Zagadnienie do egzaminu inżynierskiego 24/25

Semestr I / Semestr II

Wstęp do matematyki

1 Działania na liczbach i zbiorach

Zbiory to grupy elementów, a działania na nich pozwalają operować na tych grupach:

Działania na liczbach

Działania na liczbach to operacje arytmetyczne, które pozwalają przekształcać wartości liczbowe:

1. **Dodawanie (+)**
   Służy do łączenia dwóch liczb w jedną, wyrażając ich sumę.
   Przykład: 3+5=8
2. **Odejmowanie (-)**
   Polega na odjęciu jednej liczby od drugiej, co daje różnicę.
   Przykład: 10−4=6
3. **Mnożenie (×)**
   Umożliwia obliczenie iloczynu dwóch liczb.
   Przykład: 7×2=14
4. **Dzielenie (÷)**
   Dzieli jedną liczbę przez drugą, dając iloraz.
   Przykład: 20÷4=5
5. **Potęgowanie**
   Podnosi liczbę do określonej potęgi.
   Przykład: 3^2=9
6. **Pierwiastkowanie**
   Oblicza pierwiastek danej liczby.
   Przykład: sqrt(16)=4
7. **Modulo (%)**
   Oblicza resztę z dzielenia jednej liczby przez drugą.
   Przykład: 10mod 3=1

1. **Suma zbiorów (A∪B)**
   Łączy wszystkie elementy z dwóch zbiorów, eliminując powtórzenia.
   Przykład:
   Jeśli A={1,2,3}, B={3,4,5} to A∪B={1,2,3,4,5}
2. **Przecięcie zbiorów (A∩B)**
   Zawiera tylko te elementy, które należą jednocześnie do obu zbiorów.
   Przykład:
   A={1,2,3},B={3,4,5} $\Longrightarrow$ A∩B={3}

3. **Różnica zbiorów (A\B)**
   Zawiera elementy, które należą do zbioru A, ale nie należą do zbioru B.
   Przykład:
   $A=\{1,2,3\}, B=\{3,4,5\} \implies A \setminus B=\{1,2\}$
4. **Różnica symetryczna (A XOR B)**
   Obejmuje elementy, które należą do jednego z dwóch zbiorów, ale nie do obu jednocześnie.
   Przykład:
   $A=\{1,2,3\}, B=\{3,4,5\} \implies A \text{ XOR } B=\{1,2,4,5\}$
5. **Iloczyn kartezjański (A×B)**
   Tworzy zbiór wszystkich możliwych par uporządkowanych, gdzie pierwszy element pochodzi z A, a drugi z B.
   Przykład:
   $A=\{1,2\}, B=\{a, b\} \implies A \times B=\{(1,a),(1,b),(2,a),(2,b)\}$

Dopełnienie zbioru A'

Dopełnienie zbioru to zbiór wszystkich elementów, które nie należą do danego zbioru, ale są częścią **przestrzeni uniwersalnej** U patrz tu np. zasięg zbioru A i B . Przestrzeń uniwersalna U to zbiór zawierający wszystkie możliwe elementy rozważane w danym kontekście.

- $A'=U \setminus A$ (czyli wszystkie elementy z przestrzeni U, które nie należą do A).

Przykład:

Jeśli przestrzeń uniwersalna $U=\{1,2,3,4,5\}$ i zbiór $A=\{1,2,3\}$, to:
$A'=\{4,5\}$

Własności dopełnienia:

1. $A \cup A'=U$
   Suma zbioru i jego dopełnienia to cała przestrzeń uniwersalna.
2. $A \cap A'=\emptyset$
   Zbiór i jego dopełnienie nie mają wspólnych elementów.
3. $(A')'=A$
   Dopełnienie dopełnienia daje pierwotny zbiór.

Rozszerzenie działań z dopełnieniem

1. Suma zbiorów z dopełnieniem:

$A \cup B'$ – zbiór zawiera elementy należące do A lub nie należące do B.

2. Przecięcie zbiorów z dopełnieniem:

$A \cap B'$ – zbiór zawiera elementy należące jednocześnie do A i nie należące do B.

Przykład:

Jeśli U={1,2,3,4,5}, A={1,2,3}, B={3,4} to:

- A’={4,5}
- B’={1,2,5}
- A∩B’={1,2}

Wiec np. jak mamy A {-2,7} i B(-∞,6] to:

Dopełnienie A*:*

A’ to zbiór wszystkich liczb rzeczywistych, które **nie należą do A**:
A’=R\{−2,7}
A’=(−∞,−2)U(−2,7)U(7,∞)

Dopełnienie B’:

B’ to zbiór wszystkich liczb rzeczywistych, które **nie należą do B**:
B’=R\(−∞,6]
B’=(6,∞)

2 Rozwiązywanie równań i nierówności

Rozwiązywanie równań

Równanie to wyrażenie matematyczne, które zawiera znak równości (=) i ma postać:
f(x)=g(x)

Główne rodzaje równań:

1. **Równania liniowe**
   Mają postać ax+b=0, gdzie a≠0
   Rozwiązanie:
   x=−b/a
2. **Równania kwadratowe**
   Mają postać $ax^2+bx+c=0$
   Rozwiązanie metodami:
   - **Wzór kwadratowy:**

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

   - **Równanie iloczynowe:** gdy $ax^2+bx+c=0$ można rozłożyć na czynniki, np. (x−p)(x−q)=0 $\implies$ x=p lub x=q.

3. **Równania wielomianowe**
   Są to równania wyższych stopni (x^3,x^4,…). Rozwiązywane są zwykle poprzez:
   - Rozkład na czynniki.
   - Schemat Hornera.
   - Metody numeryczne.
4. **Równania wymierne**
   Wyrażenia z ułamkami, np. $\frac{x+1}{x-2} = 3$ Należy:
   - Usunąć mianownik (przy uwzględnieniu dziedziny).
   - Rozwiązać powstałe równanie.
5. **Równania z pierwiastkami**
   Np. $\sqrt{x+3} = 2$ Wymagają:
   - Podniesienia obu stron do odpowiedniej potęgi.
   - Sprawdzenia, czy wynik należy do dziedziny.
6. **Równania wykładnicze i logarytmiczne**
   - Wykładnicze: $2^x=8 \implies x=3$
   - Logarytmiczne: $\log_2(x)=3 \implies x=8$

Logarytmy to funkcje matematyczne, które odpowiadają na pytanie: "do jakiej potęgi trzeba podnieść podstawę logarytmu, aby uzyskać daną liczbę

1. Definicja logarytmu:

Logarytm o podstawie a z liczby x (gdzie a>0 i a≠1) to liczba y, która spełnia równość:

$a^y=x$

Zapisujemy to jako:

$\log_a(x)=y$

To oznacza, że logarytm z x przy podstawie a to wykładnik, do którego trzeba podnieść a, aby otrzymać x.

2. Podstawowe własności logarytmów:

- **Własność identycznościowa**: $\log_a(a)=1$(dla każdej podstawy a>0)
- **Logarytm z 1**: $\log_a(1)=0$ (dla każdej podstawy a>0)
- **Logarytm z liczby mniejszej od podstawy**: Jeśli 0<x<1

3. Działania na logarytmach:

3.1. Dodawanie logarytmów (wzór o dodawaniu logarytmów):

Jeśli mamy logarytmy o tej samej podstawie, to ich suma jest równa logarytmowi iloczynu:

$\log_a(x)+\log_a(y)=\log_a(xy)$

Przykład:

$\log_2(4)+\log_2(8)=\log_2(4\cdot 8)=\log_2(32)=5$

3.2. Odejmowanie logarytmów (wzór o odejmowaniu logarytmów):

Różnica dwóch logarytmów o tej samej podstawie jest logarytmem ilorazu:

$\log_a(x)-\log_a(y)=\log_a(x/y)$

Przykład:

$\log_2(8)-\log_2(2)=\log2(8/2)=\log_2(4)=2$

3.3. Mnożenie logarytmu przez stałą (wzór o mnożeniu):

Jeśli logarytm jest mnożony przez liczbę k, to można to zapisać jako logarytm z potęgi:

$k\cdot\log_a(x)=\log_a(x^k)$

Przykład:

$3\cdot\log_2(4)=\log2(4^3)=\log_2(64)=6$

3.4. Zmiana podstawy logarytmu (wzór o zmianie podstawy):

Jeżeli mamy logarytm o podstawiea, to możemy go przekształcić na logarytm o innej podstawie b przy pomocy wzoru:

$\log_a(x)=\log_b(x)/\log_b(a)$

W praktyce najczęściej stosuje się logarytmy o podstawie 10 (logarytmy dziesiętne) i o podstawie e (logarytmy naturalne). Przykład:

$\log_2(8)=\log_{10}(8)/\log_{10}(2)= 0.9031 / 0.3010 \approx 3$

3.5. Logarytm z potęgi:

Logarytm z liczby podniesionej do potęgi jest równy wykładnikowi tej liczby razy logarytm z podstawy:

$\log_a(x^k)=k\cdot\log_a(x)$

Przykład:

$\log_2(8^3)=3\cdot\log_2(8)=3\cdot 3=9$

3.6. Logarytm z iloczynu:

Wzór o logarytmie iloczynu:

$\log_a(x*y)=\log_a(x)+\log_a(y)$

jest odwrotnością wzoru o dodawaniu logarytmów. Oznacza to, że iloczyn liczb w logarytmie może zostać rozdzielony na sumę dwóch logarytmów.

3.7. Logarytm z ilorazu:

Wzór o logarytmie ilorazu:

$\log_a(x/y)=\log_a(x)-\log_a(y)$

jest odwrotnością wzoru o odejmowaniu logarytmów. Oznacza to, że iloraz liczb w logarytmie może zostać zapisany jako różnica dwóch logarytmów.

3.8. Logarytm z liczby mniejszej od 1:

Jeśli $0<x<1$, to $\log_a(x)$ jest liczbą ujemną, o ile $a>1$. Jest to wynik, który można wykorzystać przy obliczaniach związanych z logarytmami.

4. Zastosowania logarytmów:

- **Rozwiązywanie równań wykładniczych**: Logarytmy są bardzo przydatne przy rozwiązywaniu równań wykładniczych, gdzie występują zmienne w wykładniku.
- **Skala logarytmiczna**: W fizyce i naukach technicznych skale logarytmiczne (np. skala decybelowa, pH) są stosowane do wyrażania danych w formie logarytmicznej.

Etapy rozwiązywania równań:

1. Ustalenie dziedziny wyrażenia.
2. Przekształcenie równania w celu izolacji zmiennej.
3. Wykorzystanie odpowiednich metod w zależności od rodzaju równania.
4. Sprawdzenie wyników pod kątem dziedziny i podstawienie do pierwotnego równania.

2b. Rozwiązywanie nierówności

Nierówność to wyrażenie matematyczne z jednym z poniższych znaków:
$>,<,\geq,\leq$. Nierówności opisują zakres wartości zmiennych, które spełniają dane warunki.

Główne rodzaje nierówności:

1. **Nierówności liniowe**
   Mają postać $ax+b>0$
   Rozwiązanie polega na izolacji zmiennej:
   $x>-b/a$, gdzie należy zwrócić uwagę na zmianę znaku nierówności przy mnożeniu/dzieleniu przez liczbę ujemną.
2. **Nierówności kwadratowe**
   Mają postać $ax^2+bx+c>0$
   Rozwiązanie:
   - Znalezienie miejsc zerowych ($\Delta=b2-4ac$)
     Wyznaczenie przedziałów, w których wyrażenie jest dodatnie/ujemne, na podstawie przebiegu paraboli.
   - Metoda równań pomocniczych: ax2+bx+c=0
3. **Nierówności wymierne**
   Mają postać $f(x)/g(x)>0$
   Rozwiązanie:
   - Znalezienie miejsc zerowych licznika i mianownika.
   - Ustalenie znaków wyrażenia w przedziałach wyznaczonych przez te punkty.
4. **Nierówności z pierwiastkami**
   Np. $\sqrt{x+3} > 2$
   - Wyznaczenie dziedziny: x+3≥0
   - Podniesienie do potęgi i rozwiązanie.
5. **Nierówności wykładnicze i logarytmiczne**
   - Wykładnicze: $2^x>8 \implies x>3$
   - Logarytmiczne: $\log_2(x)<3 \implies x<8$

Etapy rozwiązywania nierówności:

1. Ustalenie dziedziny.
2. Przekształcenie nierówności, by uprościć jej postać.
3. Wyznaczenie wartości, dla których wyrażenie zmienia znak (miejsca zerowe).
4. Analiza znaków w przedziałach i wybór tych, które spełniają warunek.
5. Zapis rozwiązania w postaci przedziałów.

Przykład równań i nierówności:

1. **Równanie kwadratowe:**
   $x^2-5x+6=0$
   $(x-2)(x-3)=0 \implies x=2$ lub $x=3$.
2. **Nierówność kwadratowa:**
   $x^2-5x+6>$

   Rozwiązanie:

   - Miejsca zerowe: x=2,x=
   - Parabola otwarta w górę (a>0).
   - Rozwiązanie: $x\in(-\infty,2)\cup(3,\infty)$

3. **Nierówność wymierna:**
   $\frac{x+1}{x-2} \leq 0$
   Rozwiązanie:
   - Miejsca zerowe: x=−1, x=2 (miejsce wykluczone).
   - Rozwiązanie: x∈[−1,2)

3 Podstawowe własności funkcji

1. Pojęcia podstawowe

1. **Definicja funkcji**
   Funkcja to przyporządkowanie, które każdemu elementowi z jednego zbioru (dziedziny Df) przypisuje dokładnie jeden element z drugiego zbioru (przeciwdziedziny Zf).
   Notacja: f:Df→Zf, y=f(x)
2. **Dziedzina funkcji (Df)**
   Zbiór wszystkich wartości x, dla których funkcja jest określona.
3. **Zbiór wartości (Wf)**
   Zbiór wszystkich y, które funkcja przyjmuje, tj. y=f(x)

2. Podstawowe własności funkcji

1. **Monotoniczność**
   Funkcja jest monotoniczna, jeśli jej wartości zmieniają się w sposób uporządkowany na danym przedziale:
   - **Rosnąca:** Funkcja f jest rosnąca jeżeli dla dowolnych dwóch argumentów x1 oraz x2 takich, że x1<x2, zachodzi warunek f(x1)<f(x2).(idzie w góre)
   - **Malejąca:** Funkcja f jest malejąca jeżeli dla dowolnych dwóch argumentów x1 oraz x2 takich, że x1<x2, zachodzi warunek f(x1)>f(x2).(idzie w dół)
   - **Stała** Funkcja f jest stała, jeżeli dla każdego argumentu z dziedziny przyjmuje taką samą wartość.(Stoi nie opada)
   - **Nierosnąca:** Funkcja f jest nierosnącą jeżeli dla dowolnych dwóch argumentów x1 oraz x2 takich, że x1<x2, zachodzi warunek f(x1)≥f(x2). Czyli funkcja jest nierosnącą kiedy jest malejąca lub stał. a Idzie w pizdu w dół ale czasem stanie
   - **Niemalejąca:** Funkcja f jest niemalejąca jeżeli dla dowolnych dwóch argumentów x1 oraz x2 takich, że x1<x2, zachodzi warunek f(x1)≤f(x2). Czyli funkcja jest niemalejąca kiedy jest rosnąca lub stała. Idzie w pizdu w górę ale czasem stanie
2. **Ograniczoność**
   Funkcja jest ograniczona, jeśli jej wartości nie przekraczają określonej granicy:
   - **Ograniczona z góry:** Istnieje liczba M, taka że f(x)≤M dla wszystkich x ∈ Df .
   - **Ograniczona z dołu:** Istnieje liczba mmm, taka że f(x)≥m dla wszystkich x ∈ Df
   - **Ograniczona:** Funkcja jest ograniczona zarówno z góry, jak i z dołu. m<= f(x) <= M
3. **Parzystość i nieparzystość**

- **Funkcja parzysta:** $f(-x)=f(x)$ dla każdego $x \in Df$ Wykres jest symetryczny względem osi OY.
  Przykład: $f(x)=x^2$
- **Funkcja nieparzysta:** $f(-x)=-f(x)$ dla każdego $x \in Df$ Wykres jest symetryczny względem początku układu współrzędnych.
  Przykład: $f(x)=x^3$

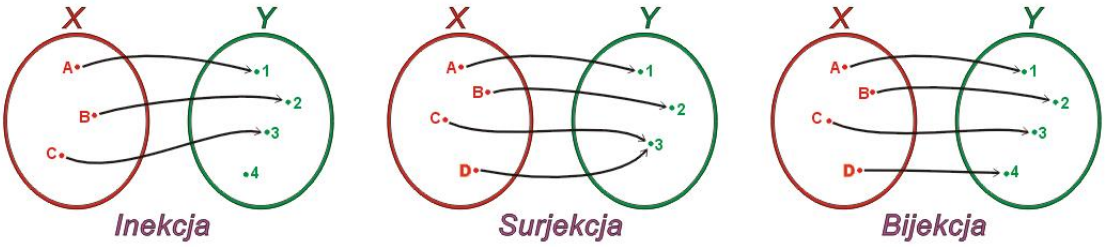
4. **Zerowanie się funkcji**
   Punkt $x_0 \in Df$ jest miejscem zerowym funkcji, jeśli $f(x_0)=0$
   Przykład: Funkcja $f(x)=x-2$ ma miejsce zerowe $x_0=2$.
5. **Okresowość**
   Funkcja f(x) jest okresowa, jeśli istnieje liczba $T>0$ taka, że:
   $f(x+T)=f(x)$ dla każdego $x \in Df$
   Przykład: Funkcja sin(x) jest okresowa z okresem $T=2\pi$
6. **Granice funkcji**
   Funkcja może dążyć do określonej wartości L w punkcie x_0 (lub na krańcu przedziału):
   - $\lim_{x \to x_0} f(x)=L$
     Przykład: Funkcja $f(x)=1/x$ ma granicę $\lim_{x \to \infty} f(x)=0$
7. **Ciągłość**
   Funkcja f(x) jest ciągła w punkcie x_0, jeśli:
   $\lim_{x \to x_0} f(x)=f(x_0)$
   Przykład: Funkcja $f(x)=x^2$ jest ciągła na R

   w dużym skrócie nie jest przerywana w żadnym momencie funkcji lub przedziału który jest nam ważny

8. **Asymptoty**
   Asymptoty opisują zachowanie funkcji w nieskończoności(linia której nigdy nie dotknie funkcja ale będzie jej bardzo blisko):
   - Asymptota pozioma: Funkcja dąży do stałej wartości (y = c).
   - Asymptota pionowa: Funkcja zbliża się do nieskończoności w punkcie x=a
   - Asymptota ukośna: Funkcja dąży do prostej ukośnej.

3. Inne ważne cechy funkcji

1. **Iniekcja, surjekcja, bijekcja**
   - **Iniekcja:** Funkcja przyporządkowuje różne wartości różnym argumentom $f(x1) \neq f(x2)$ dla $x1 \neq x2$
   - **Surjekcja:** Każdy element przeciwdziedziny jest obrazem co najmniej jednego argumentu.
   - **Bijekcja:** Funkcja jest jednocześnie iniekcją i surjekcją.



2. **Funkcja odwracalna**
   Funkcja f(x) jest odwracalna, jeśli istnieje funkcja $f^{-1}(x)$, taka że $f(f^{-1}(x))=x$
   Przykład: Funkcja f(x)=x+2 ma odwrotność $f^{-1}(x)=x-2$
3. **Maksima i minima funkcji**
   - **Lokalne maksimum:** Wartość $f(x_0)$ jest większa niż wartości funkcji w sąsiedztwie x_0.
   - **Lokalne minimum:** Wartość $f(x_0)$ jest mniejsza niż wartości funkcji w sąsiedztwie x_0.
   - **Globalne maksimum/minimum:** Ekstremum w całej dziedzinie.

4 Ciągi liczbowe (ciąg arytmetyczne i ciąg geometryczny, ciągi rekurencyjne)

Ciągi liczbowe

Ciąg liczbowy to funkcja, która każdej liczbie naturalnej n przypisuje liczbę rzeczywistą a_n. W praktyce oznacza to uporządkowany zbiór liczb, gdzie n to numer wyrazu w ciągu.

Ciągi mogą być opisane różnymi wzorami, zarówno w sposób jawny, jak i rekurencyjny.

1. Ciąg arytmetyczny

Definicja

Ciąg arytmetyczny to ciąg liczb, w którym różnica między dowolnymi dwoma kolejnymi wyrazami jest stała i nazywana **różnicą ciągu** (r).
Wzór ogólny:

$$a_n=a_1+(n-1)r$$

Gdzie:

- a_n – n-ty wyraz ciągu,
- a_1 – pierwszy wyraz ciągu,
- r – różnica ciągu.

Własności

1. Suma pierwszych n wyrazów:

   $$S_n=(n/2)\cdot(a_1+a_n)$$

lub

$S_n=(n/2)\cdot(2a_1+(n-1)r)$

Każdy wyraz (poza pierwszym i ostatnim) jest średnią arytmetyczną swoich sąsiednich wyrazów:

$$a_k = \frac{A_{k-1} + A_{k+1}}{2}$$

2. Ciąg geometryczny

Definicja

Ciąg geometryczny to ciąg liczb, w którym iloraz dowolnych dwóch kolejnych wyrazów jest stały i nazywany **ilorazem ciągu** (q).
Wzór ogólny:

$a_n=a_1\cdot q^{n-1}$

Gdzie:

- A_n – n-ty wyraz ciągu,
- a_1 – pierwszy wyraz ciągu,
- q – iloraz ciągu (stały współczynnik mnożenia).

