w problemie programistycznym.

2. **Hermetyzacja (enkapsulacja)** - to ukrycie implementacji przed użytkownikiem (prywatne pola - publiczne metody). Cytując kogoś mądrego: "Hermetyzacja polega na tym, że klasy są zamkniętymi "czarnymi skrzynkami" - inne klasy nie muszą wiedzieć dokładnie co się dzieje w środku, mogą wchodzić z nimi w interakcje poprzez elementy, które wystawiają publicznie (publiczne pola i metody)".

3. **Dziedziczenie** - jest to tworzenie hierarchii klas. Klasa dziedzicząca otrzymuje pola i metody (te może zmieniać) "klasy matki". Tworzenie bardziej wyspecjalizowanego obiektu z bardziej ogólnego. (np. klasa jabłko dziedziczy cechy po klasie owoc)

4. **Polimorfizm** - polega na dostosowaniu zachowania obiektów do oczekiwań użytkownika. Inaczej mówiąc gdy grupa obiektów dziedziczy metodę od innej klasy. Każda klasa która odziedziczyła tę metodę to ją zmieniła. Wywołanie tej odziedziczonej metody na poszczególnych obiektach wywoła wykonanie instrukcji zaimplementowanych dla każdego obiektu. (np. dla każdej figury funkcja obliczania pola będzie inna)

2.2 Idea dziedziczenia i polimorfizmu w programowaniu.

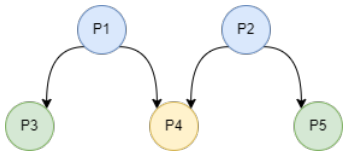
Dziedziczenie - jest to tworzenie hierarchii klas. Klasa dziedzicząca otrzymuje pola i metody (te może zmieniać) "klasy matki". Tworzenie bardziej wyspecjalizowanego obiektu z bardziej ogólnego. (np. klasa jabłko dziedziczy cechy po klasie owoc). Umożliwia ona większą czytelność kodu który jednocześnie staje się prostszy (bo po co pisać 2 razy to samo)

Polimorfizm - polega na dostosowaniu zachowania obiektów do oczekiwań użytkownika. Inaczej mówiąc gdy grupa obiektów dziedziczy metodę od innej klasy. Każda klasa która odziedziczyła tę metodę to ją zmieniła. Wywołanie tej odziedziczonej metody na poszczególnych obiektach wywoła wykonanie instrukcji zaimplementowanych dla każdego obiektu. (np. dla każdej figury funkcja obliczania pola będzie inna)

Polimorfizm umożliwia nie tylko wykonywanie odpowiednio metod dla obiektów pochodnych ale również przypisanie do bardziej ogólnego typu danych (np. Owoc jablko= new Jablko();)

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. Nastepnie obliczając drugi głowny 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++ (chociaż 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 zwykłe 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). Najsłynniejsze 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 Cechy programowania deklaratywnego.

W programowaniu deklaratywnym programista deklaruje mając jakiś element co chce z niego uzyskać (nie jest mówione jak to ma zostać wykonane, tylko jest mówione jaki ma być efekt).

Cechy:

- wzorowany na językach naturalnych
- nieliniowe wykonanie (komenda końcowa może modyfikować początek)
- bardzo częste wykonywanie rekurencji zamiast pętli
- definiuje problem i szuka odpowiedź na zadane pytanie

Do programowania deklaratywnego zalicza się:

- języki funkcyjne - zamiast instrukcji używa się funkcji (cały program to funkcja). Przykładem jest Haskell w którym Wszystko opiera się na wyrażeniach Lambda. W tym języku funkcje są sobie równe jeśli dają takie same odpowiedzi.
- języki w logice - zamiast instrukcji używa się zdań logicznych. np. w Prologu zdania logiczne na początku opisują zależności (między elementami) aby na końcu zapytać czy podane zdanie jest prawdziwe (oczywiście trzeba na koniec podać to zdanie). Tutaj wszystko jest wyrażeniem logicznym

Rozdział III

Pytania - reszta

1 Jezyk C i C++

1.1 Instrukcje sterujące w języku C.

Instrukcja sterująca jest to każda instrukcja, która może zmienić kolejność wykonywania linii w programie (np. `if` czy `for`). Dzięki nim program nie jest odczytywany jak tekst od góry do dołu, a może zawierać rozgałęzienia, powtórzenia operacji lub przedwczesne zakończenie.