Własności

1. Suma pierwszych n wyrazów (dla q≠1):

$$s_n = a_1 \cdot \frac{1 - q^n}{1 - q}$$

2. Suma nieskończonego ciągu geometrycznego (dla |q|<1|):

$$s = \frac{a_1}{1 - q}$$

3. Każdy wyraz (poza pierwszym i ostatnim) jest średnią geometryczną swoich sąsiednich wyrazów:

$$a_k = \sqrt{a_{k-1} * a_{k+1}}$$

3. Ciągi rekurencyjne

Definicja

Ciąg rekurencyjny jest określony za pomocą wzoru rekurencyjnego, który definiuje każdy wyraz ciągu na podstawie wcześniejszych wyrazów.

1. **Prosty ciąg rekurencyjny:**
   Wzór określa an+1

   $a_{n+1} = a_n + r$ (dla ciągu arytmetycznego)

   $a_{n+1} = a_n * q$ (dla ciągu geometrycznego).

   Złożony ciąg rekurencyjny:
   Wyraz a_n zależy od kilku wcześniejszych wyrazów.
   Przykład (ciąg Fibonacciego):

   $a_{n+2} = a_{n+1} + a_n$ (dla $n \geq 1$)

   gdzie a1=1 i a2=1.

Własności ciągów rekurencyjnych

- Ciągi rekurencyjne często wymagają określenia **warunków początkowych** (a_1, a_2, ...), aby jednoznacznie zdefiniować wszystkie wyrazy.
- Rozwiązanie ciągu rekurencyjnego może wymagać znalezienia wzoru jawnego.

5 Algebra wektorów (iloczyn skalarny, wektorowy, działania na wektorach)

Algebra wektorów

1. Operacje na wektorach

1. **Dodawanie wektorów**
   Dwa wektory można dodać, sumując ich odpowiadające współrzędne:

   $\vec{a} + \vec{b} = (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n)$

2. **Odejmowanie wektorów**
   Odjęcie wektora polega na odjęciu odpowiadających współrzędnych:

   $\vec{a} - \vec{b} = (a_1 - b_1, a_2 - b_2, \ldots, a_n - b_n)$

3. **Mnożenie wektora przez liczbę**
   Każdą współrzędną wektora mnoży się przez skalarną liczbę k:

$k \cdot \vec{a} = (k \cdot a_1, k \cdot a_2, \ldots, k \cdot a_n)$

4. **Długość wektora**
   Długość (norma) wektora $\vec{a} = (a_1, a_2, \ldots, a_n)$ wyrażona jest wzorem:

   $|\vec{a}| = \sqrt{(a_1)^2 + (a_2)^2 + \cdots + (a_n)^2}$

5. **Jednostkowy wektor**
   Wektor jednostkowy ma długość 1. Aby znaleźć wektor jednostkowy zgodny z kierunkiem $\vec{a}$, należy podzielić wektor przez jego długość:

$$\vec{u} = \frac{\vec{a}}{|\vec{a}|}.$$

2. Iloczyn skalarny

Iloczyn skalarny dwóch wektorów $\vec{a} = (a_1, a_2, \ldots, a_n)$ to liczba wyrażona wzorem:

$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$

Własności iloczynu skalarnego

1. Wynik iloczynu skalarnego jest liczbą.
2. Wektory są prostopadłe, jeśli $\vec{a} * \vec{b} = 0$.
3. Iloczyn skalarny można wyrazić w zależności od długości wektorów i kąta θ theta między nimi: $\vec{a} * b = |\vec{a}| * |\vec{b}| * \cos\theta$

3. Iloczyn wektorowy

Iloczyn wektorowy dwóch wektorów $\vec{a} = (a1, a2, a3)$ i $\vec{b} = (b1, b2, b3)$ definiuje nowy wektor $\vec{c}$, który jest prostopadły do $\vec{a}$ $\vec{b}$ Obliczany jest jako:

$$\vec{a} \times \vec{b} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix},$$

gdzie $\vec{i}, \vec{j}, \vec{k}$ to wektory jednostkowe w kierunku osi x,y,z
Wynikiem jest:

$\vec{a} \times \vec{b} = (a_2b_3 - a_3b_2)\vec{i} - (a_1b_3 - a_3b_1)\vec{j} + (a_1b_2 - a_2b_1)\vec{k}$

i = [1 0 0]

j = [0 1 0]

k = [0 0 1]

[i, j, k] = [1^2, 1^2, 1^2] = sqrt(1+1+1)

Dla dwóch wektorów a=(a1,a2,a3)) i b=(b1,b2,b3) w przestrzeni trójwymiarowej, ich iloczyn wektorowy a×b jest wektorem, który jest prostopadły do obu wektorów a i b. Iloczyn wektorowy jest zdefiniowany wzorem:

$$a \times b = (a_2b_3 - a_3b_2,\ a_3b_1 - a_1b_3,\ a_1b_2 - a_2b_1)$$

Oznacza to, że wynikowy wektor ma następujące współrzędne:

- Pierwsza współrzędna: $a_2b_3 - a_3b_2$
- Druga współrzędna: $a_3b_1 - a_1b_3$
- Trzecia współrzędna: $a_1b_2 - a_2b_1$

Własności iloczynu wektorowego

Prostopadłość: Iloczyn wektorowy a×b jest prostopadły do obu wektorów a i b.

Wielkość iloczynu wektorowego: Długość wektora iloczynu wektorowego daje pole równoległoboku, którego boki są wektorami a i b:

$|a \times b| = |a| \cdot |b| \cdot \sin\theta$

gdzie θ to kąt między wektorami a i b.

Nieprzemienność: Iloczyn wektorowy nie jest przemienny. Zmiana kolejności wektorów prowadzi do zmiany kierunku wyniku:

$$a \times b = -(b \times a)$$

Rozdzielność względem dodawania: Iloczyn wektorowy jest rozdzielny względem dodawania:

$$a \times (b+c) = a \times b + a \times c$$

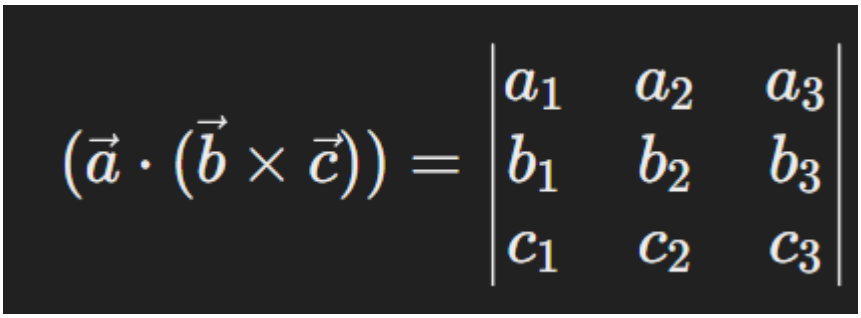
Właściwość łączności z mnożeniem przez skalar: Iloczyn wektorowy jest liniowy w odniesieniu do mnożenia przez skalar:

(k*a)×b=k*(a×b)

gdzie k to dowolny skalar.

4. Mieszane działania na wektorach

1. **Iloczyn mieszany**
   Iloczyn mieszany trzech wektorów $\vec{a},\vec{b},\vec{c}$ jest liczbą:



2. **Własności iloczynu mieszanego**
   - Wynik iloczynu mieszanego reprezentuje objętość równoległościanu, którego krawędziami są $\vec{a},\vec{b},\vec{c}$. Jeśli wynik wynosi 0, wektory leżą w jednej płaszczyźnie.

Przykład obliczenia iloczynu wektorowego

Załóżmy, że mamy dwa wektory:

a=(1,2,3), b=(4,5,6)

Obliczamy ich iloczyn wektorowy:

a × b=(2·6−3·5,3·4−1·6,1·5−2·4)
a × b=(12−15,12−6,5−8)
a × b=(−3,6,−3)

Wynikowy wektor a × b=(−3,6,−3) jest prostopadły do wektorów a i b.

Zastosowanie iloczynu wektorowego

Iloczyn wektorowy znajduje szerokie zastosowanie w wielu dziedzinach matematyki i fizyki, m.in.:

- **Obliczanie momentu siły**: W fizyce iloczyn wektorowy jest używany do obliczania momentu siły względem punktu lub osi.
- **Wektory normalne**: W grafice komputerowej iloczyn wektorowy jest używany do obliczania wektorów normalnych do powierzchni, co jest istotne w oświetleniu 3D.

- **Pole powierzchni**: Iloczyn wektorowy jest używany do obliczania pola powierzchni trójkąta lub równoległoboku.

Podstawy programowania

1 Typy danych

1. Typy całkowitoliczbowe
    - int: liczby całkowite (domyślnie 4 bajty).
    - short int (short): liczby całkowite, zwykle 2 bajty.
    - long int (long): większy zakres liczb całkowitych, zwykle 4-8 bajtów.
    - unsigned: liczby całkowite bez znaku (tylko dodatnie).
    - Bool
2. Typy zmiennoprzecinkowe
    - float: liczby zmiennoprzecinkowe (4 bajty).
    - double: liczby zmiennoprzecinkowe o podwójnej precyzji (8 bajtów).
    - long double: jeszcze większa precyzja (zwykle 16 bajtów).
3. Typ char
    - Przechowuje pojedynczy znak
4. Typ void
    - Używany dla funkcji, które nie zwracają wartości.
5. Typy złożone

Struct

struct Person { char name[50]; int age; };,

union

union Data { int i; float f; char str[20]; };,

enum

enum Day { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday };
enum Day today = Wednesday;

2 Zmienne, operatory, wyrażenia, instrukcje

1. Zmienne są miejscami w pamięci do przechowywania danych. Muszą być zadeklarowane przed użyciem:

```
int x = 10; // deklaracja i inicjalizacja
float y;    // deklaracja
```

2. Operatory
    - Arytmetyczne: +, -, *, /, %.

- Porównania: ==, !=, <, >, <=, >=.
- Logiczne: &&, ||, !.
- Bitowe: &, |, ^, ~, <<, >>.
- Przypisania: =, +=, -=, *=, /=, %=.

3. Wyrażenia
   Kombinacje zmiennych, stałych, operatorów i nawiasów. Przykład:

   ```
   int result = (a + b) * c;
   ```

4. Instrukcje
   Podstawowe jednostki programu, które są wykonywane w kolejności.
   - Instrukcja przypisania: x = x + 1;

1. Lewostronne przypisania (left-assignment)

Lewostronne przypisanie odbywa się, gdy zmienna znajduje się po lewej stronie operatora przypisania (=). W tej formie przypisujemy wartość do zmiennej.

Przykład:

```
int a = 5;  // 'a' jest po lewej stronie przypisania, otrzymuje wartość 5
float x = 3.14f;  // 'x' jest po lewej stronie przypisania, otrzymuje wartość 3.14
```

- **Przypisanie wartości do zmiennej**:

  ```
  int y;
  y = 10;  // Zmienna 'y' otrzymuje wartość 10
  ```

- **Przypisanie do wskaźnika** (wskaźnik jest również po lewej stronie przypisania):

  ```
  int* ptr;
  ptr = &y;  // Wskaźnik 'ptr' otrzymuje adres zmiennej 'y'
  ```

2. Prawostronne przypisania (right-assignment)

Prawostronne przypisanie odnosi się do sytuacji, gdy wyrażenie po prawej stronie operatora przypisania (często używane w kontekście obliczeń czy operacji) ma być obliczone przed przypisaniem do zmiennej po lewej stronie.

Przykład:

```
int a;
a = (b + c) * 2;  // Obliczamy (b + c) * 2, a następnie wynik przypisujemy do 'a'
```

Wartości po prawej stronie przypisania mogą zawierać wyrażenia, które muszą być obliczone przed przypisaniem.

- **Przypisanie zmiennej wartości obliczonej**:

```
a = b + c;  // 'a' otrzymuje wartość sumy 'b' i 'c'
```

3. Przypisania pojedyncze i podwójne

- **Pojedyncze przypisanie**: To standardowy operator przypisania, który przypisuje wartość do zmiennej. Używa się go w większości przypadków.

  Przykład:

  ```
  int x = 5;  // Pojedyncze przypisanie, zmienna 'x' otrzymuje wartość 5
  ```

- **Podwójne przypisanie**: W języku C **podwójne przypisanie** nie istnieje jako osobny operator. Możemy jednak spotkać sytuacje, gdzie przypisanie jest stosowane wielokrotnie w jednym wyrażeniu. W takim przypadku operator przypisania jest używany kilka razy w jednym wyrażeniu.

  Przykład:

  ```
  int x, y;
  x = y = 10;  // Zmienna 'z' otrzymuje wartość 10, potem 'y' otrzymuje 10, a na końcu 'x' również otrzymuje 10
  ```

W tym przykładzie wartość 10 jest przypisywana do z, a potem, przez przeniesienie wartości, również do y i x.

4. Przypisania potrójne

Potrójne przypisanie jest sytuacją, w której przypisanie wykonywane jest w jednym wyrażeniu w taki sposób, że wartości są przekazywane w łańcuchu (np. przypisanie wartości jednej zmiennej do kilku innych w jednym wierszu). Przykład:

```
int a, b, c;
a = b = c = 10;  // Zmienna 'c' otrzymuje 10, potem 'b' otrzymuje 10, a na końcu 'a' również otrzymuje 10
```

Podobnie jak w przypadku podwójnego przypisania, operator przypisania jest używany wielokrotnie w jednym wyrażeniu.

Potrójne przypisanie oznacza, że wynik przypisania wartości do zmiennej c jest następnie wykorzystywany do przypisania wartości do zmiennej b, a na końcu wynik ten przypisywany jest do zmiennej a.

 Podwójne przypisanie: Przypisanie wartości do dwóch zmiennych w jednym wyrażeniu.

 Potrójne przypisanie: Przypisanie wartości do trzech zmiennych w jednym wyrażeniu.

Przykłady w kodzie:

```
#include <stdio.h>

int main() {
   // Lewostronne przypisanie
   int a;
   a = 5;  // Zmienna 'a' otrzymuje wartość 5

   // Prawostronne przypisanie
   int b = 3, c = 4;
   a = b + c;  // 'a' otrzymuje wynik dodawania 'b' i 'c', czyli 7

   // Pojedyncze przypisanie
   int x = 10;  // Przypisanie wartości 10 do zmiennej 'x'

   // Podwójne przypisanie
   int y, z;
   y = z = 20;  // Zmienna 'z' otrzymuje 20, a następnie 'y' również otrzymuje 20

   // Potrójne przypisanie
   int p, q, r;
   p = q = r = 30;  // Zmienna 'r' otrzymuje 30, potem 'q' również, a na końcu 'p' otrzymuje 30

 return 0;
}
```

L-operandowe (lewostronne) operatory

L-operandowe operatory są to operatory, które stosowane są do lewego operandu lub wymagają operandu po lewej stronie.

Operator przypisania (=): Operator przypisania przypisuje wartość zmiennej znajdującej się po prawej stronie operatora do zmiennej po lewej stronie. Stąd, lewy operand to zmienna, do której przypisywana jest wartość.

Przykład:

```
int a, b;
a = b;  // 'a' jest L-operatorem
```

Operatory inkrementacji i dekrementacji:

- **Preinkrementacja (++x)**
- **Predekrementacja (--x)**

Stosują się do zmiennej po lewej stronie, tzn. zmieniają wartość zmiennej bezpośrednio.

Przykład:

```
int x = 5;
++x;  // Preinkrementacja (L-operandowy)
--x;  // Predekrementacja (L-operandowy)
```

Operatory bitowe:

- **Bitowa zmiana (&=, |=, ^=)**
- **Bitowe przesunięcie (<<=, >>=)**

Operator przyjmuje operand po lewej stronie, do którego przypisuje wynik operacji.

Przykład:

```
int a = 5;
a &= 3;  // Bitowa operacja AND z przypisaniem (L-operandowy)
```

R-operandowe (prawostronne) operatory

R-operandowe operatory to te, które wymagają operandów po prawej stronie operatora, czyli operują na operandach po prawej stronie.:

1. **Operator przypisania (=)**: Jak wspomniano wcześniej, operator przypisania stosuje się do operandu po lewej stronie, ale przypisuje wartość z prawego operandu. Stąd jest uważany za operator z **prawooperandowym** zastosowaniem w kontekście przypisania.
2. **Operatory arytmetyczne** (np. dodawanie, mnożenie): Operatory takie jak +, -, *, / wymagają operandów po prawej stronie, na których wykonują operację.

   Przykład:

   ```
   int a = 5, b = 3;
   a = a + b;  // Operator '+' jest prawostronny
   ```

3. **Operatory porównania**:
   - **Większy niż (>)**
   - **Mniejszy niż (<)**
   - **Równy (==)**
   - **Różne (!=)**

   Te operatory również stosują się do operandów po prawej stronie.

   Przykład:

   ```
   if (a < b) {  // Operator '<' jest prawostronny
      printf("a jest mniejsze od b\n");
   }
   ```

4. **Operatory logiczne**:

- **AND logiczne (&&)**
- **OR logiczne (||)**

Te operatory również wymagają operandów po prawej stronie.

Przykład:

```
if (a < b && b > 0) {  // Operator '&&' jest prawostronny
   printf("Warunek jest spełniony\n");
}
```

5. **Operatory bitowe**:
   - **Bitowa operacja AND (&), OR (|), XOR (^)**
   - **Bitowe przesunięcie (<<, >>)**

   Te operatory są używane z operandami po obu stronach, ale operują na operandzie po prawej stronie.

   Przykład:

```
int a = 5, b = 3;
int c = a & b;  // Operator '&' jest prawostronny
```

Przykłady operacji w języku C z L- i R-operandami:

```
#include <stdio.h>

int main() {
   int a = 5, b = 3;

   // L-operandowe operatory
   a += b;  // Operator '+=' jest L-operandowy (przypisanie po lewej stronie)
   printf("a += b: %d\n", a);

   // R-operandowe operatory
   a = b * 2;  // Operator '*' jest R-operandowy (mnożenie po prawej stronie)
   printf("a = b * 2: %d\n", a);

   // Preinkrementacja - L-operandowy
   ++a;  // Inkrementacja zmienia 'a' przed użyciem
   printf("++a: %d\n", a);

   // Predekrementacja - L-operandowy
   --a;  // Dekrementacja zmienia 'a' przed użyciem
   printf("--a: %d\n", a);

   // Porównanie (operator porównania jest R-operandowy)
   if (a == 5) {
```

```
    printf("a jest równe 5\n");
  }

  return 0;
}
```

3 Instrukcja złożona, instrukcje sterujące

1. **Instrukcja złożona (blok)**
   Składa się z wielu instrukcji zamkniętych w klamry {}.

   ```
   {
     int x = 0;
     printf("Hello World\n");
   }
   ```

2. **Instrukcje sterujące**
   - **Instrukcja warunkowa if**:

     ```
     if (x > 0) {
       printf("Liczba jest dodatnia\n");
     } else {
       printf("Liczba nie jest dodatnia\n");
     }
     ```

   - **Pętla for**:

     ```
     for (int i = 0; i < 10; i++) {
       printf("%d\n", i);
     }
     ```

   - **Pętla while**:

     ```
     while (x < 10) {
       x++;
     }
     ```

   - **Pętla do-while**:

     ```
     do {
       x++;
     } while (x < 10);
     ```

   - **Instrukcja switch**:

     ```
       switch (x) {
     case 1: printf("Jeden\n"); break;
     ```

```
        case 2: printf("Dwa\n"); break;
        default: printf("Inna liczba\n");
    }
```

4 Funkcje i struktura programu

Definicja funkcji
Funkcje są blokami kodu, które wykonują określone zadania i mogą zwracać wartość:

```
int sum(int a, int b) {
   return a + b;
}
```

Struktura programu w C

- Pliki nagłówkowe: deklaracje (#include).
- Funkcja główna main:

```
int main() {
   printf("Program w C\n");
   return 0;
}
```

5 Tablice jednowymiarowe oraz dwuwymiarowe

Tablice jednowymiarowe
Są sekwencjami elementów tego samego typu:

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

Tablice dwuwymiarowe
Tablice w formie macierzy:

```
int matrix[2][3] = {
   {1, 2, 3},
   {4, 5, 6}
};
```

Dostęp do elementów

- Tablica jednowymiarowa: `arr[0]`.
- Tablica dwuwymiarowa: `matrix[1][2]`.