- `IF` - instrukcja warunkowa

```
int a = 4;
int b = 6;

if (a==b) {
    printf ("a == b\n");
} else {
    printf ("a != b\n");
}
```

- `SWITCH` - instrukcja warunkowa (warto zwrócić uwagę, że bez `BREAK`, program będzie sprawdzał kolejne warunki `CASE`, co może być niepożądanym zachowaniem)

```
unsigned int dzieci = 3, podatek=1000;
switch (dzieci) {
    case 0: break; /* brak dzieci - czyli brak ulgi */
    case 1: /* ulga 2% */
    podatek = podatek - (podatek/100 * 2);
        break;
    case 2: /* ulga 5% */
        podatek = podatek - (podatek/100 * 5);
        break;
    default: /* ulga 10% */
        podatek = podatek - (podatek/100 * 10);
        break;
}
```

- `FOR` - instrukcja petli

```
for (int a=1; a<=10; ++a) {
    printf ("%d\n", a*a);
}
```

- `WHILE` - instrukcja petli

```
while (a <= 10) {
    printf ("%d\n", a*a);
    ++a;
}
```

- `DO␣WHILE` - instrukcja pętli, różni się tym od while że pierwsze wykonanie jest bezwarunkowe, dzięki temu kod pętli wykona się co najmniej jeden raz

```
do {
    printf ("%d\n", a*a*a);
    ++a;
} while (a <= 10);
```

- `BREAK` - przerywa wykonanie pętli, w której się znajduje. W przypadku zagnieżdżonych pętli, przerwana jest tylko "najniższa" petla w hierarchii

```
for (int i = 0; ;++i) { /* pomimo braku warunku zakonczenia
                           BREAK zapewnia,
                           ze petla jest skonczona */
    if (i == 5) break;
}
```

- `CONTINUE` - przechodzi do kolejnej iteracji pętli pomijając pozostałe instrukcje dla bieżącej iteracji

```
for (i = 1 ; i <= 50 ; ++i) {
    if (i%4==0) continue ; /* zapobiega wyswietlaniu liczbom
                              podzielnym przez 4 */
    printf ("%d,␣", i);
}
```

- `EXIT` - zakańcza działanie programu zwracając kod błędu. Kod 0 oznacza poprawne wykonanie programu.

```
exit(0);
```

1.2 Zarządzanie pamięcią w języku C.

Zarządzanie pamięcią odbywa się za pomocą 3 możliwych operacji: alokacja, zmiana rozmiaru, wzolnienie pamięci. Ważne jest prawidłowe operowanie zwalnianiem pamięci, ponieważ błędy programistyczne na tej płaszczyźnie mogą doprowadzić do powstania zaalokowanych bloków pamięci, do których program nie ma już odwołania (wyciek pamięci). Wycieki pamięci powodują stałe ograniczenie zasobu pamięci na czas wykonywania programu, co może prowadzić do spowolnienia programu a nawet niepowodzenia w jego wykonaniu.

Możwile operacje zarządzania pamięcią w języku C to:

- `malloc` - przydziela obszar pamięci o podanym rozmiarze i zwraca do niego "surowy" wskaźnik. Aby było wiadomo jaki typ danych znajduje się pod podanym adresem należy podać typ używany do konwersji (podawany w nawiasie przed formułą). Świeżo zajęta pamięć oczywiście zawiera "śmieci" - losowe wartości.
  Podajemy ilość bajtów jaką chcemy zająć.

  ```
  int* x = (int*) malloc(sizeof(int));
  int* tab = (int*) malloc(sizeof(int)*n);
  ```

- `calloc` - robi to samo co malloc, ale uzupełnia każdy zaalokowany adres wartością 0. Przydatne jeżeli potrzebujemy tablicy z zerami, albo wstaźnika na pojedynczą zmienną 0.
  Podajemy następująco: ilość elementów, wielkość każdego elementu w bajtach.

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

- `realloc` - zmienia rozmiar wcześniej zaalokowanej pamięci. Nowo zaalokowana pamięć będzie miała najprawdopodobniej inny adres od pierwowzoru - będzie kopią (zależy od kompilatora).
  Podajemy następująco: wskaźnik i nowy rozmiar.

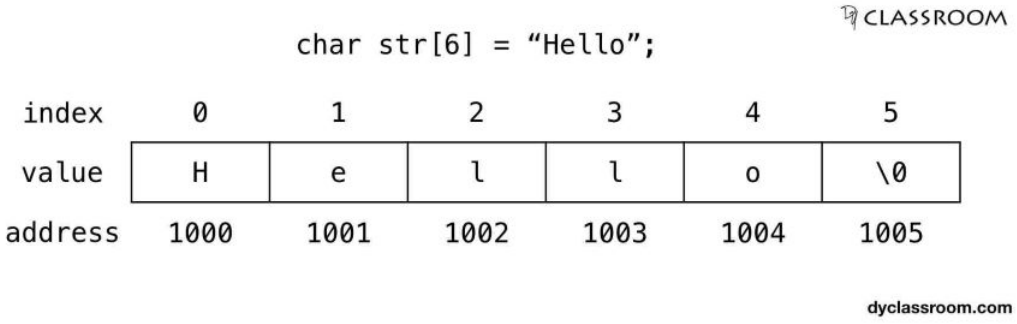
  ```
  int* tab = (int*) realloc(tab,sizeof(int)*n);
  ```

- `free` - służy do zwalniania dynamicznie zaalokowanej pamięci. Uwaga! Zwalnianie pamięci nie jest rekurencyjne, więc kolejność zwalniania ma znaczenie, aby nie doprowadzić do wycieków pamięci najpierw chcemy zwolnić elementy najniżej w hierarchii idąc w górę. Dla przykładu mając dynamicznie zaalokowaną klasę i jej atrybuty, chcemy zacząć od atrybutów i na końcu zwolnić pamięć jaką zajmuje element klasy.
  Podajemy wskaźnik na zmienną.

  ```
  free(tab);
  ```

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

Łańcuchy znaków są to tablice zmiennych typu char, ostatni element tablicy to "\0". Za ich pomocą reprezentowane są napisy. Napis 5-cio literowy musi być reprezentowany w co najmniej 6-cio elementowej tablicy (możliwa jest jego reprezentacja w dłuższej tablicy pod warunkiem zakończenia napisu symbolem - \0).



Rysunek III.1: Przykład reprezentacji napisu "Hello"

Budowa

Przykłady deklaracji łańcucha znakowego:

- przykład 1:

```
char string1[] = "A string declared as an array.\n";
```

- przykład 2:

```
char *string2 = "A string declared as a pointer.\n";
```

- przykład 3:

```
char string3[30];
strcpy(string3, "A string constant copied in.\n");
```

Możliwe jest również dynamiczne zaalokowanie łańcucha, przykład:

```
char **tab = (char**)malloc(sizeof(char*) * 3);
/* 3 lancuchy znakow */
for (int i=0; i<3; i++){
    tab[i] = (char*)malloc(sizeof(char) * 200);
    /* Po 200 bajtow na lancuch */
}
strcpy(tab[0], "abcxyz");
strcpy(tab[1], "zzzzzyyyyxxx");
strcpy(tab[2], "ab123");
printf("%s %s %s\n", tab[0], tab[1], tab[2]);
```

Obsługa łańcuchów

Do odsługi łańcuchów możemy zaliczyć następujące komendy:

- `strcmp` - służy do porównywania łańcuchów znaków. Jego działanie jest lekko odwrotne niż możnaby się spodziewać, ponieważ zwrócona wartość 0 oznacza identyczność łańcuchów. Funkcja zwraca wartość ¿0 jeżeli pierwszy nieidentyczny znak jest większy (wartościowo po ASCII) w pierwszym napisie. Analogicznie zwrócona wartośc ujemna oznacza, że pierwszy różny znak był mniejszy wartościowo (po ASCII) w pierwszym napisie.

```
if (strcmp(A, B) == 0){
    /* lancuchy sa identyczne */
}
else if (strcmp(A, B) > 0){
    /* lancuch A jest wiekszy w kolejnosci alfabetycznej */
}
else /* (strcmp(A, B) < 0) */{
    /* lancuch A jest mniejszy w kolejnosci alfabetycznej */
}
```

Formatowanie łańcuchów

Istnieje możliwość sparametryzowania istaniejących łańcuchów, w celu tym wykorzystuje się wstawki typu `%.2f`, gdzie po nich podawana jest wartośc. Niektóre ze wstawek mogą być modyfikowane np. `%.2f` to tak naprawdę zmodyfikowana `%f`, liczba zmiennoprzecinkowa zostala ograniczona do 2 miejsc po przecinku.