6 Parametry funkcji typu wskaźnikowego. Tablice i wskaźniki.

Wskaźniki
Wskaźnik to zmienna przechowująca adres pamięci:

```
int x = 10;
int *ptr = &x;
```

Parametry wskaźnikowe
Używane do przekazywania zmiennych przez referencję:

```
void increment(int *x) {
  (*x)++;}
```

Tablice i wskaźniki
Tablica jest wskaźnikiem do swojego pierwszego elementu:

```
int arr[3] = {1, 2, 3};
int *p = arr; // to samo co &arr[0]
```

7 Struktury

1. **Definicja struktury**
   Struktura to złożony typ danych przechowujący różne typy w jednym miejscu:

   ```
   struct Point {
     int x;
     int y;};
   ```

2. **Tworzenie i dostęp do pól**

   ```
   struct Point p1;
   p1.x = 10;
   p1.y = 20;
   ```

3. **Struktury w funkcjach**
   Struktury mogą być przekazywane jako argumenty lub zwracane z funkcji:

   ```
   struct Point createPoint(int x, int y) {
     struct Point p = {x, y};
     return p;}
   ```

   Przykładowe pytania

- 1. Omów wykorzystanie wskaźników w języku C.
- 2. Przedstaw pojęcie funkcji, jej deklarację i wywołanie. Omów sposoby przekazywania parametrów do funkcji w języku C.
- 3. Omów rodzaje pętli w języku C.
- 4. Sposób dołączania bibliotek do programu w języku C. Omów kilka funkcji dostępnych w bibliotekach *stdio.h*, *stdlib.h*, *math.h*.
- 5. Przedstaw związek tablic i wskaźników w języku C.
- 6. Tablice i tablice dwuwymiarowe, sposób przekazywania tablic do funkcji w języku C.
- 7. Omów rodzaje typów (prostych /podstawowych) w języku C.
- 8. Wyrażenia i operatory w języku C.
- 9. Wymień podstawowe paradygmaty programowania i omów dokładniej trzy wybrane paradygmaty programowania.

- 10. Struktury w języku C – definiowanie, przekazywanie do funkcji, tablice struktur.
- 11. Instrukcja switch vs instrukcja if else.

1. Wykorzystanie wskaźników w języku C

Wskaźniki w języku C to zmienne, które przechowują adresy pamięci innych zmiennych. Dzięki wskaźnikom można manipulować danymi w pamięci w sposób bardziej efektywny i elastyczny. Oto podstawowe zastosowania wskaźników:

- **Przekazywanie danych do funkcji przez referencję**: Umożliwia modyfikowanie wartości zmiennych w funkcji bez konieczności zwracania ich.
- **Dynamiczna alokacja pamięci**: Wskaźniki pozwalają dynamicznie rezerwować i zwalniać pamięć przy użyciu funkcji takich jak malloc(), calloc(), realloc(), i free().
- **Tworzenie struktur danych**: Wskaźniki są wykorzystywane do tworzenia struktur danych, takich jak listy połączone, drzewa, itp.
- **Tablice i wskaźniki**: Wskaźniki są wykorzystywane do pracy z tablicami, ponieważ nazwa tablicy w C to wskaźnik na jej pierwszy element.

Przykład użycia wskaźników:

```
#include <stdio.h>
void zmien_wartosc(int *x) {
   *x = 20;  // Zmienia wartość zmiennej, na którą wskazuje wskaźnik
}

int main() {
   int a = 10;
   zmien_wartosc(&a);  // Przekazanie adresu zmiennej
   printf("Nowa wartosc a: %d\n", a);  // Drukuje 20
   return 0;}
```

2. Pojęcie funkcji, jej deklaracja i wywołanie. Przekazywanie parametrów

Funkcja to blok kodu, który wykonuje określoną operację i może przyjmować dane wejściowe (parametry) oraz zwracać wynik. W języku C funkcje są deklarowane, definiowane i wywoływane w następujący sposób:

- **Deklaracja funkcji** (prototyp):

```
int dodaj(int, int);  // Deklaracja funkcji, która przyjmuje dwa int i zwraca int
```

- **Definicja funkcji**:

```
int dodaj(int a, int b) {
   return a + b;
}
```

- **Wywołanie funkcji**:

```
int wynik = dodaj(3, 4);  // Wywołanie funkcji
```

Przekazywanie parametrów do funkcji:

- **Przez wartość**: Funkcja otrzymuje kopię zmiennej, więc zmiany w funkcji nie wpływają na oryginalną zmienną.
- **Przez wskaźnik**: Funkcja otrzymuje adres zmiennej, dzięki czemu może modyfikować wartość tej zmiennej w pamięci.
- **Przez referencję** (C nie ma bezpośredniego wsparcia dla referencji jak w C++): Można osiągnąć to poprzez wskaźniki.

3. Rodzaje pętli w języku C

W języku C dostępne są trzy podstawowe rodzaje pętli:

1. **Pętla for**:
   - Używana, gdy liczba iteracji jest znana z góry.

```
for (int i = 0; i < 10; i++) {
   printf("%d\n", i);
}
```

2. **Pętla while**:
   - Używana, gdy liczba iteracji nie jest znana, ale pętla powinna się wykonywać dopóki warunek jest prawdziwy.

```
int i = 0;
while (i < 10) {
   printf("%d\n", i);
   i++;
}
```

3. **Pętla do...while**:
   - Podobna do while, ale warunek jest sprawdzany po pierwszej iteracji, więc pętla jest wykonywana przynajmniej raz.

```
int i = 0;
do {
   printf("%d\n", i);
   i++;
} while (i < 10);
```

4. Sposób dołączania bibliotek do programu w języku C

Aby dołączyć bibliotekę w C, używa się dyrektywy #include:

```
#include <stdio.h>  // Standardowa biblioteka wejścia/wyjścia
#include <stdlib.h>  // Biblioteka standardowa
#include <math.h>  // Biblioteka matematyczna
```

Funkcje w standardowych bibliotekach:

- **stdio.h**:
  - printf(), scanf(), fopen(), fclose(), fputs(), fgets()
- **stdlib.h**:
  - malloc(), free(), exit(), atoi(), rand(), srand()
- **math.h**:
  - sqrt(), pow(), sin(), cos(), tan(), log()

5. Związek tablic i wskaźników w języku C

Tablice i wskaźniki są ściśle związane, ponieważ nazwa tablicy w C jest wskaźnikiem do jej pierwszego elementu. Oznacza to, że tablica może być traktowana jak wskaźnik w kontekście dostępu do jej elementów.

Przykład:

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

int *ptr = tablica;  // Wskaźnik na pierwszy element tablicy

printf("%d\n", *(ptr + 2));  // Dostęp do trzeciego elementu tablicy (3)
```

6. Tablice i tablice dwuwymiarowe, sposób przekazywania tablic do funkcji w języku C

Tablice w C są jednowymiarowe lub wielowymiarowe (np. tablica dwuwymiarowa). Tablicę przekazujemy do funkcji, przekazując wskaźnik na pierwszy element tablicy.

Przykład przekazywania tablicy jednowymiarowej:

```
void wyswietl(int tab[], int rozmiar) {
   for (int i = 0; i < rozmiar; i++) {
      printf("%d ", tab[i]);
   }
}

int main() {
   int tablica[] = {1, 2, 3, 4};
   wyswietl(tablica, 4);
   return 0;
}
```

Tablice dwuwymiarowe:

```
void wyswietl(int tab[][3], int wiersze) {
   for (int i = 0; i < wiersze; i++) {
      for (int j = 0; j < 3; j++) {
         printf("%d ", tab[i][j]);
      }
```

```
        printf("\n");
    }
}

int main() {
    int tablica[2][3] = {{1, 2, 3}, {4, 5, 6}};
    wyswietl(tablica, 2);
    return 0;
}
```

7. Rodzaje typów w języku C

W C wyróżnia się różne rodzaje typów danych:

- **Typy proste**:
    - **Podstawowe**: int, char, float, double, void, bool.
    - **Zmienne typu wyliczeniowego**: enum.
    - **Typy wskaźnikowe**: Wskaźniki na różne typy.
- **Typy złożone**:
    - **Tablice**: Kolekcje zmiennych tego samego typu.
    - **Struktury**: Zbiór zmiennych o różnych typach.
    - **Unie**: Typy, które pozwalają na przechowywanie różnych typów danych w tym samym obszarze pamięci.

8. Wyrażenia i operatory w języku C

Wyrażenia to kombinacje zmiennych, stałych, operatorów i funkcji, które obliczają wartości. W C mamy kilka rodzajów operatorów:

- **Arytmetyczne**: +, -, *, /, %
- **Porównania**: ==, !=, <, >, <=, >=
- **Logiczne**: &&, ||, !
- **Bitowe**: &, |, ^, ~, <<, >>
- **Przypisania**: =, +=, -=, *=, /=, %= itd.
- **Inkrementacja i dekrementacja**: ++, --

9. Podstawowe paradygmaty programowania i ich omówienie

1. **Programowanie imperatywne**:
    - Skupia się na wykonywaniu działań na danych, zmieniając stan programu przez instrukcje.
2. **Programowanie obiektowe**:
    - Skupia się na obiektach, które zawierają dane i metody operujące na tych danych. Zasady to enkapsulacja, dziedziczenie i polimorfizm.
3. **Programowanie funkcyjne**:
    - Skupia się na funkcjach, które są traktowane jako obywatel pierwszej klasy. Zamiast modyfikować stan, funkcje zwracają wartości w sposób czysty (bez efektów ubocznych).

10. Struktury w języku C

Struktury to typy danych, które pozwalają grupować różne zmienne w jedną jednostkę.

- **Definicja struktury**:

```
struct Osoba {
   char imie[50];
   int wiek;
};
```

- **Przekazywanie struktur do funkcji**: Struktury można przekazywać przez wartość (kopiowanie) lub przez wskaźnik.

```
void wypisz(struct Osoba o) {
   printf("%s, %d\n", o.imie, o.wiek);
}

void wypisz(struct Osoba *o) {
   printf("%s, %d\n", o->imie, o->wiek);
}
```

- **Tablice struktur**:

```
struct Osoba osoby[3];
osoby[0].wiek = 25;
```

11. Instrukcja switch vs instrukcja if else

- **switch**:
  Wykonuje porównania jednej zmiennej z wieloma stałymi wartościami. Jest bardziej wydajne w przypadku wielu porównań tej samej zmiennej.
- **if...else**:
  Może wykonywać bardziej złożone porównania (np. złożone warunki logiczne), ale jest mniej wydajne w przypadku wielu przypadków do porównania.

Wprowadzenie do systemów operacyjnych

1 Rodzaje i mechanizmy działania systemu operacyjnego

W zależności od sposobu organizacji i celów, systemy operacyjne dzielą się na różne typy, z których najczęściej wymienia się:

Rodzaje systemów operacyjnych:

1. **Systemy operacyjne jednozadaniowe**:
   - W systemach tych użytkownik może wykonywać tylko jedno zadanie w danym momencie. Przykład: systemy wbudowane.
2. **Systemy operacyjne wielozadaniowe**:

- Umożliwiają uruchomienie wielu programów (procesów) jednocześnie. Systemy te dzielą zasoby między różne procesy, zapewniając ich równoczesne działanie (przełączanie kontekstów).
- Przykłady: Windows, Linux, macOS.

3. **Systemy operacyjne czasu rzeczywistego (RTOS)**:
   - Systemy, które zapewniają gwarancje czasowe dla operacji. Czas odpowiedzi musi być precyzyjnie kontrolowany.
   - Przykłady: FreeRTOS, QNX.
4. **Systemy operacyjne wielodostępowe**:
   - Umożliwiają jednoczesny dostęp wielu użytkowników do systemu. Systemy te zarządzają sesjami użytkowników, zapewniając bezpieczeństwo i oddzielność procesów.
   - Przykład: Unix, Linux.

Mechanizmy działania systemu operacyjnego:

1. **Zarządzanie procesami**:
   - Procesy to jednostki wykonawcze programów. System operacyjny odpowiada za ich tworzenie, planowanie, synchronizację i kończenie.
   - Mechanizmy: planowanie procesów, zarządzanie czasem CPU, synchronizacja (mutexy, semafory).
2. **Zarządzanie pamięcią**:
   - System operacyjny zarządza pamięcią operacyjną (RAM), przydzielając ją poszczególnym procesom i zapewniając izolację pamięci.
   - Mechanizmy: segmentacja, stronicowanie, pamięć wirtualna.
3. **Zarządzanie plikami**:
   - System operacyjny organizuje dane na dysku w formie plików i katalogów. Odpowiada za dostęp do plików, ich tworzenie, usuwanie i modyfikowanie.
   - Mechanizmy: systemy plików (FAT, NTFS, ext4).
4. **Zarządzanie urządzeniami wejścia/wyjścia**:
   - SO zarządza komunikacją z urządzeniami peryferyjnymi (drukarki, dyski, klawiatura, monitor).
   - Mechanizmy: sterowniki urządzeń, buforowanie, komunikacja z urządzeniami przez przerwania.

1. Płyta główna (Motherboard)

Płyta główna jest centralnym elementem komputera, który łączy wszystkie inne komponenty. Jest to fizyczna platforma, na której montowane są inne podzespoły, takie jak procesor, pamięć RAM, karta graficzna, karta dźwiękowa i inne.

Główne cechy płyty głównej:

- **Gniazdo procesora (Socket)**: Miejsce, w którym montowany jest procesor (CPU).
- **Złącza pamięci RAM**: Sloty na pamięć operacyjną (RAM).
- **Złącza PCI/PCIe**: Porty do podłączenia kart rozszerzeń (np. karty graficzne, karty sieciowe).
- **Chipset**: Układ sterujący komunikacją między procesorem a innymi podzespołami.
- **Porty wejścia/wyjścia**: USB, audio, HDMI, Ethernet, itp.
- **Złącze dla dysku twardego (HDD/SSD)**: SATA lub NVMe.

2. Procesor (CPU - Central Processing Unit)

Procesor to "mózg" komputera, który wykonuje większość obliczeń i operacji w systemie. Jest to układ scalony, który odpowiada za przetwarzanie danych i sterowanie działaniem innych komponentów. Procesor komunikuje się z pamięcią RAM, dyskami twardymi i innymi urządzeniami.

Główne cechy procesora:

- **Rdzenie (Cores)**: Współczesne procesory mają wiele rdzeni, co pozwala na równoczesne wykonywanie wielu zadań (wielozadaniowość).
- **Wątki (Threads)**: Wspiera równoczesne przetwarzanie danych, szczególnie w procesorach obsługujących technologię Hyper-Threading.
- **Taktowanie (Clock Speed)**: Wyrażane w gigahercach (GHz), mówi o szybkości, z jaką procesor wykonuje operacje.

3. Pamięć RAM (Random Access Memory)

Pamięć RAM to pamięć o dostępie swobodnym, w której komputer przechowuje dane tymczasowe, używane w trakcie pracy. Pamięć RAM jest bardzo szybka, ale jej zawartość jest tracona po wyłączeniu komputera.

Główne cechy pamięci RAM:

- **Pojemność**: Typowo w komputerach osobistych to 4GB, 8GB, 16GB lub więcej.
- **Prędkość**: Określa, jak szybko RAM może odczytywać i zapisywać dane.
- **Typ**: DDR4 i DDR5 to najczęściej spotykane standardy w nowoczesnych komputerach.

4. Dysk twardy (HDD) / Dysk SSD (Solid State Drive)

Dyski twarde są używane do przechowywania danych na stałe, takich jak system operacyjny, aplikacje i pliki użytkownika. Współczesne komputery często używają SSD (dyski półprzewodnikowe) zamiast tradycyjnych HDD, ponieważ SSD są znacznie szybsze.

Główne cechy dysków:

- **Pojemność**: Dyski SSD mogą mieć od 120GB do kilku terabajtów, a HDD mają większą pojemność (do kilku TB), ale są wolniejsze.
- **Prędkość**: SSD oferują szybszy dostęp do danych niż HDD, co wpływa na ogólną wydajność systemu.

5. Karta graficzna (GPU - Graphics Processing Unit)

Karta graficzna jest odpowiedzialna za renderowanie obrazu na monitorze. Przetwarza dane graficzne, takie jak gry, filmy, aplikacje 3D czy programy do edycji wideo. Niektóre komputery mają zintegrowaną kartę graficzną (część procesora), podczas gdy inne używają dedykowanej karty graficznej.

Główne cechy karty graficznej:

- **Procesor graficzny** (GPU): Specjalny procesor odpowiadający za renderowanie obrazów.
- **Pamięć VRAM**: Pamięć dedykowana do przechowywania danych graficznych.
- **Porty wyjścia**: HDMI, DisplayPort, DVI, VGA do podłączenia monitorów.

6. Zasilacz (PSU - Power Supply Unit)

Zasilacz dostarcza energię elektryczną do wszystkich komponentów komputera. Ma różne złącza i napięcia, które pasują do wymagań zasilania różnych elementów komputera.

Główne cechy zasilacza:

- **Moc**: Mierzona w watach (W), zwykle w granicach 300W - 1000W, w zależności od komponentów komputera (np. dedykowana karta graficzna wymaga większej mocy).
- **Sprawność**: Określa, jak efektywnie zasilacz przekształca energię z gniazdka na zasilanie do komponentów.

7. Obudowa (Case)

Obudowa komputera to fizyczna obudowa, która mieści wszystkie komponenty komputera. Obejmuje przestrzeń na płytę główną, zasilacz, dyski, napędy, karty rozszerzeń i inne elementy. Obudowa zapewnia również chłodzenie i wentylację.

Główne cechy obudowy:

- **Rozmiar**: Istnieją obudowy ATX, mATX i mini-ITX, które różnią się wielkością i ilością miejsca na komponenty.
- **Wentylacja**: Obudowa posiada wentylatory, które pomagają chłodzić komponenty, zapobiegając przegrzewaniu się.
- **Porty zewnętrzne**: USB, audio, przyciski włączania i resetu.

8. Napędy optyczne (opcjonalnie)

Chociaż w dzisiejszych czasach napędy optyczne (np. DVD, Blu-ray) stają się coraz mniej popularne, w niektórych komputerach nadal są dostępne.

9. Karty rozszerzeń

Komputer może mieć dodatkowe karty rozszerzeń, które oferują różne funkcje, takie jak:

- **Karta dźwiękowa** – dla zaawansowanej jakości dźwięku.
- **Karta sieciowa** – do łączności z siecią komputerową.
- **Karty PCIe** – np. dodatkowe porty USB, karty sieciowe Wi-Fi, itp.

10. Porty i interfejsy zewnętrzne

Komputer ma różnorodne porty do podłączania zewnętrznych urządzeń, takich jak:

- **USB** (do podłączania myszek, klawiatur, drukarek itp.)
- **HDMI** (do podłączenia monitora lub telewizora)
- **Ethernet** (do podłączenia komputera do sieci lokalnej)
- **Audio** (słuchawki, mikrofon)

11. Chłodzenie

Wysokiej jakości chłodzenie jest kluczowe dla zachowania stabilności systemu i zapobiegania przegrzewaniu się komponentów. Współczesne komputery mogą mieć:

- **Chłodzenie powietrzne**: Wentylatory na procesorze, obudowie lub karcie graficznej.
- **Chłodzenie cieczą**: W bardziej zaawansowanych konfiguracjach, zwłaszcza w przypadku podkręcania procesora (overclocking), może być stosowane chłodzenie cieczą.

2 Zadania poinstalacyjne w systemach operacyjnych

Po zainstalowaniu systemu operacyjnego należy przeprowadzić kilka czynności, które pozwolą na jego prawidłowe skonfigurowanie i przygotowanie do użycia. Do zadań poinstalacyjnych należą:

1. **Aktualizacja systemu**:
    - Instalowanie najnowszych poprawek i aktualizacji zabezpieczeń, aby zapewnić bezpieczeństwo i stabilność systemu.
2. **Instalacja sterowników**:
    - W zależności od sprzętu, konieczne może być zainstalowanie odpowiednich sterowników, które pozwolą systemowi operacyjnemu współpracować z urządzeniami zewnętrznymi (drukarki, karty graficzne, sieciowe itp.).
3. **Tworzenie użytkowników i grup**:
    - Konfiguracja kont użytkowników i grup, nadanie odpowiednich uprawnień dostępu do plików i zasobów systemowych.
4. **Konfiguracja sieci**:
    - Ustawienie połączeń sieciowych, konfiguracja protokołów TCP/IP, adresów IP, bramy domyślnej, DNS.
5. **Instalacja oprogramowania**:
    - Instalacja wymaganych aplikacji, które będą używane na komputerze (np. edytory tekstu, oprogramowanie biurowe, aplikacje internetowe).
6. **Zabezpieczenia systemu**:
    - Ustawienie zapory sieciowej (firewalla), instalacja oprogramowania antywirusowego, włączenie szyfrowania danych.

3 Zarządzanie użytkownikami, uprawnienia

Systemy operacyjne umożliwiają tworzenie kont użytkowników i przypisywanie im odpowiednich uprawnień do zasobów systemowych.

Zarządzanie użytkownikami:

- **Tworzenie użytkowników**: Użytkownicy mogą być dodawani za pomocą komend (np. w systemie Linux: useradd).
- **Usuwanie użytkowników**: Użytkownicy mogą być usuwani, a ich zasoby (takie jak pliki) mogą zostać przeniesione lub usunięte (np. userdel).

Grupy użytkowników:

- Użytkownicy mogą być organizowani w grupy (np. admini, użytkownicy zwykli), co ułatwia zarządzanie uprawnieniami.
- Komenda do dodawania grupy w systemie Linux: groupadd.

Uprawnienia:

- System operacyjny pozwala na przypisanie uprawnień do plików i katalogów. W systemach uniksowych, uprawnienia są reprezentowane przez trzy grupy użytkowników: właściciela pliku, grupę, oraz innych użytkowników.
- **Uprawnienia**:
  - **r** (czytanie),
  - **w** (zapis),
  - **x** (wykonywanie).

  Przykład uprawnień:

  rwxrw-r-- użytkownik grupa plik

- Uprawnienia mogą być zmieniane za pomocą komend chmod (zmiana uprawnień), chown (zmiana właściciela) i chgrp (zmiana grupy).

4 Instalacja i konfiguracja oprogramowania

Instalacja oprogramowania w systemie operacyjnym zależy od jego rodzaju i używanych narzędzi. W systemach operacyjnych występują różne metody instalacji, w tym:

Systemy Linux/Unix:

- **Instalacja oprogramowania za pomocą menedżerów pakietów**:
  - **APT** (Debian/Ubuntu): sudo apt install program
  - **YUM** (Red Hat/CentOS): sudo yum install program
  - **Zypper** (openSUSE): sudo zypper install program
- **Kompilacja ze źródeł**:
  - Wymaga pobrania kodu źródłowego, jego kompilacji i instalacji za pomocą narzędzi takich jak make, gcc.

Systemy Windows:

- **Instalacja za pomocą plików instalacyjnych (.exe, .msi)**:
  - Programy są zazwyczaj instalowane za pomocą interfejsu graficznego (instalator).
- **Instalacja przy użyciu menedżera pakietów**:
  - Na Windowsie można używać menedżerów pakietów takich jak **Chocolatey**: choco install program.

Konfiguracja oprogramowania:

- **Ustawienia systemowe**: Zazwyczaj oprogramowanie umożliwia konfigurację poprzez pliki konfiguracyjne lub interfejsy graficzne, gdzie użytkownik może dostosować opcje oprogramowania (np. ścieżki, porty, uprawnienia).
- **Zarządzanie usługami**: Oprogramowanie może być uruchamiane jako usługi w tle (np. serwery WWW, bazy danych). Konfiguracja usług odbywa się za pomocą odpowiednich plików konfiguracyjnych lub poleceń systemowych (np. systemctl w systemie Linux).

Teoretyczne podstawy informatyki

1. Znaczenie, działanie oraz najczęściej występujące typy kanałów informacyjnych

Kanał informacyjny to medium służące do przesyłania informacji od nadawcy do odbiorcy. W teorii informacji kanały modelują sposób, w jaki dane są transmitowane w obecności zakłóceń.

Znaczenie kanałów informacyjnych:

- Zapewniają środek przekazu informacji w systemach komunikacyjnych.
- Analizują wpływ zakłóceń i strat danych na transmisję informacji.
- Pomagają projektować bardziej niezawodne systemy transmisji danych.

Działanie kanału informacyjnego:

1. **Nadawca** generuje dane (np. sygnały, wiadomości).
2. Dane są **kodowane** w celu poprawy odporności na zakłócenia.
3. Dane są przesyłane przez **kanał transmisyjny**.
4. **Zakłócenia** mogą zniekształcać dane.
5. Odbiorca odkodowuje otrzymane dane, próbując odzyskać pierwotny sygnał.

Typy kanałów informacyjnych:

1. **Kanał bezszumowy**:
   - Przesyła dane bez żadnych zakłóceń.
   - Wartość informacji na wejściu i wyjściu jest identyczna.
2. **Kanał z szumem**:
   - Przesyła dane, ale mogą wystąpić błędy z powodu zakłóceń.
   - Przykład: transmisja radiowa zakłócona przez interferencję.

3. **Kanał dyskretny**:
    - Przesyła dane w postaci dyskretnych sygnałów (np. bity w komputerze).
4. **Kanał ciągły**:
    - Przesyła dane w postaci ciągłych sygnałów (np. fale dźwiękowe, sygnały analogowe).

2. Typy kodów oraz wielkości charakteryzujące kody

Typy kodów:

1. **Kody liniowe**:
    - Oparte na operacjach liniowych, stosowane w korekcji błędów.
    - Przykłady: kody Hamming, kody BCH.

Kod Hamminga
Do czego służy? No do tego:
Kod Hamminga wykrywa i koryguje błędy polegające na przekłamaniu jednego bitu może wykrywać ale nie korygować błędów podwójnych

Zastostosowanie znajduje w:
Przetwarzanie danych
Telekomunikacja
Kompresja danych
Rozwiązywanie zagadek i kodów turbo
Satelity
Plasma CAM
Ekranowane przewody
Modemy
Pamięć komputera
Otwarte złącza
Systemy wbudowane i procesor
Zalety:
Może zapewnić wykrywanie błędów, a także wskazuje bit, który zawiera błąd do korekcji
Kody Hamminga są bardzo łatwe i najlepsze w użyciu w pamięci komputera oraz jednobitowej korekcji i wykrywaniu błędów.
Wady:
Jest najlepszy tylko do jednobitowej korekcji i wykrywania błędów. W przypadku błędów wielu bitów całość może zostać uszkodzona.
Algorytm kodu Hamminga może rozwiązać tylko błędy jednobitowe
Ten koda jest prosty ugólem mamy np. 11010 to zapisujemy to w ten sposób
1_101_0__
98765432**1** następny byłby na miejscu 16,32,64,128,256,512 itp.
Słowa zamieniamy na ascii no wiec tak masz takie kongo to teraz bierzesz miejsce gdzie jest zapisana 1 I zapisujesz jej pozycję binarnie 5 – 0101, 7-0111, 9 – 1001
Co teraz? Gówno pomyśl przez chwile

Dobra chwila minęła słuchaj bałwanie musisz dać XOR jak nie wiesz co to to się dowiedz

0 1 0 1 0 1 1 1 1 0 0 1 XOR
0 0 0 1 tym uzupełniamy słowo i mamy
101010001
Lokalizacja błędu

1 1 1 0 1 0 1
7 6 5 4 3 2 1
Bierzemy same 1 i dodajemy do siebie
001
011
101
110

111 AND

101
001
110 ==> 6 miejsce błędu negujemy wartość
1 0 1 0 1 0 1

7 6 5 4 3 2 1

2. **Kody binarne**:
    - Dane są reprezentowane za pomocą ciągów 0 i 1.
    - Przykład: ASCII, kodowanie UTF-8.
3. **Kody nadmiarowe**:
    - Dodają dodatkowe bity do danych w celu wykrywania lub korekcji błędów.
    - Przykład: CRC (Cyclic Redundancy Check).
4. **Kody źródłowe**:
    - Służą do reprezentacji danych w sposób bardziej efektywny.
    - Przykład: Huffman, kodowanie arytmetyczne.

Wielkości charakteryzujące kody:

1. **Długość kodu**:
    - Liczba bitów używanych do reprezentacji pojedynczej wiadomości.
2. **Szybkość kodu (rate)**:
    - Stosunek liczby bitów informacyjnych do całkowitej liczby bitów w kodzie.
3. **Odległość Hamminga**:
    - Liczba bitów różniących dwa ciągi kodowe. Odległość Hamminga określa odporność na błędy.
4. **Efektywność kodu**:
    - Miara stopnia wykorzystania kodu w porównaniu z optymalnym kodowaniem.

3. Źródła informacji: bezpamięciowe i Markowa

Źródło informacji:

- Generuje dane (symbole, wiadomości) do przesłania przez kanał informacyjny.

Źródło bezpamięciowe:

- Generuje symbole niezależnie od siebie.
- Prawdopodobieństwo wystąpienia danego symbolu nie zależy od poprzednich symboli.
- Przykład: rzut monetą (symbole "orzeł" i "reszka").

Źródło Markowa:

- Generuje symbole, gdzie prawdopodobieństwo bieżącego symbolu zależy od poprzednich symboli.
- Przykład: tekst generowany przez model Markowa, gdzie prawdopodobieństwo kolejnego słowa zależy od poprzedniego.

4. Ilość informacji, entropia, twierdzenie Shannona o kodowaniu kanałów bezszumowych

Ilość informacji:

- Miara niepewności związanej z wynikiem zdarzenia.
- Dla zdarzenia o prawdopodobieństwie P:
  $I=-\log_2(P)$
- Jednostka: bit.

Entropia (H):

- Średnia ilość informacji przypadająca na jedno zdarzenie w źródle informacji.
- Definiowana jako: $H=-\sum P(x)\log_2 P(x)$
- Wyraża stopień niepewności źródła.

Twierdzenie Shannona:

- W kanale bezszumowym minimalna długość kodu do przesłania informacji wynosi tyle, ile wynosi entropia źródła.
- Praktyczne znaczenie: minimalizuje redundancję w przesyłanych danych.

5. Liczbowe systemy pozycyjne: system binarny i system U2

System binarny:

- Pozycyjny system liczbowy o podstawie 2.
- Cyfry: 0, 1.
- Przykład liczby: $1011_2=1\cdot2^3+0\cdot2^2+1\cdot2^1+1\cdot2^0=11_{10}$

System U2 (kod uzupełnienia do dwóch):

- Sposób reprezentacji liczb całkowitych w systemie binarnym.
- Wartość liczby ujemnej:

  $U_2(x)=2^n-x$

  gdzie n to liczba bitów.

- Przykład:
  - Liczba −3 w systemie 4-bitowym: $1111_2+1=1101_2$

System U2 (uzupełnienie do dwóch)

to jeden z najczęściej stosowanych sposobów reprezentowania liczb całkowitych w systemie binarnym, szczególnie w komputerach. System ten pozwala na reprezentację zarówno liczb dodatnich, jak i ujemnych, przy czym operacje arytmetyczne na liczbach w tym systemie (dodawanie, odejmowanie) są bardzo proste i mogą być wykonywane bez rozróżniania liczb dodatnich i ujemnych.

Reprezentacja liczb w systemie U2:

- **Liczby dodatnie** są reprezentowane w standardowy sposób, podobnie jak w systemie binarnym, bez potrzeby dopełnienia.
- **Liczby ujemne** są reprezentowane za pomocą tzw. "uzupełnienia do dwóch". Oznacza to, że aby przedstawić liczbę ujemną, należy wziąć jej wartość bezwzględną w postaci binarnej, odwrócić wszystkie bity (zmieniając 0 na 1, a 1 na 0), a następnie dodać 1 do wyniku.

Wzór:

Aby obliczyć reprezentację liczby ujemnej w systemie U2 na n bitach:

$U2(x)=2^n-|x|$

gdzie:

- x to liczba, którą chcemy przedstawić (liczba ujemna),
- n to liczba bitów w reprezentacji liczby,
- U2(x) to liczba zapisana w systemie U2.

Przykład: Liczba -3 w systemie U2 (na 4 bitach)

1. **Zaczynamy od liczby 3**. Chcemy przedstawić liczbę **-3** w systemie U2 na 4 bitach.
2. **Zapisujemy liczbę 3 w systemie binarnym**:

   $3_{10}=0011_2$(zapis na 4 bitach)

3. **Odwracamy wszystkie bity**:

   $0011_2 \rightarrow 1100_2$

4. **Dodajemy 1 do wyniku**:

   $1100_2+1=1101_2$

   Ostateczny wynik: **1101** to reprezentacja liczby **-3** w systemie U2 na 4 bitach.

Wartość liczby ujemnej w systemie U2:

Dla systemu U2, wartość liczby ujemnej jest obliczana jako:

$U2(x) = 2^n - |x|$

gdzie x to liczba ujemna, a n to liczba bitów.

Dla naszego przykładu (gdzie n=4):

$U2(-3) = 2^4 - 3 = 16 - 3 = 13$

I w systemie binarnym: $13_{10} = 1101_2$, co odpowiada uzyskanej reprezentacji liczby -3 w systemie U2.

Reprezentacja liczb w systemie U2:

Liczby dodatnie:

Liczby dodatnie są reprezentowane po prostu w systemie binarnym. Na przykład:

- $+3_{10} = 0011_2$ (w systemie U2 na 4 bitach).

Liczby ujemne:

Liczby ujemne są reprezentowane przez uzupełnienie do dwóch. Na przykład:

- $-3_{10} = 1101_2$ (w systemie U2 na 4 bitach),
- $-1_{10} = 1111_2$ (w systemie U2 na 4 bitach),
- $-2_{10} = 1110_2$ (w systemie U2 na 4 bitach).

Zalety systemu U2:

1. **Prostota operacji arytmetycznych**: Dodawanie i odejmowanie liczb binarnych w systemie U2 jest bardzo proste. Wszystkie operacje arytmetyczne mogą być wykonywane tak samo, niezależnie od tego, czy liczby są dodatnie, czy ujemne.
2. **Brak konieczności oddzielnego traktowania liczb ujemnych i dodatnich**: Dzięki temu, że liczby ujemne są reprezentowane jako uzupełnienie do dwóch, proces obliczeń jest uproszczony, a operacje na liczbach są realizowane na tej samej zasadzie.
3. **Jednoznaczność reprezentacji**: W systemie U2 jest tylko jedna reprezentacja liczby 0 (wszystkie bity są 0), co eliminuje problem istnienia dwóch reprezentacji zera (jak to ma miejsce w systemie z kodowaniem sign-magnitude).

Zastosowanie systemu U2:

System U2 jest powszechnie używany w komputerach i mikroprocesorach do reprezentowania liczb całkowitych, ponieważ upraszcza procesy arytmetyczne i logiczne, a także umożliwia wygodne wykonywanie operacji na liczbach ujemnych i dodatnich.

Dodawanie binarne

- 0 + 0 = 0
- 1 + 0 = 1 (i odwrotnie)
- 1 + 1 = 0 (przeniesienie 1 do następnej kolumny)

```
   1101
  +1011
 ________
  11000
```

1. 1 + 1 = 0 (przeniesienie 1),
2. 0 + 1 + 1 (przeniesienie) = 0 (przeniesienie 1),
3. 1 + 0 + 1 (przeniesienie) = 0 (przeniesienie 1),
4. 1 + 1 (przeniesienie) = 1.

Ostateczny wynik: 11000

Odejmowanie binarne

Będziemy stosować metodę pożyczania, podobnie jak w systemie dziesiętnym.

1. Rozpisujemy liczby w systemie binarnym, zapisując je w jednej linii:

```
  1011
- 0110
------------
```

2. Zaczynamy od najmniej znaczącego bitu (od prawej):

```
  1011
- 0110
------------
       1
```

3. Kolejny bit (drugi bit od prawej):

- **1 - 1** = **0** (brak pożyczania).

```
  1011
- 0110
------------
      01
```

4. Kolejny bit (trzeci bit od prawej):

- **0 - 1** = **pożyczamy**, więc zmieniamy wartość bitu na **10** i pożyczamy 1 do następnego bitu:

```
  1011
- 0110
------------
   1001
```

Teraz na trzecim bicie mamy **10** (bo pożyczyliśmy 1 z wyższego miejsca).

5. Ostatni bit (najbardziej znaczący):

- **0 - 0** z pożyczoną 1 daje **1** (nie ma potrzeby pożyczania, ponieważ już zostało pożyczone).

```
  1011
- 0110
------------
   0101
```

Mnożenie

Krok po kroku:

1. **Zapisujemy liczby w systemie binarnym:**

```
  101_2
×  11_2
---------
```

2. **Mnożymy pierwszy bit drugiej liczby (1) przez całą pierwszą liczbę (101):**

1×101=1011

Przepisujemy wynik:

```
  101_2
×  11_2
---------
  101    ← Pierwszy wynik (mnożenie przez 1)
```

3. **Mnożymy drugi bit drugiej liczby (1) przez całą pierwszą liczbę (101), ale przesuwamy wynik o jedno miejsce w lewo:**

1×101=101

Przesuwamy wynik o jedno miejsce w lewo (dodajemy 0 na końcu):

```
  101_2
×  11_2
---------
  101    ← Pierwszy wynik (mnożenie przez 1)
+ 1010    ← Drugi wynik (mnożenie przez 1, przesunięcie w lewo)
---------
```

4. **Dodajemy oba wyniki:**

```
  101
+ 1010
---------
 1111
```

Ostateczny wynik:

101×11=1111

Dzielnie

1. **Rozpisujemy liczby binarne:**

```
  1110_2 (dzielna)
÷   10_2  (dzielnik)
-------------
```

2. **Określamy, ile razy dzielnik (10) mieści się w pierwszych bitach dzielnej (1110).**

Rozpoczynamy dzielenie od najbardziej znaczącego bitu w dzielnej. Zaczynamy dzielić pierwsze dwa bity (11), ponieważ dzielnik (10) ma dwie cyfry.

10_2 mieści się w 11_2 raz.

Pisujemy pierwszy bit wyniku: **1**. Następnie mnożymy $10_2 \times 1_2 = 10_2$ i odejmujemy wynik od pierwszych dwóch bitów dzielnej:

```
  1110_2
-  10_2
------------
   010_2
```

Po tej operacji, mamy resztę 01_2, a wynik dzielenia na razie to **1**.

3. **Przenosimy kolejny bit dzielnej (0) do reszty.** Teraz mamy 010_2 i dzielimy przez 10_2.

10_2 mieści się w 010_2 zero razy.

Pisujemy drugi bit wyniku: **0**. Następnie przenosimy kolejny bit z dzielnej:

```
  1110_2
-  10_2
```

```
------------
   010_2
```

4. **Teraz dzielimy pozostałe 1002100_21002 przez 10210_2102.**

10_2 mieści się w 100_2 dwa razy..

Pisujemy trzeci bit wyniku: **1**. Mnożymy $10_2 \times 1_2 = 10_2$ i odejmujemy:

```
  1110_2
-  10_2
------------
   010_2
 -  10_2
------------
    00_2
```

5. **Ostateczny wynik:** Po wszystkich operacjach dzielenia otrzymujemy wynik dzielenia $1110_2 \div 10_2 = 111_2$, co odpowiada 7_{10}.

Wynik: 111_2

6. Działanie maszyny Turinga

Maszyna Turinga:

- Abstrakcyjny model obliczeń opisujący podstawy teorii algorytmów.
- Składa się z:
    - **Taśmy**: teoretycznie nieskończonej, podzielonej na komórki (zawierają symbole).
    - **Głowicy**: poruszającej się w lewo/prawo i modyfikującej symbole.
    - **Tablicy stanów**: określającej reguły przetwarzania.

Działanie maszyny Turinga:

1. **Odczyt symbolu** z bieżącej pozycji taśmy.
2. **Zmiana stanu** zgodnie z tablicą przejść.
3. **Zapis nowego symbolu** na taśmie.
4. **Przesunięcie głowicy** w lewo/prawo.
5. Powtarzanie kroków aż do osiągnięcia stanu końcowego.

Znaczenie:

- Maszyna Turinga jest uniwersalnym modelem obliczeniowym, który może symulować dowolny algorytm.
- Dowodzi istnienia problemów nierozstrzygalnych (np. problem stopu).

1. Omów znaczenie, działanie oraz najczęściej występujące typy kanałów informacyjnych.
2. Omów typy kodów oraz wielkości charakteryzujące kody.
3. Porównaj źródła bez pamięciowe oraz źródła Markowa.
4. Opisz pojęcia ilości informacji oraz entropii.
5. Omów twierdzenie Shannon'a o kodowaniu kanałów bezszumowych.
6. Omów liczbowe systemy pozycyjne oraz porównaj system binarny z systemem U2.
7. Opisz działanie maszyny Turinga.
8. Omów własności języków bezkontekstowych.
9. Omów własności języków regularnych.

1. Znaczenie, działanie oraz najczęściej występujące typy kanałów informacyjnych

Kanały informacyjne służą do przesyłania informacji między nadawcą a odbiorcą. W teorii informacji analizuje się ich właściwości, by zoptymalizować transmisję danych.

- **Znaczenie**: Umożliwiają efektywną komunikację, są podstawą systemów telekomunikacyjnych, internetu i transmisji danych.
- **Działanie**: Informacje kodowane są w formie sygnałów, które są przesyłane przez medium transmisyjne. Mogą być zakłócane przez szum lub straty danych.

Typy kanałów informacyjnych:

1. **Kanał bezszumowy**:
    - Brak zakłóceń, przesyłane dane są identyczne z odbieranymi.
2. **Kanał z szumem**:
    - Zakłócenia mogą wprowadzać błędy do przesyłanych danych.
3. **Kanał ciągły**:
    - Dane przesyłane w sposób ciągły (np. sygnały analogowe).
4. **Kanał dyskretny**:
    - Dane przesyłane w postaci dyskretnych symboli (np. bity).

2. Typy kodów oraz wielkości charakteryzujące kody

Typy kodów:

1. **Kody liniowe**: Stosowane w korekcji błędów (np. Hamming, Reed-Solomon).
2. **Kody binarne**: Dane reprezentowane jako ciągi 0 i 1.
3. **Kody źródłowe**: Kompresują dane (np. Huffman, arytmetyczne).
4. **Kody nadmiarowe**: Umożliwiają wykrywanie i korekcję błędów.

Wielkości charakteryzujące kody:

1. **Długość kodu**: Liczba bitów przypisana wiadomości.
2. **Szybkość kodu**: Stosunek bitów informacyjnych do całkowitej liczby bitów.
3. **Odległość Hamminga**: Liczba różnic między symbolami dwóch kodów.
4. **Efektywność**: Miara stopnia wykorzystania kodu w przesyłaniu informacji.

3. Porównaj źródła bez pamięciowe oraz źródła Markowa

Źródło bez pamięci:

- Generuje symbole niezależne od poprzednich.
- Przykład: rzut kostką.

Źródło Markowa:

- Prawdopodobieństwo wygenerowania symbolu zależy od poprzednich symboli.
- Modelowane za pomocą łańcuchów Markowa.

Porównanie:

- Źródła bez pamięci są prostsze w analizie, ale mniej realistyczne.
- Źródła Markowa lepiej odzwierciedlają złożone zależności (np. w języku naturalnym).

4. Opisz pojęcia ilości informacji oraz entropii

- **Ilość informacji**:
  - Miara redukcji niepewności związanej z wynikiem zdarzenia.
  - $I = -\log_2(P)$, gdzie P to prawdopodobieństwo zdarzenia.
- **Entropia (H)**:
  - Średnia ilość informacji przypadająca na symbol.
  - Wzór: $$H = -\sum P(x) \log_2 P(x)$$
  - Wyraża stopień niepewności lub losowości źródła

5. Omów twierdzenie Shannona o kodowaniu kanałów bezszumowych

Twierdzenie Shannona mówi, że:

- Minimalna długość kodu wymagana do przesłania informacji wynosi tyle, ile entropia źródła.
- Zastosowanie: projektowanie kodów o minimalnej redundancji.

6. Omów liczbowe systemy pozycyjne oraz porównaj system binarny z systemem U2

Liczbowe systemy pozycyjne:

- Wartość cyfry zależy od jej pozycji i podstawy systemu.
- Przykład: system dziesiętny, binarny.

System binarny:

- Podstawa: 2.

- Cyfry: 0, 1.
- Przykład: $101_2=5_{10}$

System U2 (uzupełnienie do dwóch):

- Reprezentacja liczb całkowitych w systemie binarnym.
- Liczby ujemne są kodowane jako $2^n–x$, gdzie n to liczba bitów.

Porównanie:

- System binarny używany do kodowania danych.
- System U2 służy do operacji arytmetycznych w komputerach.

7. Opisz działanie maszyny Turinga

Maszyna Turinga to abstrakcyjny model obliczeniowy.

- **Elementy**:
  - Taśma: nieskończona, podzielona na komórki.
  - Głowica: odczytuje/zapisuje symbole na taśmie.
  - Tablica przejść: reguły określające działanie maszyny.
- **Działanie**:

  1. Odczyt symbolu z taśmy.
  2. Zmiana stanu i zapis nowego symbolu.
  3. Przesunięcie głowicy.

Maszyna Turinga jest podstawą teorii algorytmów i modeli obliczeń.

8. Omów własności języków bezkontekstowych

- **Definicja**: Języki bezkontekstowe są definiowane za pomocą gramatyk, gdzie reguły mają postać $A→γ$ gdzie A to symbol nieterminalny, a γ to ciąg symboli.
- **Własności**:
  1. Mogą być rozpoznawane przez **automaty niedeterministyczne ze stosem**.
  2. Używane do definiowania składni języków programowania.
  3. Mają hierarchiczną strukturę, np. wyrażenia matematyczne.

9. Omów własności języków regularnych

- **Definicja**: Języki regularne są definiowane za pomocą wyrażeń regularnych lub automatów skończonych.
- **Własności**:
  1. Mogą być rozpoznawane przez **automaty skończone**.
  2. Nie obsługują rekurencyjnych struktur.
  3. Są używane do walidacji wzorców (np. w przetwarzaniu tekstu).
  4. Mają ograniczoną moc wyrazu w porównaniu do języków bezkontekstowych.

Oprogramowanie użytkowe

1. Sekcje w dokumentach wielostronicowych

Sekcje w dokumentach wielostronicowych umożliwiają podział dokumentu na różne części, z których każda może mieć niezależne formatowanie i układ.

- **Znaczenie sekcji**:
  - Umożliwiają różne układy strony (np. poziomy i pionowy) w jednym dokumencie.
  - Pozwalają na zmianę numeracji stron, nagłówków, stopki w różnych częściach dokumentu.
- **Zastosowanie**:
  - Rozdziały książek, raporty z różnymi układami graficznymi, dokumenty z dodatkami.
- **Przykładowe formaty w sekcjach**:
  - Różne marginesy i orientacje stron.
  - Oddzielne nagłówki i stopki.
  - Różne formatowanie kolumn.

2. Makropolecenia jako automatyzacja pracy użytkownika

Makropolecenia to zestawy instrukcji zapisane w formie skryptów, które umożliwiają automatyzację powtarzalnych zadań w oprogramowaniu biurowym, np. w programach takich jak MS Excel czy Word.

- **Znaczenie**:
  - Przyspieszają wykonywanie rutynowych zadań.
  - Minimalizują ryzyko błędów manualnych.
  - Pozwalają użytkownikowi na zaawansowaną personalizację pracy.
- **Działanie**:
  - Tworzenie makra przez nagranie wykonywanych czynności lub pisanie kodu w języku VBA (Visual Basic for Applications).
  - Wywołanie makra uruchamia zapisane operacje.
- **Zastosowanie**:
  - Formatowanie dużych zbiorów danych.
  - Generowanie raportów.
  - Automatyczne sortowanie i filtrowanie danych.

3. Adresowanie w arkuszach kalkulacyjnych: względne, bezwzględne i mieszane

Adresowanie określa sposób odwoływania się do komórek w arkuszach kalkulacyjnych (np. MS Excel).

- **Adresowanie względne**:
  - Przykład: `A1`.
  - Odnosi się do pozycji względnej komórki względem formuły.
  - Przy kopiowaniu formuła zmienia odniesienia do innych komórek.
- **Adresowanie bezwzględne**:
  - Przykład: `$A$1`.
  - Odnosi się do dokładnej pozycji komórki.
  - Przy kopiowaniu formuła zawsze odnosi się do tej samej komórki.
- **Adresowanie mieszane**:

- Przykład: $A1 lub A$1.
- Część odwołania jest stała, a część względna.
- Przykład zastosowania: kopiowanie formuł w tabelach zawierających wiersze i kolumny odniesienia.

4. Narzędzia do analizy i przetwarzania danych w arkuszach kalkulacyjnych

Arkusze kalkulacyjne, takie jak MS Excel, oferują szeroki zakres narzędzi do analizy i przetwarzania danych:

1. **Filtrowanie danych**:
   - Wyodrębnianie interesujących danych na podstawie określonych kryteriów.
2. **Sortowanie**:
   - Porządkowanie danych w sposób rosnący, malejący lub wg niestandardowych reguł.
3. **Tabele przestawne**:
   - Narzędzie do dynamicznego podsumowywania i analizy dużych zbiorów danych.
4. **Funkcje analityczne**:
   - Funkcje statystyczne (ŚREDNIA, MEDIANA, MAX, MIN).
   - Funkcje logiczne (JEŻELI, ORAZ, LUB).
5. **Wykresy**:
   - Wizualizacja danych w postaci słupków, linii, wykresów kołowych itp.
6. **Scenariusze i analiza „co-jeśli”**:
   - Przewidywanie wyników na podstawie zmiany danych wejściowych.
7. **Solver**:
   - Narzędzie optymalizacyjne do znajdowania najlepszych rozwiązań dla złożonych problemów.

5. Projektowanie relacyjnej bazy danych

Relacyjne bazy danych są zorganizowane w sposób pozwalający na przechowywanie danych w tabelach i nawiązywanie relacji między nimi.

Elementy relacyjnej bazy danych:

1. **Tabele**:
   - Podstawowe struktury do przechowywania danych w wierszach (rekordach) i kolumnach (atrybutach).
2. **Kwerendy**:
   - Zapytania pozwalające na przeszukiwanie i filtrowanie danych (np. w SQL: SELECT, WHERE, JOIN).
3. **Formularze**:
   - Graficzne interfejsy do wprowadzania i edycji danych.
4. **Raporty**:
   - Sformatowane podsumowania danych, często używane do prezentacji wyników.
5. **Makra**:
   - Automatyzują operacje w bazie danych (np. dodawanie rekordów).

Zasady projektowania:

1. **Normalizacja**:
   - Rozdzielenie danych na tabele w sposób eliminujący redundancję.
2. **Klucze główne i obce**:
   - Klucz główny identyfikuje unikalnie każdy rekord.
   - Klucz obcy wskazuje na powiązanie między tabelami.

Przykład zastosowania:

- Projekt bazy dla sklepu internetowego:
  - Tabela **Produkty**: nazwa, cena, kategoria.
  - Tabela **Zamówienia**: data, klient, kwota.
  - Relacja między tabelami: produkt -> zamówienie.

1. Wyjaśnij na czym polega typ relacji jeden-do-wielu w relacyjnych bazach danych. Przedstaw na przykładowym schemacie.
2. Wyjaśnij pojęcie klucz główny (podstawowy) i rolę jaką pełni w relacyjnych bazach danych.
3. Rodzaje adresowania w arkuszu kalkulacyjnych EXCEL.
4. Podaj podstawowe grupy funkcji i wymień kilka przykładów.

1. Relacja jeden-do-wielu w relacyjnych bazach danych

Wyjaśnienie:

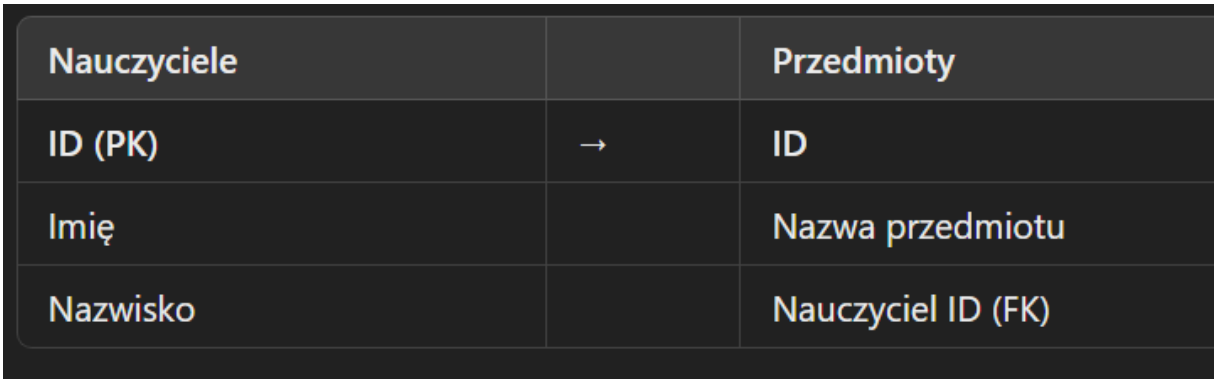
Relacja jeden-do-wielu (1:N) w relacyjnych bazach danych oznacza, że jeden rekord w jednej tabeli jest powiązany z wieloma rekordami w innej tabeli. Jest to typowa relacja stosowana w systemach bazodanowych.

Przykład:

Rozważmy bazę danych dla szkoły, gdzie mamy dwie tabele: **Nauczyciele** i **Przedmioty**.

- Jeden nauczyciel może uczyć wielu przedmiotów.
- Każdy przedmiot jest przypisany tylko do jednego nauczyciela.

Schemat:



- **PK** – Klucz główny.
- **FK** – Klucz obcy.

Relacja jest realizowana przez **klucz obcy** w tabeli **Przedmioty**, który odnosi się do klucza głównego w tabeli **Nauczyciele**.

2. Klucz główny (Primary Key) w relacyjnych bazach danych

Definicja:

Klucz główny (Primary Key) to kolumna lub zestaw kolumn w tabeli, które jednoznacznie identyfikują każdy rekord.

Cechy klucza głównego:

1. **Unikalność**: Każda wartość w kluczu głównym musi być unikalna.
2. **Brak wartości NULL**: Klucz główny nie może zawierać wartości pustych.
3. **Stabilność**: Klucz główny nie powinien ulegać zmianom w czasie.

Rola klucza głównego:

- Identyfikacja rekordów w tabeli.
- Tworzenie relacji między tabelami w bazie danych (np. powiązanie z kluczem obcym).

Przykład:

W tabeli **Nauczyciele** klucz główny to kolumna **ID**, ponieważ każdemu nauczycielowi przypisujemy unikalny identyfikator.

3. Rodzaje adresowania w arkuszu kalkulacyjnym Excel

W Excelu stosowane są trzy typy adresowania komórek:

1. Adresowanie względne:

- Przykład: A1.
- Odnosi się do pozycji komórki względem formuły.
- Przy kopiowaniu formuła zmienia odniesienie do innych komórek.

2. Adresowanie bezwzględne:

- Przykład: A1.
- Odnosi się do dokładnej lokalizacji komórki, niezależnie od miejsca, gdzie zostanie skopiowana formuła.

3. Adresowanie mieszane:

- Przykład: $A1 lub A$1.
- Część odwołania jest stała, a część względna:

- $A1: Kolumna jest stała, a wiersz zmienia się przy kopiowaniu.
- A$1: Kolumna zmienia się, a wiersz pozostaje stały.

4. Podstawowe grupy funkcji w Excelu i przykłady

Grupy funkcji:

1. **Funkcje matematyczne i statystyczne**:
   - SUMA: Dodawanie liczb.
   - ŚREDNIA: Obliczanie średniej arytmetycznej.
   - MIN, MAX: Znalezienie wartości minimalnej i maksymalnej.
2. **Funkcje logiczne**:
   - JEŻELI: Warunkowe wykonanie operacji.
   - LUB, ORAZ: Operacje logiczne.
3. **Funkcje tekstowe**:
   - LEWY: Wyciągnięcie znaków z lewej strony tekstu.
   - DŁ: Obliczenie długości tekstu.
   - ZŁĄCZ.TEKST: Łączenie tekstów.
4. **Funkcje daty i czasu**:
   - DZIŚ: Zwraca dzisiejszą datę.
   - TERAZ: Zwraca bieżącą datę i godzinę.
   - ROK, MIESIĄC: Wydobycie roku lub miesiąca z daty.
5. **Funkcje wyszukiwania i adresowania**:
   - WYSZUKAJ.PIONOWO: Wyszukiwanie wartości w kolumnie.
   - ADR.POŚREDNI: Dynamiczne odwoływanie się do adresu komórki.
6. **Funkcje finansowe**:
   - PMT: Obliczanie raty kredytu.
   - NPV: Obliczanie wartości bieżącej netto.

Przykłady zastosowań:

- Tworzenie raportów finansowych (np. za pomocą SUMA i JEŻELI).
- Analiza danych sprzedaży (np. ŚREDNIA, MAX).
- Tworzenie harmonogramów i kalendarzy (np. DZIŚ, TERAZ).

Algorytmy i struktury danych

1. Złożoność obliczeniowa i notacja asymptotyczna

Złożoność obliczeniowa:

Złożoność obliczeniowa mierzy efektywność algorytmu w kontekście:

- **Czasu**: Ile czasu zajmuje wykonanie algorytmu w zależności od rozmiaru danych wejściowych.
- **Pamięci**: Ile pamięci wymaga algorytm podczas działania.

Notacja asymptotyczna:

Notacje asymptotyczne opisują zachowanie algorytmu dla bardzo dużych danych wejściowych n.

- **Notacja O (O-złożoność, "O-od górna")**:
  - Górna granica wzrostu funkcji.
  - Opisuje najgorszy przypadek.
  - Przykład: $O(n^2)$, O(logn)
- **Notacja Θ (Theta-złożoność)**:
  - Ścisła granica wzrostu funkcji.
  - Algorytm działa w czasie proporcjonalnym do tej funkcji.
  - Przykład: $\Theta(n^2)$
- **Notacja Ω (Omega-złożoność)**:
  - Dolna granica wzrostu funkcji.
  - Opisuje najlepszy przypadek działania algorytmu.

2. Algorytmy wyszukiwania i sortowania

Algorytmy wyszukiwania:

- **Wyszukiwanie liniowe**:
  - Przeszukiwanie każdego elementu sekwencyjnie.