Możliwe wstawki to:

- `s` - wypisuje cały string (wypisuje całą tablicę typu char do symbolu końca linii)

```
printf("Ciag znakow ASCII: %s", "jakis tekst");
```

- `c` - int podany w argumencie zostanie zamieniony na znak ASCII

```
printf("Znak ASCII o kodzie 65: %c", 65);
```

- `d` - wypisuje liczbę w postaci dziesiętnej
  `printf("Liczba dziesietna calkowita ze znakiem: %d", -65);`

- `e` - liczba jest przedstawiana w postaci naukowej gdzie 0,45 to 4,5E-2
  `printf("Liczba dziesietna w zapisie naukowym: %e", -165.1235);`

- `f` - liczba (dziesiętna) jest przedstawiana w postaci zmiennoprzecinkowej
  `printf("Liczba dziesietna zmiennoprzecinkowa: %f", -165.1235);`

- `o` - liczba dziesiętna przedstawiona w systemie ósemkowym
  `printf("Liczba osemkowa calkowita bez znaku: %o", 64);`

- `u`- liczba (dodatnia, ujemna, zmiennoprzecinkowa) jest jako naturalna
  `printf("Liczba dziesietna calkowita bez znaku: %u", 65);`

- `x` - liczba dziesiętna przedstawiona w systemie szesnastkowym
  `printf("Liczba szesnastkowa bez znaku: %x", 346874);`

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

Zacięg i czas życia to elementy charakteryzujące okres życia każdaj zmiennej. Zasięg opisuje w jakim obrębie programu umożliwiony jest dostęp do zmiennej. Czas życia rozpoczyna się w miejscu alokacji zmiennej i trwa aż do momentu jest jawnego lub (częściej) niejawnego zdealokowania. W przypadku zmiennych zaalokowanych niedynamicznie ich zycie dopiega końca gdy program nie ma już dostępu do ich zasięgu.

Zniemme zaalokowane w sposób dynamiczny (wskaźniki) muszą zostać manualnie zwolnione za pomocą `free` (C i C++) lub `del` (C++).

Zasięg zmiennej można podzielić na:

1. **Zasięg głobalny** - (inaczej mówiąc dla całego programu) to zasięg zmiennej lub obiektu w całym pliku. Najczęściej gdy programy są pisane w postaci skryptu jako 1 plik w którym to takie zmienne są inicjalizowane na czas całego programu.

2. **Zasięg lokalny** - zakres zainicjalizowanych zmiennych bądź obiektów w bloku ogranicza się do tego bloku. Dotyczy to zwykłych bloków ({ }), pętli oraz warunków.

3. **Zasięg w ciele funkcji** - czas życia zmiennych oraz obiektów ogranicza się do czasu wykonywania funkcji. Poza nią zmienne są niedostępne, a po jej zakończeniu zwalniane.

1.5 Obsługa wyjątków w języku C++.

Obsługa wyjątków to nic innego jak typowa struktura "try - catch". Struktura ta zamyka wybrany zakres kodu w bloku `TRY`. W przypadku niepowodzenia wykonania bloku try wykonywany jest kod z bloku `CATCH`. Zazwyczaj w bloku `CATCH` następuje wyświetlenie błędu, wykonanie alternatywnej operacji do tej w bloku `TRY` lub zakończenie programu.

Po bloku `TRY` może występować wiele bloków `CATCH` z różnymi warunkami, warunki są sprawdzane po kolei i w wypadku natrafienia na pasujący wyjątek, wykonywany jest kod z tego bloku `CATCH`.

```
try{  // boimy sie niepowodzenia operacji fun() i fun2()
    fun();
    fun2(); //podejrzane funkcje
}
catch(std::string obj)
{
    /* tutaj mozemy wyswietlic blad
     lub wywolac alternatywne operacje */
}
```

Wyjątki mogą byś samodzielnie wywołane używając instrukcji `throw`.

```
try
{
    throw 123;
}
catch(int w)
{
    cout << w << endl; /* wypisze: 123 */
}
```

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

Obiekt to pojedyncza instancja konkretnej klasy. Każdy obiekt może posiadać atrybuty z unikalnymi wartościami np. (waga=59; wiek=24; kolor=różowy).