  - Złożoność: O(n)
- **Wyszukiwanie binarne**:
  - Działa na posortowanych danych. Podział zbioru na pół i przeszukiwanie odpowiedniej części.
  - Złożoność: O(logn)

Algorytmy sortowania:

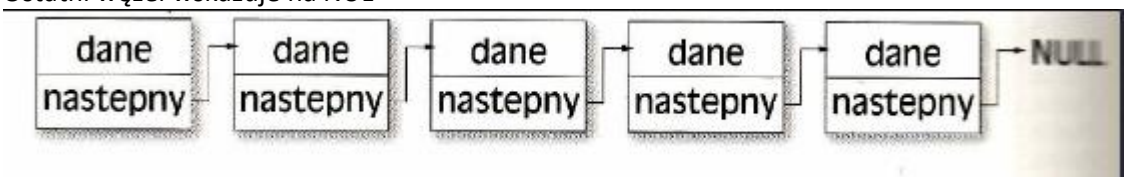
- **Bąbelkowe (Bubble Sort)**:
  - Porównywanie sąsiednich elementów i ich zamiana.
  - Złożoność: $O(n^2)$
- **Szybkie (Quick Sort)**:
  - Podział zbioru (partycjonowanie) i rekurencyjne sortowanie części.
  - Złożoność: O(nlogn) w typowym przypadku.
- **Scalanie (Merge Sort)**:
  - Dziel i zwyciężaj: dzielenie na mniejsze części i scalanie.
  - Złożoność: O(nlogn)
- **Sortowanie przez wstawianie**:
  - Wstawianie elementu w odpowiednie miejsce w uporządkowanym fragmencie.
  - Złożoność: $O(n^2)$

3. Listy z dowiązaniami

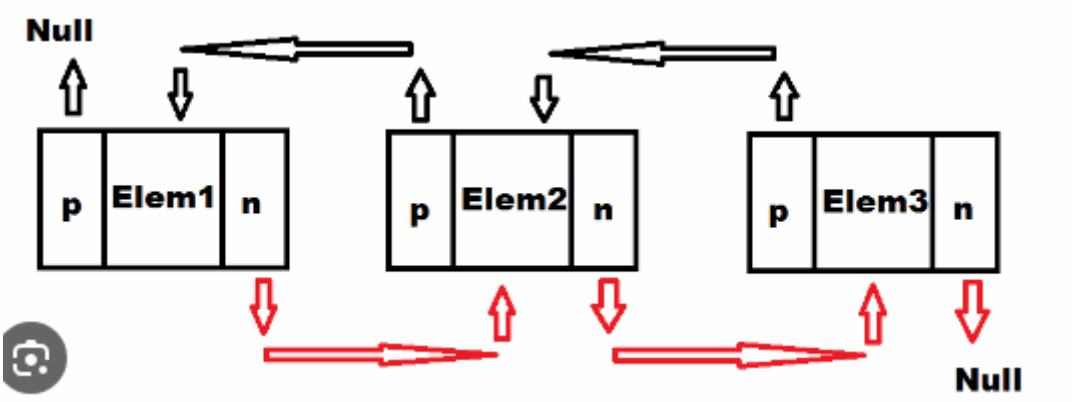
Listy z dowiązaniami to struktura danych, gdzie elementy (węzły) są połączone za pomocą wskaźników.

Rodzaje:

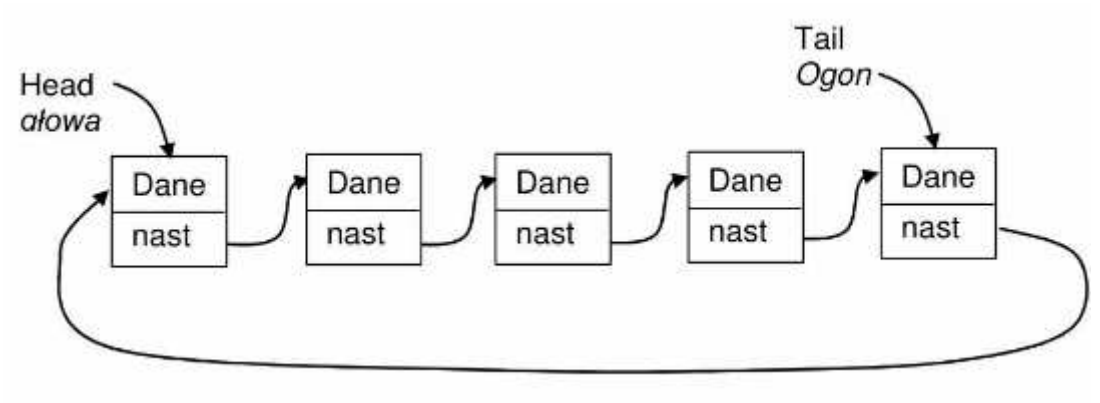
- **Lista jednokierunkowa**:
  - Każdy węzeł wskazuje na następny węzeł.
  - Ostatni węzeł wskazuje na NUL

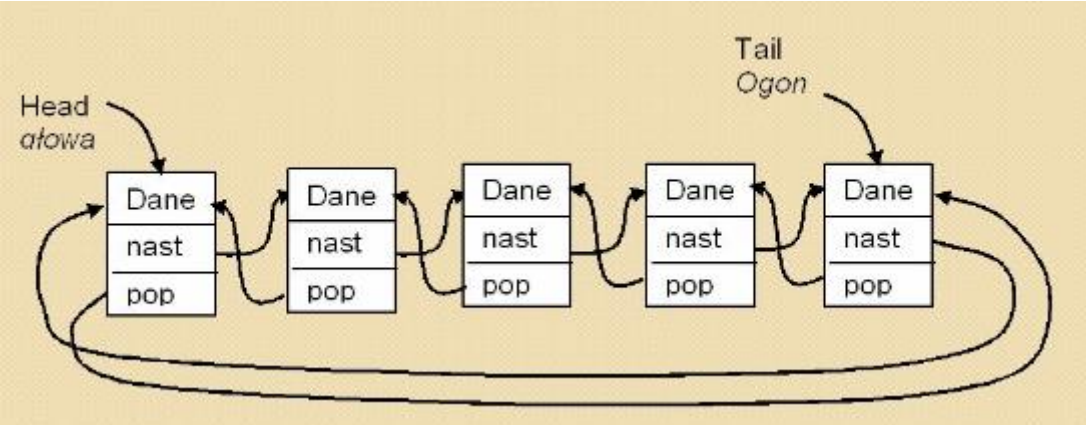


- **Lista dwukierunkowa**:
  - Każdy węzeł wskazuje na poprzedni i następny węzeł.



- **Lista cykliczna**:
  - Ostatni węzeł wskazuje na pierwszy, tworząc cykl.





Operacje:

- Dodawanie, usuwanie węzłów.
- Przeszukiwanie i iteracja.

4. Stosy, kolejki, kopiec binarny, kolejki priorytetowe

Stosy:

- LIFO (Last In, First Out): ostatni element, który został dodany, jest usuwany jako pierwszy.
- Operacje:
  - **Push**: dodanie elementu.
  - **Pop**: usunięcie elementu.
  - **Peek**: podgląd elementu na szczycie.
- Zastosowanie:
  - Rekurencja, zarządzanie wywołaniami funkcji.

Kolejki:

- FIFO (First In, First Out): pierwszy dodany element jest usuwany jako pierwszy.
- Operacje:
  - **Enqueue**: dodanie elementu.
  - **Dequeue**: usunięcie elementu.
- Zastosowanie:
  - Przetwarzanie wątków, planowanie zadań.

Kopiec binarny:

- Struktura drzewa binarnego, gdzie każdy węzeł spełnia warunek:
  - W kopcu minimalnym: wartość węzła jest mniejsza niż jego dzieci.
  - W kopcu maksymalnym: wartość węzła jest większa niż jego dzieci.
- Zastosowanie:
  - Implementacja kolejek priorytetowych, algorytmy sortowania.

Kolejki priorytetowe:

- Elementy mają przypisane priorytety.

- Usuwany jest element z najwyższym priorytetem.

5. Drzewa

Drzewa binarne:

- Każdy węzeł ma maksymalnie dwóch potomków.
- Zastosowanie: analiza wyrażeń, sortowanie.

Drzewa BST (Binary Search Tree):

- Wartości w lewym poddrzewie są mniejsze, a w prawym większe od węzła rodzica.
- Zastosowanie: szybkie wyszukiwanie, wstawianie i usuwanie.

Drzewa AVL:

- Zrównoważone drzewa BST, gdzie różnica wysokości lewego i prawego poddrzewa wynosi maksymalnie 1.
- Zastosowanie: minimalizacja złożoności operacji.

B-drzewa:

- Drzewa z wieloma kluczami w jednym węźle, używane w systemach baz danych.
- Zastosowanie: struktury indeksów w bazach danych.

6. Grafy i podstawowe algorytmy grafowe

Grafy:

- Zbiór węzłów (wierzchołków) i krawędzi (połączeń między wierzchołkami).

Rodzaje grafów:

- **Skierowane i nieskierowane**.
- **Ważone** (krawędzie mają przypisane wartości).

Reprezentacje grafów:

- Lista sąsiedztwa.
- Macierz sąsiedztwa.

Algorytmy grafowe:

- **Przeszukiwanie wszerz (BFS)**:
  - Przeszukiwanie wierzchołków warstwami, zaczynając od wierzchołka startowego.
  - Złożoność: O(V+E) gdzie V to liczba wierzchołków, a E to liczba krawędzi.
- **Przeszukiwanie w głąb (DFS)**:
  - Przeszukiwanie wzdłuż ścieżki do najdalszego wierzchołka, potem cofanie.
  - Złożoność: O(V+E)

- **Algorytm Dijkstry**:
  - Znajdowanie najkrótszej ścieżki w grafach ważonych (bez ujemnych wag).
  - Złożoność: O((V+E)logV)
- **Algorytm Kruskala**:
  - Znajdowanie minimalnego drzewa rozpinającego.
  - Złożoność: O(ElogE)
- **Algorytm Floyda-Warshalla**:
  - Znajdowanie najkrótszych ścieżek między wszystkimi parami wierzchołków.
  - Złożoność: $O(V^3)$

- 1. Wymień podstawowe algorytmy sortowania i opisz ich klasy złożoności czasowej.
- 2. Wyjaśnij pojęcia pesymistycznej i średniej złożoności algorytmów.
- 3. Tablice haszujące - idea, rozwiązania problemu kolizji.
- 4. Rekurencja – idea, przykłady algorytmów rekurencyjnych.
- 5. Podstawowe klasy złożoności obliczeniowej.
- 6. Algorytmy przechodzenia po grafach – przykłady algorytmów, idea, złożoność.
- 7. Algorytmy wyszukiwania najkrótszych dróg w grafach – przykłady algorytmów, idea, złożoność.
- 8. Binarne drzewa poszukiwań - definicja, operacje na drzewach.
- 9. Metoda dziel i zwyciężaj konstrukcji algorytmów - idea, przykłady algorytmów.

1. Podstawowe algorytmy sortowania i ich złożoności czasowe

Algorytmy sortowania:

1. **Sortowanie bąbelkowe (Bubble Sort)**:
   - Złożoność pesymistyczna, średnia i optymistyczna: $O(n^2)$
   - Prosty do implementacji, ale nieefektywny dla dużych zbiorów danych.
2. **Sortowanie przez wstawianie (Insertion Sort)**:
   - Złożoność pesymistyczna i średnia: $O(n^2)$
   - Złożoność optymistyczna (dla posortowanego zbioru): O(n)
3. **Sortowanie przez wybieranie (Selection Sort)**:
   - Złożoność pesymistyczna, średnia i optymistyczna: $O(n^2)$
   - Minimalna liczba zamian, ale czasochłonny.
4. **Sortowanie szybkie (Quick Sort)**:
   - Złożoność pesymistyczna: $O(n^2)$ (dla złego podziału).
   - Złożoność średnia i optymistyczna: O(nlogn)
5. **Sortowanie przez scalanie (Merge Sort)**:
   - Złożoność pesymistyczna, średnia i optymistyczna: O(nlogn) Wymaga dodatkowej pamięci.
6. **Sortowanie kubełkowe (Bucket Sort)**:
   - Złożoność optymistyczna: O(n) (dla równomiernego rozkładu).
   - Średnia zależy od liczby kubełków: O(n+k), gdzie k to liczba kubełków.
7. **Sortowanie przez kopcowanie (Heap Sort)**:
   - Złożoność pesymistyczna, średnia i optymistyczna: O(nlogn)

2. Pojęcia pesymistycznej i średniej złożoności algorytmów

- **Pesymistyczna złożoność**:
  - Maksymalny czas działania algorytmu dla najgorszego przypadku danych wejściowych.
  - Przykład: Wyszukiwanie liniowe O(n) dla nieznalezionego elementu.
- **Średnia złożoność**:
  - Uśredniony czas działania algorytmu dla wszystkich możliwych danych wejściowych.
  - Przykład: Wyszukiwanie binarne O(logn) dla uporządkowanego zbioru.

3. Tablice haszujące – idea, rozwiązania problemu kolizji

- **Idea tablic haszujących**:
  - Struktura danych pozwalająca na szybkie wyszukiwanie, wstawianie i usuwanie elementów w czasie O(1) w typowych przypadkach.
  - Każdy element ma przypisany klucz, przekształcany na indeks za pomocą funkcji haszującej.
- **Rozwiązania problemu kolizji**:

1. **Łańcuchowanie**:
   - Każdy indeks tablicy zawiera listę elementów.
   - Kolizje są przechowywane w tej liście.
2. **Adresowanie otwarte**:
   - Znajdowanie innego indeksu w tablicy, gdy wystąpi kolizja (np. liniowe lub kwadratowe przesuwanie).

4. Rekurencja – idea, przykłady algorytmów rekurencyjnych

- **Idea rekurencji**:
  - Funkcja wywołuje samą siebie z mniejszymi wartościami danych wejściowych.
  - Wymaga warunku stopu, aby uniknąć nieskończonego wywoływania.
- **Przykłady algorytmów rekurencyjnych**:

1. Obliczanie silni: $n!=n\cdot(n-1)!$
2. Ciąg Fibonacciego: $F(n)=F(n-1)+F(n-2)$
3. Sortowanie szybkie (Quick Sort):
   - Rekurencyjny podział na części mniejsze i większe.

5. Podstawowe klasy złożoności obliczeniowej

1. **P**:
   - Problemy, które można rozwiązać w czasie wielomianowym (np. $O(n^2)$)
2. **NP**:
   - Problemy, dla których rozwiązanie można zweryfikować w czasie wielomianowym.
3. **NP-zupełne**:
   - Najtrudniejsze problemy w klasie NP.
4. **NP-trudne**:
   - Problemy równie trudne co NP-zupełne, ale niekoniecznie w NP.

6. Algorytmy przechodzenia po grafach

Przykłady algorytmów:

1. **Przeszukiwanie wszerz (BFS)**:
    - Przegląda wierzchołki warstwami.
    - Złożoność: O(V+E)
2. **Przeszukiwanie w głąb (DFS)**:
    - Eksploruje jak najgłębsze ścieżki, a potem się cofa.
    - Złożoność: O(V+E)

7. Algorytmy wyszukiwania najkrótszych dróg w grafach

Przykłady algorytmów:

1. **Dijkstra**:
    - Znajduje najkrótszą ścieżkę w grafach ważonych bez ujemnych wag.
    - Złożoność: O((V+E)logV)
2. **Bellman-Ford**:
    - Działa na grafach z ujemnymi wagami.
    - Złożoność: O(V·E)
3. **Floyd-Warshall**:
    - Oblicza najkrótsze ścieżki między wszystkimi parami wierzchołków.
    - Złożoność: $O(V^3)$

8. Binarne drzewa poszukiwań (BST)

- **Definicja**:
    - Drzewo binarne, w którym dla każdego węzła wartości w lewym poddrzewie są mniejsze, a w prawym większe.
- **Operacje**:

1. **Wstawianie**: Dodanie nowego węzła w odpowiedniej lokalizacji.
2. **Wyszukiwanie**: Znalezienie węzła o określonej wartości.
3. **Usuwanie**: Trzy przypadki:
    - Brak dzieci.
    - Jedno dziecko.
    - Dwoje dzieci (zamiana z następnikiem).

9. Metoda dziel i zwyciężaj

- **Idea**:
    - Problem jest dzielony na mniejsze podproblemy, które są rozwiązywane niezależnie.
    - Wyniki podproblemów są łączone, aby uzyskać rozwiązanie całości.
- **Przykłady algorytmów**:

- Sortowanie szybkie (Quick Sort).

- Sortowanie przez scalanie (Merge Sort).

- **Algorytm Karatsuby**: szybkie mnożenie dużych liczb.

Organizacja i architektura komputerów

1. Reprezentacja informacji w komputerze - arytmetyka i logika

Reprezentacja liczb:

- **Liczby całkowite**:
  - System binarny: liczby są reprezentowane w postaci ciągu bitów (0 i 1).
  - Uzupełnienie do dwóch (U2): metoda reprezentowania liczb całkowitych ze znakiem.
- **Liczby zmiennoprzecinkowe**:
  - Format zgodny ze standardem IEEE 754:
    - Składa się z bitu znaku, mantysy i wykładnika.
    - Używany do reprezentacji liczb rzeczywistych.

Logika:

- **Podstawowe operacje logiczne**:
  - **AND**, **OR**, **NOT**, **XOR**.
- **Arytmetyka binarna**:
  - Dodawanie, odejmowanie, mnożenie i dzielenie w systemie binarnym.
  - Wykorzystywane w arytmetyczno-logicznych jednostkach CPU (ALU).

p	q	FAŁSZ	p AND q	p XOR q	p OR q	p NOR q	p XNOR q	NOT q	q → p	NOT p	p → q	p NAND q	PRAWDA
0	**0**	0	0	0	0	1	1	1	1	1	1	1	1
0	**1**	0	0	1	1	0	0	0	0	1	1	1	1
1	**0**	0	0	1	1	0	0	1	1	0	0	1	1
1	**1**	0	1	0	1	0	1	0	1	0	1	0	1

2. Układy kombinacyjne i sekwencyjne

Układy kombinacyjne:

- **Definicja**:
  - Wyjścia zależą tylko od bieżących wejść.
- **Przykłady**:
  1. Sumator:
     - Dodaje dwie liczby binarne.
  2. Dekoder:
     - Zamienia kod binarny na sygnały wyjściowe.
  3. Multiplekser:
     - Wybiera jedno z wielu wejść do przesłania na wyjście.

- **Projektowanie**:
    - Przy użyciu tabel Karnaugha do upraszczania funkcji logicznych.

Układy sekwencyjne:

- **Definicja**:
    - Wyjścia zależą od bieżących wejść i poprzednich stanów (z pamięcią).
- **Przykłady**:
    1. Przerzutniki (RS, JK, D, T):
        - Pamięć jednego bitu.

Typy przerzutników:

1. **Przerzutnik RS (Set-Reset):**
    - **Opis:** Najprostszy typ przerzutnika, działa na dwóch wejściach: **Set (S)** i **Reset (R)**.
    - **Zasada działania:**
        - Jeśli S=1 i R=0, przerzutnik ustawia się na stan **1** (Set).
        - Jeśli S=0 i R=1, przerzutnik ustawia się na stan **0** (Reset).
        - Jeśli S=0 i R=0, przerzutnik utrzymuje swój poprzedni stan.
        - Jeśli S=1 i R=1, stan jest **nieokreślony** (zabroniony).
2. **Przerzutnik JK:**
    - **Opis:** Modyfikacja przerzutnika RS, eliminuje stan zabroniony. Dwa wejścia: J (Set) i K (Reset).
    - **Zasada działania:**
        - Jeśli J=1 i K=0, przerzutnik przechodzi w stan **1**.
        - Jeśli J=0 i K=1, przerzutnik przechodzi w stan **0**.
        - Jeśli J=1 i K=1, przerzutnik zmienia stan na przeciwny (przełączenie).
        - Jeśli J=0 i K=0, przerzutnik utrzymuje poprzedni stan.
3. **Przerzutnik D (Data/Delay):**
    - **Opis:** Przerzutnik pamiętający dane. Ma jedno wejście D.
    - **Zasada działania:**
        - Jeśli zegar (CLK) aktywny, przerzutnik przyjmuje wartość wejścia D i przechowuje ją do następnego impulsu zegara.
4. **Przerzutnik T (Toggle):**
    - **Opis:** Przełącza stan między **0** i **1** przy każdym impulsie zegara, gdy wejście T=1.
    - **Zasada działania:**
        - Jeśli T=0, przerzutnik utrzymuje swój stan.
        - Jeśli T=1, przerzutnik zmienia swój stan na przeciwny.

    2. Liczniki:
        - Układy zliczające impulsy.
    3. Rejestry:
        - Przechowywanie i przesyłanie danych.

- **Projektowanie**:
    - Wymaga analizy czasowej i diagramów stanów.

3. Jednostka centralna i pamięć

Jednostka centralna (CPU):

- **Składniki**:
    1. **ALU (Arytmetyczno-Logiczna Jednostka)**:
        - Wykonuje operacje matematyczne i logiczne.
    2. **CU (Jednostka Sterująca)**:
        - Zarządza wykonywaniem instrukcji.
    3. **Rejestry**:
        - Przechowują dane tymczasowe.
- **Cykl maszyny**:
    1. Faza pobrania (fetch): instrukcja jest pobierana z pamięci.
    2. Faza dekodowania (decode): instrukcja jest analizowana.
    3. Faza wykonania (execute): operacja jest wykonywana.

Pamięć:

- **Rodzaje**:
    1. **RAM (Random Access Memory)**:
        - Pamięć ulotna, przechowuje dane tymczasowe.
    2. **ROM (Read-Only Memory)**:
        - Pamięć nieulotna z zapisanymi na stałe danymi.
    3. **Cache**:
        - Szybka pamięć bliska procesorowi, używana do przechowywania często używanych danych.
    4. **Pamięć masowa**:
        - Dyski HDD, SSD.
- **Hierarchia pamięci**:
    - Rejestry → Cache → RAM → Dysk.

4. Organizacja równoległa

Definicja:

- Organizacja równoległa pozwala na wykonywanie wielu operacji jednocześnie, co zwiększa wydajność systemu komputerowego.

Rodzaje równoległości:

1. **Równoległość na poziomie instrukcji (ILP)**:
    - Wykorzystanie potokowania (pipelining), gdzie instrukcje są wykonywane równolegle.
2. **Równoległość na poziomie wątków**:
    - Równoczesne wykonywanie wątków na różnych rdzeniach CPU.
3. **Równoległość danych**:

- Wykorzystanie SIMD (Single Instruction, Multiple Data) do przetwarzania wielu danych jedną instrukcją.

Architektury równoległe:

1. **MIMD (Multiple Instruction, Multiple Data)**:
   - Każdy procesor wykonuje różne instrukcje na różnych danych.
2. **SIMD (Single Instruction, Multiple Data)**:
   - Jedna instrukcja wykonywana równocześnie na wielu danych.
3. **Potokowanie**:
   - Podział procesu na etapy, wykonywane jednocześnie.

Zastosowania:

- W obliczeniach naukowych, przetwarzaniu grafiki, analizach dużych danych i symulacjach.

1. Z sumuj oraz przemnóż pisemnie dwie liczby binarne: X i Y. Wyniki podaj w systemie binarnym, dziesiętnym, ósemkowym oraz szesnastkowym.
2. Na podstawie tabeli prawdy podanej na rysunku zminimalizuj dowolną metodą funkcję dla poszczególnych zmiennych. Wynik zapisz jako sumy iloczynów.
3. Narysuj z wykorzystaniem bramek logicznych realizację funkcji.

1. Dodawanie i mnożenie liczb binarnych

Dodawanie dwóch liczb binarnych:

Przykład:
X=1101_2 (13 w systemie dziesiętnym)
Y=1011_2 (11 w systemie dziesiętnym)

Dodajemy pisemnie:

```
 1111
  1101
+ 1011
----------
 11000
```

Wynik w systemie binarnym: 11000211000_2110002

- Dziesiętnie: 24_{10}
- Ósemkowo: 30_8
- Szesnastkowo: 18_{16}

Mnożenie dwóch liczb binarnych:

Przykład:
X=1101_2
Y=1011_2

Mnożymy pisemnie (podobnie jak w systemie dziesiętnym, ale z użyciem tylko 0 i 1):

```
    1101
  × 1011
  ----------
    1101      (1101 × 1)
+   1101      (1101 × 1, przesunięte o 1 pozycję)
+0000         (1101 × 0, przesunięte o 2 pozycje)
+ 1101        (1101 × 1, przesunięte o 3 pozycje)
  ----------
  1001111
```

Wynik w systemie binarnym: 1001111_2

Dziesiętnie: 143_{10}

Ósemkowo: 217_8

Szesnastkowo: $8F_{16}$

2. Minimalizacja funkcji logicznej z tabeli prawdy

Przykładowa tabela prawdy (dla trzech zmiennych A,B,C):

A	B	C	$F(A,B,C)$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Metoda minimalizacji (mapa Karnaugha):

$AB \rightarrow C \downarrow$	00	01	10	11
0	1	0	1	0
1	1	0	1	0

Grupujemy jedynki:

1. Grupa 1 (pierwsza kolumna): A=1
2. Grupa 2 (trzecia kolumna): C=0

Wynik:

Minimalna postać:
F(A,B,C)=A+¬C
(w postaci sumy iloczynów: F=A+¬C)

¬ C➔ Negacja C

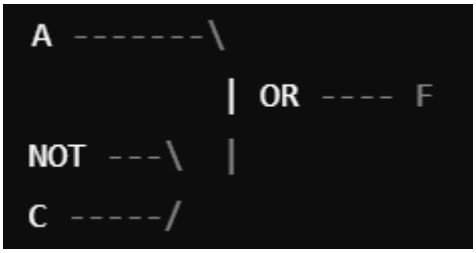
3. Realizacja funkcji za pomocą bramek logicznych

Funkcja F(A,B,C)=A+¬C wymaga dwóch bramek:

1. **NOT**: realizuje ¬C.
2. **OR**: realizuje sumę A+¬C

Schemat:

1. Wejścia: A, C.
2. Bramka NOT:
    - Wejście: C.
    - Wyjście: ¬C
3. Bramka OR:
    - Wejścia: A,¬C
    - Wyjście: F.



Matematyka 1

1. Tautologie i kontrtautologie klasycznego rachunku zdań oraz klasycznego rachunku kwantyfikatorów

Tautologia
Tautologia to wyrażenie logiczne, które jest zawsze prawdziwe, niezależnie od wartości logicznych zdań składowych. W klasycznym rachunku zdań przykładami tautologii są:

- $p \vee \neg p$ (prawo wyłączonego środka),
- $(p \implies q) \vee (\neg p)$ (prawo wyłączonej alternatywy).

W rachunku kwantyfikatorów tautologie mogą dotyczyć wszystkich możliwych wartości zmiennych, np.:

- $\forall x(P(x) \vee \neg P(x))$
- $\exists x(P(x)) \implies \neg(\forall x(\neg P(x)))$

Kontrtautologia
Kontrtautologia to wyrażenie logiczne, które jest zawsze fałszywe, niezależnie od wartości logicznych zdań składowych. Przykłady kontrtautologii w klasycznym rachunku zdań:

- $p \wedge \neg p$ (sprzeczność).
  W rachunku kwantyfikatorów przykładem może być:
- $\forall x(P(x) \wedge \neg P(x))$.

2. Funkcje zdaniowe

Funkcja zdaniowa to wyrażenie logiczne zależne od zmiennych, które staje się zdaniem logicznym, gdy te zmienne przyjmą konkretne wartości. Przykładowo:

- Funkcja zdaniowa: $P(x):x>0$
- Gdy $x=2$, $P(x)$ staje się zdaniem $2>0$, które jest prawdziwe.

Funkcje zdaniowe mogą być używane w rachunku kwantyfikatorów do definiowania właściwości obiektów, np. $\forall x(P(x))$, gdzie $P(x)$ jest funkcją zdaniową.

3. Indukcja matematyczna i jej zastosowania

Zasada indukcji matematycznej
Służy do dowodzenia prawdziwości zdań dla wszystkich liczb naturalnych n. Składa się z dwóch kroków:

1. **Baza indukcji**: Udowadnia się, że zdanie jest prawdziwe dla $n=1$.
2. **Krok indukcyjny**: Zakłada się, że zdanie jest prawdziwe dla $n=k$ (założenie indukcyjne), i dowodzi się, że jest prawdziwe dla $n=k+1$.

Zastosowania:

- Dowód wzorów sumacyjnych, np. $\sum i=(n(n+1))/2$,
- Dowody dotyczące podzielności liczb,
- Problemy kombinatoryczne i własności ciągów.

Rozwiązywanie problemów za pomocą indukcji matematycznej – krok po kroku

Indukcja matematyczna to metoda dowodzenia, która pozwala wykazać prawdziwość twierdzeń dla nieskończonego zbioru liczb naturalnych n. Oto szczegółowy opis procesu:

Założenie wstępne:

Chcesz dowieść, że pewne twierdzenie P(n) jest prawdziwe dla wszystkich $n \in N$

Krok 1: Baza indukcji

1. Sprawdź, czy twierdzenie P(n) jest prawdziwe dla najmniejszej liczby naturalnej n_0 (zwykle $n_0=1$)
2. Wykaż, że P(n_0) spełnia formułę.

Przykład: Dowód wzoru na sumę

$$S(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

- Sprawdź dla n=1:

S(1)=1,a według wzoru: (1 * (1+ 1))/2=1

Baza indukcji jest prawdziwa.

Krok 2: Założenie indukcyjne

1. Przyjmij, że twierdzenie P(k) jest prawdziwe dla pewnej liczby $k \geq n_0$ tj.:
   P(k) jest prawdziwe, czyli S(k)=(k(k+1))/2. To nazywa się **założeniem indukcyjnym**.
2. **Nie dowodzisz tego kroku** – zakładasz jego prawdziwość.

Krok 3: Krok indukcyjny

1. Udowodnij, że jeśli P(k) jest prawdziwe, to P(k+1) też jest prawdziwe.
2. Zastąp n=k+1 w formule i pokaż, że jest zgodna z założeniem indukcyjnym.

Przykład (kontynuacja): Dowód wzoru na sumę.
Zakładamy, że dla n=k:

$$S(k) = \frac{k(k+1)}{2}$$

Musimy wykazać, że dla n=k+1:

$$S(k+1) = \frac{(k+1)(k+1+1)}{2}$$

Z definicji sumy:

$$S(k+1) = s(k) + (k+1)$$

- Podstaw założenie indukcyjne :

$$S(k+1) = \frac{k(k+1)}{2} + (k+1)$$

- Sprowadź do wspólnego mianownika:

$$S(k+1) = \frac{k(k+1) + 2(k+1)}{2}$$

Wyłącz (k+1) jako wspólny czynnik:

$$S(k+1) = \frac{(k+1)(k+2)}{2}$$

Dowód zakończony.

4. Relacje. Relacje równoważności oraz klasy abstrakcji. Relacje porządku.

1. Relacja zwrotna

Relacja $\rho \subseteq A \times A$ jest zwrotna, jeśli każdy element zbioru A jest w relacji z samym sobą:

$\forall a \in A$,

Przykład:
Zbiór A={1,2,3}. Relacja „równość” (=) jest zwrotna, bo każdy element jest równy samemu sobie:

- 1=1,
- 2=2,
- 3=3

2. Relacja przeciwzwrotna

Relacja $\rho \subseteq A \times A$ jest przeciwzwrotna, jeśli żaden element zbioru A nie jest w relacji z samym sobą:

$\forall a \in A,\ \neg(a \rho a)$

Przykład:
Zbiór A={1,2,3} Relacja „większe” (>) jest przeciwzwrotna, bo żadna liczba nie jest większa od samej siebie:

- 1>1– fałsz,
- 2>2– fałsz,
- 3>3– fałsz.

3. Relacja symetryczna

Relacja $\rho \subseteq A \times A$ jest symetryczna, jeśli gdy a jest w relacji z b, to b jest w relacji z a:

$\forall a,b \in A,\ (a \rho b \implies b \rho a)$

Przykład:
na zbiorze A={1,2,3} jest symetryczna:

- Jeśli 1=1 to 1 = 1
- Jeśli 2=2 to 2 = 2.

4. Relacja przeciwsymetryczna

Relacja $\rho \subseteq A \times A$ jest przeciwsymetryczna, jeśli gdy a jest w relacji z b, to b nie jest w relacji z a:

$\forall a,b \in A,\ (a \rho b \implies \neg(b \rho a))$

Przykład:
Zbiór A={1,2,3}. Relacja „bycia starszym” (starszy od\text{starszy od}starszy od) jest przeciwsymetryczna:

- Jeśli osoba a jest starsza od osoby b, to osoba b nie może być starsza od osoby a.

A={1,2,3} jest przeciwsymetryczna:

Jeśli 2>1to 1 (not >) 2

5. Relacja antysymetryczna

Relacja $\rho \subseteq A \times A$ jest antysymetryczna, jeśli gdy a jest w relacji z b, a b jest w relacji z a, to a i b muszą być równe:

$\forall a,b \in A, (a \rho b \wedge b \rho a \implies a=b)$.

Przykład:
Zbiór A={1,2,3}. Relacja jest antysymetryczna:

- Jeśli a≤b i b≤a to a=b

  zbiorze A={1,2,3} jest antysymetryczna:

- Jeśli 1≤2 i 2≤1, to 1=2 (co jest niemożliwe w tym przypadku).
- Jeśli 2≤2, to 2=2

6. Relacja przechodnia

Relacja ρ⊆A×A jest przechodnia, jeśli gdy a jest w relacji z b, a b jest w relacji z c, to a jest w relacji z c:

∀a,b,c∈A,(aρb∧bρc⟹aρc).

Przykład:
Zbiór A={1,2,3} Relacja jest przechodnia:

- Jeśli 1<2 i 2<3 , to 1<3

Relacje równoważności
Relacja R na zbiorze A jest równoważnością, jeśli spełnia:

1. Zwrotność: ∀x∈A,(xRx)
2. Symetryczność: ∀x,y∈A,(xRy ⟹ yRx)
3. Przechodniość: ∀x,y,z∈A,(xRy∧yRz ⟹ xRz)
4. Każda relacja równoważności dzieli zbiór A na klasy abstrakcji (zbiory elementów równoważnych).

Relacje porządku
Relacja ≤ jest porządkiem, jeśli jest:

1. Zwrotna,
2. Przechodnia,
3. Antysymetryczna: ∀x,y∈A,(x≤y∧y≤x ⟹ x=y)

Porządek może być:

- **Częściowy**: np. podzbiory zbioru P(A)
- **Liniowy**: każdy element jest porównywalny.

5. Funkcja jako relacja - własności funkcji (injekcja, surjekcja, bijekcja, składanie funkcji)

Funkcja to szczególny przypadek relacji f:A→Bf: A \to Bf:A→B, gdzie każdy element a∈Aa \in Aa∈A jest związany z dokładnie jednym b∈Bb \in Bb∈B.

Własności funkcji:

- **Injekcja (jednoznaczność)**: Funkcja f jest różnowartościowa, jeśli $f(a_1)=f(a_2) \implies a_1=a_2$
- **Surjekcja (na)**: Funkcja f jest "na", jeśli $\forall b \in B, \exists a \in A: f(a)=b$
- **Bijekcja**: Funkcja jest zarówno injekcją, jak i surjekcją.

Składanie funkcji: Jeśli $f:A \to B$ i $g:B \to C$ to $g \circ f: A \to C$ jest ich składaniem.

1. Przesunięcia

- **W poziomie**:
  - Funkcja $f(x-c)$: przesunięcie w prawo o c.
  - Funkcja $f(x+c)$: przesunięcie w lewo o c.
- **W pionie**:
  - Funkcja $f(x)+d$: przesunięcie w górę o d.
  - Funkcja $f(x)-d$: przesunięcie w dół o d.

Przykład:
$f(x)=x^2$, $g(x)=(x-2)^2+3$ to f(x) przesunięta w prawo o 2 i w górę o 3.

2. Skalowanie

- **W poziomie**:
  - Funkcja $f(kx)$: kompresja w poziomie (jeśli $|k|>1$) lub rozciągnięcie (jeśli $0<|k|<1$).
- **W pionie**:
  - Funkcja $a*f(x)$: rozciągnięcie w pionie (jeśli $|a|>1$) lub kompresja (jeśli $0<|a|<1$).

Przykład:
$f(x)=x^2$, $g(x)=3x^2$ to f(x) rozciągnięta w pionie trzykrotnie.

3. Odbicia

- **Względem osi x**:
  - Funkcja $-f(x)$: odbicie wykresu funkcji względem osi x.
- **Względem osi y**:
  - Funkcja $f(-x)$: odbicie wykresu funkcji względem osi y.

Przykład:
$f(x)=x^2$, $g(x)=-x^2$ to f(x) odbita względem osi x.

4. Złożone przekształcenia

Przekształcenia można łączyć, wykonując je w odpowiedniej kolejności. Kolejność zwykle jest następująca:

1. Skalowanie,
2. Odbicie,
3. Przesunięcie.

Przykład:
Funkcja $f(x)=x^2$, przekształcenie $g(x)=-2(x-1)^2+3$:

1. Rozciągnięcie w pionie przez współczynnik 2,
2. Odbicie względem osi x,
3. Przesunięcie w prawo o 1,
4. Przesunięcie w górę o 3.

5. Przekształcenia osiowe

- Funkcja h(x)=f(|x|): zachowuje prawą stronę wykresu, a lewa jest lustrzanym odbiciem.
- Funkcja h(x)=|f(x)|: wszystkie ujemne wartości funkcji są zastąpione wartościami dodatnimi (odbicie części pod osią x).

1. Definicja składania funkcji

Jeśli mamy dwie funkcje:
$f:A\to B$ i $g:B\to C$
to ich złożenie $g\circ f$ (czytane jako „g po f") jest funkcją, która przypisuje każdemu $x\in A$.
Formalnie:

$(g \circ f)(x)=g(f(x))$dla każdego $x\in A$

2. Składanie więcej niż dwóch funkcji

Składanie może być rozszerzone na więcej funkcji, np. h, g, i f:

$(h \circ g \circ f)(x)=h(g(f(x)))$

Tutaj, działamy w następującej kolejności:

1. Najpierw obliczamy f(x),
2. Następnie g(f(x)),
3. Na końcu h(g(f(x))).

3. Przykład z trzema funkcjami na liczbach

Funkcje:

- $f(x)=x+2f(x) = x + 2$,
- $g(x)=2xg(x) = 2x$,
- $h(x)=x2h(x) = x^2$.

Złożenie $h \circ g \circ f$:

$(h \circ g \circ f)(x)=h(g(f(x)))$.

1. Najpierw $f(x)=x+2$,
2. Następnie $g(f(x))=2(x+2)=2x+4$,
3. Na końcu $h(g(f(x)))=(2x+4)^2$.

Ostateczna funkcja:

$(h \circ g \circ f)(x)=(2x+4)^2$.

4. Właściwości składania funkcji

1. **Nieprzemienność:**
   W ogólności składanie funkcji nie jest przemienne, czyli:
   $f \circ g \neq g \circ f$.
   Przykład: Jeśli $f(x)=x+2$ i $g(x)=2x$,
   - $(f \circ g)(x)=f(g(x))=(2x)+2=2x+2$,
   - $(g \circ f)(x)=g(f(x))=2(x+2)=2x+4$.
     Wyniki są różne.

6. Kombinatoryka i rachunek prawdopodobieństwa. Zmienna losowa.

Kombinatoryka
Zajmuje się liczeniem liczby sposobów wyboru lub ułożenia elementów:

- **Permutacje**: n! liczba ułożeń n elementów,
- **Kombinacje**: $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, liczba sposobów wyboru k elementów z n,
- **Z wariacjami**: Gdy uwzględnia się kolejność, np. n^k

Rachunek prawdopodobieństwa
Opisuje zjawiska losowe:

- Prawdopodobieństwo zdarzenia A: $P(A)=|A|/|\Omega|$ (przy zdarzeniach równoprawdopodobnych),
- **Zasada dodawania**: $P(A \cup B)=P(A)+P(B)-P(A \cap B)$

Zmienna losowa
Zmienna losowa to funkcja $X:\Omega \rightarrow R$ która przypisuje liczby zdarzeniom elementarnym.

- Dyskretna: np. liczba wyrzuconych oczek na kostce,
- Ciągła: np. czas oczekiwania na autobus.

Rozkład zmiennej losowej określa prawdopodobieństwa przyjmowanych wartości, np.:

- Dyskretna: $P(X=x)$
- Ciągła: gęstość $f_X(x)$ i prawdopodobieństwo $P(a \leq X \leq b)$

Języki Hipertekstowe i tworzenie stron WWW

1. Typy selektorów w CSS

Selektory w CSS służą do określania, które elementy HTML mają być stylizowane. Istnieje wiele typów selektorów, z których najważniejsze to:

- **Selektory elementów (tagów)** – wybierają wszystkie elementy o określonym tagu:

```
p {
  color: red;
}
```

Stylizuje wszystkie elementy <p> na czerwono.

Selektor nagłówków (h)

- **Selektory klas** – wybierają elementy z określoną klasą:

```
.class-name {
  font-size: 16px;
}
```

Stylizuje wszystkie elementy z klasą class-name.

- **Selektory identyfikatorów (id)** – wybierają pojedynczy element z określonym id:

```
#element-id {
  background-color: blue;
}
```

Stylizuje element o id="element-id" na niebiesko.

- **Selektory atrybutów** – wybierają elementy na podstawie atrybutów:

```
input[type="text"] {
  border: 1px solid black;
}
```

Stylizuje wszystkie elementy <input> z atrybutem type="text".

- **Selektory potomków (descendant)** – wybierają elementy będące potomkami innego elementu:

```
div p {
  color: green;
}
```

Stylizuje wszystkie elementy <p>, które są wewnątrz elementów .

- **Selektory dzieci (child)** – wybierają elementy, które są bezpośrednimi dziećmi innego elementu:

```
ul > li {
  list-style-type: square;
}
```

Stylizuje tylko bezpośrednie dzieci <li> wewnątrz <ul>.

- **Selektory rodzeństwa (sibling)** – wybierają elementy, które są rodzeństwem innych elementów:

```
h1 + p {
  color: orange;
}
```

Stylizuje pierwszy element <p> po każdym <h1>.

2. Sposoby pozycjonowania elementów w dokumencie HTML

- **Pozycjonowanie statyczne (default)** – to domyślne ustawienie, w którym elementy są pozycjonowane zgodnie z normalnym przepływem dokumentu:

```
div {
  position: static;
}
```

- **Pozycjonowanie względne** – element jest pozycjonowany w stosunku do swojej normalnej pozycji w dokumencie:

```
div {
  position: relative;
  top: 10px;
}
```

- **Pozycjonowanie absolutne** – element jest pozycjonowany względem najbliższego elementu o pozycjonowaniu innym niż static (zwykle jest to element z position: relative):

```
div {
  position: absolute;
  top: 50px;
  left: 100px;
}
```

- **Pozycjonowanie stałe (fixed)** – element jest pozycjonowany względem okna przeglądarki (pozostaje w tej samej pozycji nawet podczas przewijania strony):

```
div {
  position: fixed;
  top: 10px;
  right: 0;
}
```

- **Pozycjonowanie przyklejone (sticky)** – element jest pozycjonowany jak relative do momentu, aż przewiniemy go poza jego kontener; wtedy staje się fixed:

```
div {
  position: sticky;
  top: 0;
}
```

3. Podstawowe konstrukcje w języku JS

- **Zmienne** – w JavaScript zmienne mogą być deklarowane przy użyciu var, let lub const. let pozwala na modyfikowanie zmiennej, a const deklaruje stałą.

```
let x = 5; (widocny tylko w obrębię bloku a var widoczny w obrębie całości)
const y = 10;
```

- **Operatory** – JavaScript obsługuje wiele operatorów, takich jak:
  - Operator przypisania (=)
  - Operatory arytmetyczne: +, -, *, /, %
  - Operatory porównania: ==, ===, !=, !==, >, <, >=, <=
  - Operatory logiczne: && (AND), || (OR), ! (NOT)

```
let a = 10;
let b = 5;
let sum = a + b; // 15
```

- **Instrukcje warunkowe** – umożliwiają wykonywanie różnych działań w zależności od spełnienia warunków.

```
let age = 18;
if (age >= 18) {
  console.log("Jesteś dorosły");
} else {
  console.log("Jesteś nieletni");
}
```

- **Pętle** – umożliwiają wielokrotne wykonanie tego samego kodu.
  - **Pętla for**:

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

  - **Pętla while**:

```
let i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```

4. Semantyka w HTML5

Semantyczne znaczenie tagów HTML5 polega na tym, że elementy HTML mają przypisaną konkretną rolę, która wpływa na zrozumienie treści dokumentu zarówno przez przeglądarki, jak i roboty wyszukiwarek. Do semantycznych tagów w HTML5 należą m.in.:

- **<header>** – definiuje nagłówek strony lub sekcji.
- **<footer>** – definiuje stopkę strony lub sekcji.

- **<article>** – definiuje samodzielną jednostkę treści.
- **<section>** – definiuje sekcję tematyczną w dokumencie.
- **<nav>** – zawiera linki nawigacyjne.
- **<aside>** – definiuje treść poboczną.
- **<main>** – określa główną treść dokumentu.

Semantyczne znaczenie pomaga poprawić dostępność i optymalizację pod kątem SEO.

5. Metoda POST i GET przekazywania danych w formularzach

- **GET** – dane są przesyłane w adresie URL. Metoda ta jest używana, gdy dane mają charakter publiczny (np. wyszukiwanie). Ogranicza ilość przesyłanych danych.

```
<form method="GET" action="/search">
  <input type="text" name="query">
  <button type="submit">Szukaj</button>
</form>
```

- **POST** – dane są przesyłane w ciele żądania HTTP, co pozwala na przesyłanie większej ilości danych i jest bezpieczniejsze (dane nie są widoczne w URL).

```
<form method="POST" action="/submit">
  <input type="text" name="username">
  <button type="submit">Zarejestruj</button>
</form>
```

6. CSS – dziedziczenie własności

W CSS niektóre właściwości są dziedziczone przez elementy potomne, inne nie. Na przykład, właściwości takie jak color, font-family i line-height są dziedziczone, natomiast margin, padding czy background – nie. Można używać słowa kluczowego inherit, aby jawnie wskazać, że dana właściwość powinna być dziedziczona.

```
p {
  color: red; /* Dziedziczone przez dzieci */
}

div {
  background-color: blue; /* Nie dziedziczone przez dzieci */
}
```

7. Drzewo dokumentu HTML i selektory odwołujące się do jego poszczególnych elementów

- **Drzewo dokumentu HTML** to struktura hierarchiczna, w której elementy są zagnieżdżone w sobie, tworząc relacje rodzic–dziecko. Każdy element w drzewie może mieć inne elementy wewnętrzne (potomki) oraz sąsiednie elementy (rodzeństwo).

- **Selektory potomków (descendant)** – wybierają wszystkie elementy wewnątrz innego elementu:

```
div p { /* Wybiera <p> wewnątrz <div> */
  color: blue;
}
```

- **Selektory dzieci (child)** – wybierają tylko bezpośrednie dzieci:

```
div > p { /* Wybiera tylko bezpośrednie dzieci <p> w <div> */
  color: red;
}
```

- **Selektory rodzeństwa (sibling)** – wybierają elementy, które znajdują się obok siebie w hierarchii:

```
h1 + p { /* Wybiera pierwszy <p> po <h1> */
  color: green;
}
```

- **Selektory wszystkich rodzeństw**:

```
h1 ~ p { /* Wybiera wszystkie <p> po <h1> */
  color: yellow;
}
```

Każdy z tych selektorów pozwala na dokładne dopasowanie stylów do określonych elementów w strukturze HTML.