Klasa to szablon (przepis) na podstawie którego powstają obekty. Klasa może posiadać atrybuty (zmienne) oraz metody (funkcje) różnego typu dostępu. Możliwe są typy:

- `PRYWATNY␣(private)` - brak bezpośredniego dostępu poza danym obiektem
- `CHRONIONY␣(protected)` - jak private lecz dostęp do elementów klasy jest możliwy dodatkowo z klas dziedziczących po tej klasie.
- `PUBLICZNY␣(public)` - nieograniczony dostęp, jeżeli mamy dostęp do obiektu to i do jego wszystkich elementów typu public.

Wśród klas można dodatkowo wyodrębnić metody wirtualne deklarowane za pomocą instrukcji `virtual`, np.

```
virtual void a() = 0;
```

Metody wirtualne nie posiadają ciała.

Klasy zawierające co najmniej 1 metodę wirtualną to klasy abstrakcyjne, nie można tworzyć za ich pomocą obiektów i służą jedynie do dziedziczenia.

Specjalną klasą abstrakcyjną jest interfejs - klasa zawierająca **jedynie** metody wirtualne.

Szablon klasy to pewnego rodzaju wzorzec do tworzenia klas. Szczególnie użyteczny w przypadku gdy planowane klasy różnią się jedynie typem atrybutów. Stosowanie szablonów pozwala zminimalizować powtarzalność kodu.

Deklaracja szablonu:

```
template <typename T>
```

Przykład szablonu klasy:

```
class Punkt
{
    public:
    Punkt( T argX, T argY, T argZ )
    : x(argX), y(argY), z(argX)
    { }
    T x, y, z;
};
```

Wykorzystanie szablonu podczas tworzenia obiektu klasy:

```
Punkt<int> A(0, -10, 0);
Punkt<float> A(0.4, 10.5, 5.1);
```

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.

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

$$L_{ZM} = (-1)^{b_{n-1}} \cdot (b_{n-2}2^{n-2} + ... + b_2 2^2 + b_1 2^1 + b_0 2^0)$$

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

$$26 \longrightarrow \mathbf{0}0011010$$
$$-26 \longrightarrow \mathbf{1}0011010$$

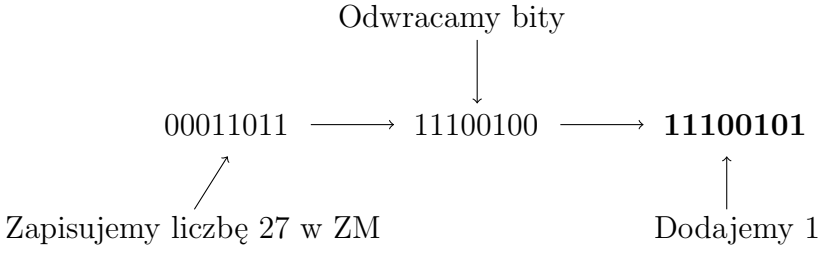
Proste, logiczne, fajne. Pytania, problemy? To jedziemy dalej.

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 częśd ułamkowa zajmuje 3 najmłodsze bity, ma postać:

$$\underbrace{00110}_{\text{część całkowita}}\overbrace{010}^{\text{część ułamkowa}}$$

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: $1 \leqslant M \leqslant 2$). Ponieważ przed przecinkiem stoi zawsze 1, więc można ją przedstawić w postaci $1 + m$, gdzie m jest liczbę ułamkową: $0 \leqslant m \leqslant 1$)

$C = c - BIAS$ - cecha (liczba całkowita), która dzięki zastosowaniu stałej BIAS pozwoli przedstawid 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

Załóżmy, że operujemy następującym zmiennopozycyjnym formatem zapisu liczby rzeczywistej:

- na zapis przeznaczamy 16 bitów
- najstarszy bit (b_{15}) to bit znaku (będziemy stosowad 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: $1.\underline{\mathbf{0111}}_2$
- Cecha: $-6 = 3 - 9 = 11_2 - BIAS$

Oto liczba 0,0224609375 zapisana w zadanym formacie:

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

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