Programowanie Proceduralne

1. Wskaźniki i operacje na wskaźnikach w języku C

Wskaźnik w języku C to zmienna, która przechowuje adres innej zmiennej w pamięci. Operacje na wskaźnikach obejmują:

- **Deklaracja wskaźnika**: Wskaźnik deklarujemy używając operatora *:

  ```
  int *ptr; // wskaźnik na zmienną typu int
  ```

- **Inicjalizacja wskaźnika**: Wskaźnik może być inicjalizowany za pomocą adresu zmiennej:

  ```
  int a = 10;
  int *ptr = &a; // ptr przechowuje adres zmiennej a

  int arr[3] = {10, 20, 30};
  int *ptr = arr;
  printf("%d", ptr[1]); // Wypisze 20
  ```

- **Dereferencja wskaźnika**: Aby uzyskać wartość przechowywaną pod adresem, używamy operatora *:

  int value = *ptr; // Zmienna value przyjmuje wartość 10

- **Operacje arytmetyczne na wskaźnikach**: Można wykonywać operacje arytmetyczne na wskaźnikach (np. inkrementacja, dekrementacja):

  ptr++; // Przesuwa wskaźnik do następnej komórki pamięci (w przypadku typu int – o 4 bajty)

  ```
  int arr[] = {10, 20, 30};
  int *ptr = arr;
  ptr++; // Wskaźnik wskazuje teraz na drugi element tablicy
  printf("%d", *ptr); // Wypisze 20
  ```

- **Porównanie wskaźników**: Można porównywać wskaźniki:

  ```
  if (ptr == &a)
   { // Sprawdza, czy wskaźnik ptr wskazuje na zmienną a
     // ...
  }
  ```

Różnica wskaźników:
Jeśli dwa wskaźniki wskazują na elementy tej samej tablicy, można obliczyć odległość między nimi.

```
int arr[] = {10, 20, 30};
int *ptr1 = &arr[0];
int *ptr2 = &arr[2];
printf("%ld", ptr2 - ptr1);  // Wypisze 2
```

2. Wskaźnik podwójny, tablice wskaźników w języku C

- **Wskaźnik podwójny**: Wskaźnik podwójny (wskaźnik na wskaźnik) przechowuje adres wskaźnika. Może być używany, gdy musimy modyfikować wskaźniki w funkcji (np. alokować pamięć dynamicznie):

  ```
  int a = 10;
  int *ptr = &a;     // wskaźnik na a
  int **ptr2 = &ptr; // wskaźnik na wskaźnik ptr
  ```

- **Tablica wskaźników**: Tablica wskaźników jest tablicą, gdzie każdy element jest wskaźnikiem do jakiegoś typu danych. Na przykład:

  int *arr[10]; // Tablica wskaźników na 10 elementów typu int

Każdy element arr[i] jest wskaźnikiem do zmiennej typu int.

a) Tablica dwuwymiarowa jako pojedynczy blok pamięci

W tym podejściu tworzymy tablicę jako jeden blok pamięci i traktujemy ją jako dwuwymiarową. To podejście jest bardziej efektywne, ponieważ cała pamięć jest alokowana w jednym kawałku. Wskaźniki wskazują na odpowiednie elementy tej tablicy.

Przykład:

```
int rows = 3, cols = 4;
int *arr = (int *)malloc(rows * cols * sizeof(int));  // Alokacja pamięci na 3 wiersze i 4 kolumny

// Przypisywanie wartości do elementów tablicy
arr[0 * cols + 0] = 10;  // arr[0][0]
arr[0 * cols + 1] = 20;  // arr[0][1]
arr[1 * cols + 0] = 30;  // arr[1][0]

// Dostęp do elementów tablicy
int value = arr[1 * cols + 2];  // arr[1][2] - dostęp do elementu w 2. wierszu i 3. kolumnie

// Zwolnienie pamięci
free(arr);
```

W tym przypadku traktujemy tablicę dwuwymiarową jako jednowymiarową (wskaźnik do pierwszego elementu) i używamy indeksu i * cols + j do uzyskania odpowiednich elementów wiersza i kolumny. To podejście pozwala na swobodne zarządzanie pamięcią.

b) Tablica dwuwymiarowa jako tablica wskaźników do tablic

To podejście jest bardziej intuicyjne, ale pamięć jest alokowana na każdą tablicę wiersza z osobna. Każdy wskaźnik w tablicy głównej wskazuje na jeden wiersz tablicy.

Przykład:

```
int rows = 3, cols = 4;
int **arr = (int **)malloc(rows * sizeof(int *));  // Alokacja wskaźników do 3 wierszy

// Alokacja pamięci dla każdego wiersza
for (int i = 0; i < rows; i++) {
   arr[i] = (int *)malloc(cols * sizeof(int));  // Każdy wiersz to tablica 4 elementów
}

// Przypisywanie wartości do elementów tablicy
arr[0][0] = 10;  // arr[0][0]
arr[1][2] = 20;  // arr[1][2]
arr[2][3] = 30;  // arr[2][3]

// Zwolnienie pamięci
for (int i = 0; i < rows; i++) {
```

```
    free(arr[i]);  // Zwolnienie pamięci dla każdego wiersza
}
free(arr);  // Zwolnienie wskaźników na wiersze
```

W tym przypadku mamy wskaźnik do wskaźników (int **arr), gdzie arr[i] jest wskaźnikiem na tablicę, którą traktujemy jako wiersz w tablicy dwuwymiarowej. To podejście daje więcej elastyczności, pozwala na tworzenie tablic o różnych rozmiarach w każdym wierszu (tzw. "tablica nieregularna").

- **Zwalnianiu pamięci**: Po zakończeniu pracy z dynamicznie przydzieloną pamięcią, należy ją zwolnić, aby uniknąć wycieków pamięci. Używamy funkcji free().
- **Zarządzanie wskaźnikami**:
    - Wskaźnik na tablicę dwuwymiarową, jak w przykładzie int **arr, wymaga zwolnienia pamięci dla każdego wiersza (ponieważ każdy wiersz jest osobną tablicą), a potem zwolnienia pamięci dla tablicy wskaźników.
    - W przypadku alokacji jako pojedynczego bloku pamięci (gdzie traktujemy tablicę jako jednowymiarową), pamięć zwalniamy po zakończeniu używania jej w jednym wywołaniu free().

4. Zarządzanie pamięcią dla tablic o nieregularnych rozmiarach

Jeśli wiersze tablicy mają różną długość (np. w przypadku "tablic nieregularnych"), należy zarządzać pamięcią dla każdego wiersza indywidualnie. W takim przypadku, przy alokacji pamięci dla każdego wiersza oddzielnie, zachowujemy elastyczność, ale trzeba dokładnie śledzić, gdzie każda część pamięci została alokowana i zwolnić ją osobno.

3. Dynamiczna alokacja pamięci w języku C. Wycieki pamięci

Dynamiczna alokacja pamięci pozwala na przydzielanie pamięci w czasie wykonywania programu. W C używamy funkcji z biblioteki standardowej (stdlib.h) do alokowania i zwalniania pamięci:

- **Alokacja pamięci**:
    - malloc(size_t size) – przydziela pamięć o rozmiarze size (w bajtach), ale nie inicjalizuje jej wartości.
    - calloc(size_t num, size_t size) – przydziela pamięć dla num elementów, każdy o rozmiarze size, i inicjalizuje ją na 0.
    - realloc(void *ptr, size_t size) – zmienia rozmiar wcześniej przydzielonej pamięci.

    Przykład:

    ```
    int *arr = (int *)malloc(5 * sizeof(int)); // Alokacja pamięci na tablicę 5 elementów typu int
    ```

- **Zwalnianie pamięci**: Po zakończeniu używania pamięci dynamicznej należy ją zwolnić, aby zapobiec wyciekowi pamięci:

    ```
    free(arr); // Zwalnia przydzieloną pamięć
    ```

- **Wycieki pamięci**: Wycieki pamięci występują, gdy przydzielona pamięć nie zostaje zwolniona, co prowadzi do utraty zasobów systemowych. Należy pamiętać o wywołaniu free() po zakończeniu używania dynamicznie alokowanej pamięci.
- **Zarządzanie wskaźnikami**:
    - Wskaźnik na tablicę dwuwymiarową, jak w przykładzie int **arr, wymaga zwolnienia pamięci dla każdego wiersza (ponieważ każdy wiersz jest osobną tablicą), a potem zwolnienia pamięci dla tablicy wskaźników.
    - W przypadku alokacji jako pojedynczego bloku pamięci (gdzie traktujemy tablicę jako jednowymiarową), pamięć zwalniamy po zakończeniu używania jej w jednym wywołaniu free().

4. Łańcuchy znakowe w języku C

Łańcuchy znakowe w C są reprezentowane jako tablice znaków zakończone specjalnym znakiem '\0' (null character), który sygnalizuje koniec łańcucha:

```
char str[] = "Hello, World!"; // Tablica znaków zakończona '\0'
```

Operacje na łańcuchach znakowych:

- **Funkcje standardowe**:
    - strlen(str) – zwraca długość łańcucha znakowego.
    - strcpy(dest, src) – kopiuje łańcuch z src do dest.
    - strcmp(str1, str2) – porównuje dwa łańcuchy znakowe.
    - strcat(dest, src) – łączy dwa łańcuchy.

Przykład:

```
char str1[50], str2[] = "World";
strcpy(str1, "Hello, ");
strcat(str1, str2); // str1 = "Hello, World"
```

5. Rekurencja w języku C

Rekurencja to technika, w której funkcja wywołuje samą siebie. Zwykle wykorzystywana jest w problemach, które mogą być rozwiązane w sposób podzielny na mniejsze, podobne problemy.

Przykład rekurencyjnej funkcji obliczającej silnię:

```
int factorial(int n) {
   if (n == 0) return 1;  // warunek zakończenia
   return n * factorial(n - 1); // rekurencyjne wywołanie
}
```

Rekurencja powinna mieć warunek zakończenia, w przeciwnym razie może prowadzić do nieskończonego wywoływania funkcji.

6. Pliki w języku C. Pliki tekstowe i binarne. Dostęp sekwencyjny i swobodny do pliku

W C operacje na plikach są realizowane przez funkcje z biblioteki standardowej (stdio.h).

- **Otwarcie pliku**: Można otworzyć plik do różnych operacji za pomocą funkcji fopen(). Do najczęściej używanych trybów otwierania pliku należą:
  - "r" – otwiera plik do odczytu.
  - "w" – otwiera plik do zapisu, tworzy nowy plik.
  - "a" – otwiera plik do dopisywania.
  - "rb", "wb", „ab” – otwiera plik binarny.

  Przykład:

  ```
  FILE *file = fopen("example.txt", "r");
  ```

- **Dostęp sekwencyjny**: Dane w pliku są odczytywane lub zapisywane w kolejności, w jakiej się znajdują. Używamy funkcji takich jak fgetc(), fputc(), fgets(), fputs().
- **Dostęp swobodny (losowy)**: Pozwala na odczyt lub zapis do dowolnego miejsca w pliku. Używa się funkcji fseek(), ftell() i fread().
- **Zamykanie pliku**: Po zakończeniu operacji na pliku należy go zamknąć:

  ```
  fclose(file);
  ```

Cechy	Pliki tekstowe	Pliki binarne
Czytelność	Czytelne dla człowieka	Nieczytelne dla człowieka
Wielkość pliku	Większe (formatowanie)	Mniejsze (brak formatowania)
Wydajność	Wolniejsze	Szybsze
Złożoność przetwarzania	Prostsze do przetwarzania manualnie	Trudniejsze w interpretacji
Zastosowanie	Przechowywanie tekstu, konfiguracji	Dane binarne, multimedia

Odczyt/zapis do pliku tekstowego:

- `fprintf` / `fscanf` dla formatowanego odczytu/zapisu.
- `fgets` / `fputs` dla linii tekstu.

Odczyt/zapis do pliku binarnego:

- `fwrite` / `fread` dla zapisu/odczytu danych binarnych.

Dostęp swobodny

- Możliwy dzięki funkcjom pozwalającym na przesunięcie wskaźnika pliku na dowolną pozycję:
    - `fseek`: Ustawia wskaźnik pliku na określoną pozycję.
    - `ftell`: Pobiera aktualną pozycję wskaźnika pliku.
    - `rewind`: Ustawia wskaźnik na początek pliku.

Tryby `fseek`:

- `SEEK_SET`: Od początku pliku.
- `SEEK_CUR`: Od bieżącej pozycji.
- `SEEK_END`: Od końca pliku.

Funkcje do otwierania i zamykania plików

1. **`fopen`**
    - Otwiera plik w określonym trybie (np. do odczytu, zapisu, dopisywania).
    - Zwraca wskaźnik do struktury `FILE` lub `NULL` w przypadku błędu.
2. **`fclose`**
    - Zamyka wcześniej otwarty plik i zwalnia zasoby.
    - Zwraca `0` przy powodzeniu, a wartość różną od zera w przypadku błędu.

Funkcje do odczytu z plików

1. **`fgetc`**
    - Odczytuje jeden znak z pliku.
    - Zwraca odczytany znak lub `EOF`, jeśli osiągnięto koniec pliku.
2. **`fgets`**
    - Odczytuje linię tekstu z pliku (do napotkania nowej linii lub limitu znaków).
    - Zwraca wskaźnik do odczytanego ciągu lub `NULL`, jeśli osiągnięto koniec pliku.
3. **`fscanf`**
    - Odczytuje dane sformatowane z pliku (działa podobnie do `scanf`).
    - Zwraca liczbę poprawnie odczytanych elementów.
4. **`fread`**
    - Odczytuje dane binarne z pliku do bufora.
    - Przyjmuje wskaźnik na bufor, rozmiar elementu, liczbę elementów i wskaźnik do pliku.

Funkcje do zapisu do plików

1. **`fputc`**
    - Zapisuje jeden znak do pliku.
    - Zwraca zapisany znak lub `EOF` w przypadku błędu.
2. **`fputs`**
    - Zapisuje ciąg znaków (do napotkania `\0`) do pliku.
    - Zwraca wartość różną od zera w przypadku błędu.
3. **`fprintf`**
    - Zapisuje dane sformatowane do pliku (działa podobnie do `printf`).
    - Zwraca liczbę zapisanych znaków lub wartość ujemną w przypadku błędu.
4. **`fwrite`**

- Zapisuje dane binarne z pamięci do pliku.
- Przyjmuje wskaźnik na dane, rozmiar elementu, liczbę elementów i wskaźnik do pliku.

Funkcje do zarządzania wskaźnikiem pliku

1. **`fseek`**
   - Ustawia wskaźnik pliku na określoną pozycję.
   - Przyjmuje offset i punkt odniesienia (`SEEK_SET`, `SEEK_CUR`, `SEEK_END`).
2. **`ftell`**
   - Zwraca bieżącą pozycję wskaźnika w pliku jako wartość liczbową.
   - Przydatne do ustalania miejsca w pliku.
3. **`rewind`**
   - Ustawia wskaźnik pliku na początek pliku.
   - Nie zwraca wartości.

Funkcje diagnostyczne

1. **`feof`**
   - Sprawdza, czy osiągnięto koniec pliku.
   - Zwraca wartość różną od zera, jeśli wskaźnik osiągnął koniec pliku.
2. **`ferror`**
   - Sprawdza, czy podczas operacji na pliku wystąpił błąd.
   - Zwraca wartość różną od zera, jeśli wystąpił błąd.

7. Wskaźniki do funkcji

Wskaźniki do funkcji pozwalają na przechowywanie adresów funkcji i ich wywoływanie w sposób dynamiczny. Wskaźnik do funkcji jest deklarowany w następujący sposób:

```
return_type (*ptr)(arg_types);  // wskaźnik do funkcji
```

Przykład:

```
#include <stdio.h>

void greet() {
   printf("Hello, world!\n");
}

int main() {
   void (*func_ptr)() = greet; // wskaźnik do funkcji greet
   func_ptr();  // wywołanie funkcji przez wskaźnik
   return 0;
}


void update(int *ptr) {
   *ptr = 20;  // Zmiana wartości zmiennej wskazywanej przez wskaźnik
}
int main() {
   int x = 10;
```

```
    update(&x);
    printf("%d", x);  // Wypisze 20
    return 0;
}
```

8. Struktura programu w języku C. Klasy zmiennych. Czas trwania zmiennych, zasięg i łączność

Struktura programu w języku C

Program w języku C składa się z kilku podstawowych elementów:

1. **Funkcja main()** - jest to główna funkcja, od której zaczyna się wykonanie programu.
2. **Deklaracje zmiennych** - zmienne muszą być zadeklarowane przed ich użyciem, aby określić ich typ i rozmiar.
3. **Instrukcje** - operacje wykonywane w ramach funkcji.
4. **Wywołania funkcji** - funkcje mogą być wywoływane w innych funkcjach, w tym w main().

Oto przykładowa struktura programu w C:

```
#include <stdio.h>

int zmienna_globalna = 10; // zmienna globalna

void funkcja() {
    int zmienna_lokalna = 5;  // zmienna lokalna
    printf("Zmienna lokalna: %d\n", zmienna_lokalna);
    printf("Zmienna globalna: %d\n", zmienna_globalna);
}

int main() {
    funkcja();  // wywołanie funkcji
    return 0;
}
```

Rodzaje zmiennych w języku C

1. **Zmienne lokalne**
    - Zadeklarowane wewnątrz funkcji, dostępne tylko w tej funkcji.
    - Przykład:

      ```
      void funkcja() {
          int zmienna_lokalna = 10;  // zmienna lokalna
          printf("%d\n", zmienna_lokalna);  // dostępna tylko w funkcji
      }
      ```

2. **Zmienne globalne**
    - Zadeklarowane poza funkcjami, dostępne w całym programie.
    - Przykład:

```
int zmienna_globalna = 20;  // zmienna globalna

void funkcja() {
   printf("Zmienne globalna: %d\n", zmienna_globalna);  // dostępna w każdej funkcji
}

int main() {
   funkcja();  // może korzystać ze zmiennej globalnej
   return 0;
}
```

3. **Zmienne statyczne**
   - Zachowują swoją wartość między wywołaniami funkcji. Są użyteczne, gdy potrzebujemy, aby zmienna nie była usuwana po zakończeniu funkcji, ale także nie była dostępna na zewnątrz funkcji.
   - Przykład:

```
void funkcja() {
   static int zmienna_statystyczna = 0;  // zmienna statyczna
   zmienna_statystyczna++;
   printf("Zmienne statyczna: %d\n", zmienna_statystyczna);
}

int main() {
   funkcja();  // wywołanie funkcji po raz pierwszy
   funkcja();  // wywołanie funkcji po raz drugi
   return 0;
}
```

   Wynik:

```
Zmienna statyczna: 1
Zmienna statyczna: 2
```

4. **Zmienne automatyczne**
   - Tworzone na stosie, usuwane po zakończeniu funkcji. Są to domyślnie zmienne lokalne w C, które przechowują swoje wartości tylko w obrębie funkcji.
   - Przykład:

```
void funkcja() {
   int zmienna_automatyczna = 10;  // zmienna automatyczna
   printf("Zmienne automatyczna: %d\n", zmienna_automatyczna);
}

int main() {
   funkcja();
   // zmienna_automatyczna nie jest dostępna tutaj, po zakończeniu funkcji
   return 0;
}
```

Czas trwania zmiennych

- **Zmienne lokalne** są tworzone na początku funkcji i usuwane po jej zakończeniu.
- **Zmienne globalne i statyczne** istnieją przez cały czas trwania programu. Zmienne globalne są dostępne w każdej funkcji, natomiast zmienne statyczne przechowują swoją wartość między wywołaniami funkcji, ale są dostępne tylko wewnątrz tej funkcji.

Zasięg zmiennych

1. **Zasięg lokalny** - zmienna jest dostępna tylko w funkcji, w której została zadeklarowana.
    - Zmienna lokalna ma zasięg tylko w funkcji, w której została zadeklarowana. Nie jest dostępna w innych funkcjach.
    - Przykład:

```
void funkcja1() {
   int zmienna1 = 5;  // zmienna dostępna tylko w funkcji1
}

void funkcja2() {
   // zmienna1 nie jest dostępna w tej funkcji
}
```

2. **Zasięg globalny** - zmienna jest dostępna w całym programie, we wszystkich funkcjach, które ją wykorzystują.
    - Zmienna globalna jest zadeklarowana poza wszystkimi funkcjami, na początku programu.
    - Przykład:

```
int zmienna_globalna = 30;  // zmienna dostępna w każdej funkcji

void funkcja1() {
   printf("%d\n", zmienna_globalna);  // dostęp do zmiennej globalnej
}

void funkcja2() {
   printf("%d\n", zmienna_globalna);  // dostęp do zmiennej globalnej
}

int main() {
   funkcja1();
   funkcja2();
   return 0;
}
```

Łączność zmiennych

Zmienne mogą być połączone z różnymi funkcjami, a także mogą korzystać z **wskaźników**, aby przekazywać wartości między funkcjami. Wskaźniki umożliwiają przekazywanie adresu zmiennej, dzięki czemu funkcje mogą modyfikować jej wartość bez konieczności przekazywania jej kopii.

Przykład użycia wskaźników:

```
#include <stdio.h>

void funkcja(int *x) {
   *x = 20;  // modyfikacja zmiennej, na którą wskazuje wskaźnik
}

int main() {
   int zmienna = 10;
   printf("Przed: %d\n", zmienna);
   funkcja(&zmienna);  // przekazanie adresu zmiennej
   printf("Po: %d\n", zmienna);  // zmienna została zmieniona przez funkcję
   return 0;
}
```

Wynik:

```
Przed: 10
Po: 20
```

By Hubertus Bubertus