Numer indeksu …….

1. (8 pkt) Uzupełnić tabelę adresacji:

Przykładowy adres IP	**Maska podsieci**	**Brama domyślna**	**Adres rozgłoszeniowy**	**Liczba adresów**
192.168.50.100	/25	192.168.50.1	192.168.50.127	127
170.20.55.[2-14]	/28	170.20.55.1	170.20.55.15	15
10.[0-255]. [0-255]. [0-254/255]	/8	10.0.0.1	10.255.255.255	2^24 - 1
192.168.1.[2-62]	/26	192.168.1.1	192.168.1.63	63

2. (2 pkt) Uzupełnić stos TCP/IP :

Warstwa aplikacji
Warstwa prezentacji
Warstwa sesji
Warstwa transportowa
Warstwa sieci
Warstwa ł. danych
Warstwa fizyczna

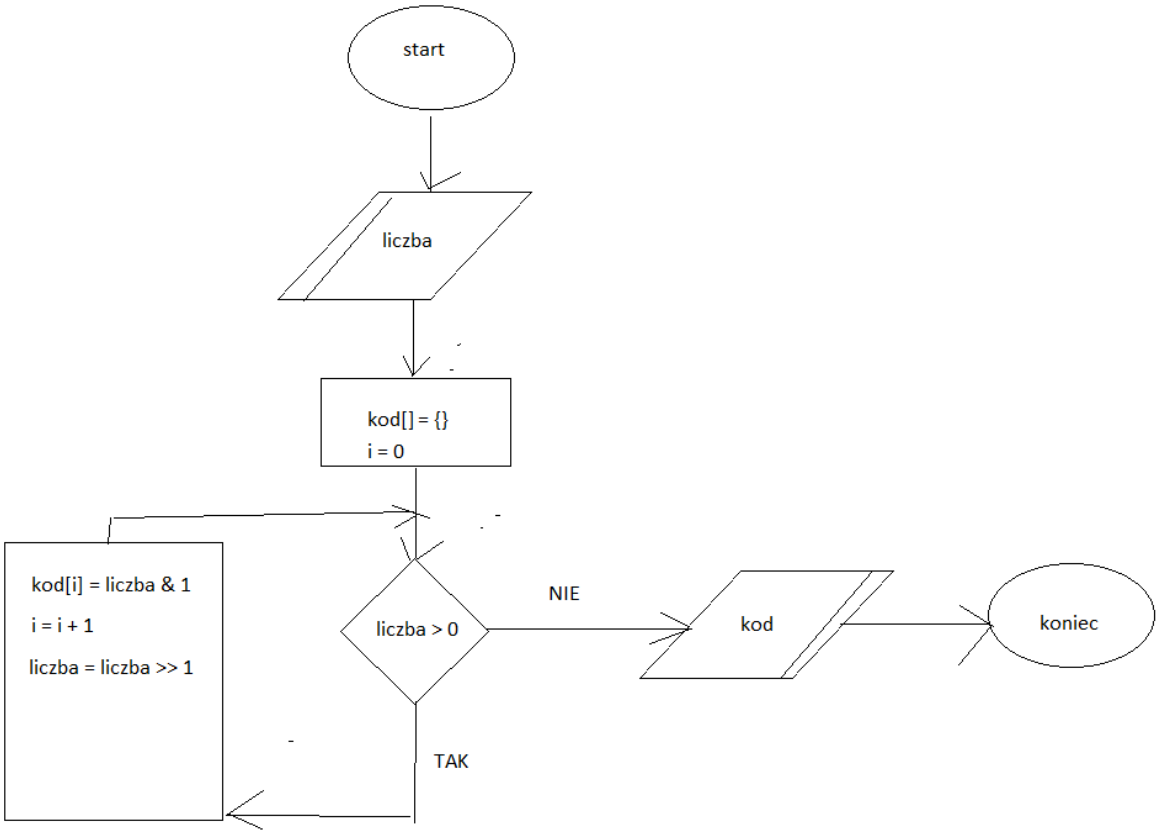
3. (4 pkt) Wybrać słowa kluczowe w jeżyku SQL:
   WHERE FILTER WHILE **SELECT** **JOIN** **GROUP BY** SORT BY
   MUST **HAVING** ALBUMS **ORDER BY** EXCEPT **IN** **LIKE** **NOT** **AND**
   RETURN CALCULATE

4. (4 pkt) Jaką wartość dziesiętną ma ciąg bitów: $0xc1440000_{(IEEE754\ p=16)} = (-1) * 2^3 * (1 + \frac{17}{32})_{(10)}$

5. (3 pkt) Dla bazy Chinook.db napisać zapytanie SQL: **Znaleźć 10 najdłuższych utworów (tracks) i z jakiego albumu pochodzą (albums) które są z gatunku (genre) „Vaporwave" zespołu „Rolling Stoners".**

```
SELECT
        Tracks.Name,
        Albums.Name
FROM
        Tracks
        INNER JOIN Genres ON Tracks.GenreId = Genres.GenreId
        INNER JOIN Albums ON Tracks.AlbumId = Albums.AlbumId
        INNER JOIN Artists ON Artists.ArtistId = Albums.ArtistId
WHERE
        Genres.Name LIKE 'Vapourwave'
        AND Artists.Name LIKE 'Rolling Stoners'
ORDER BY
        Milliseconds DESC
LIMIT 10;
```

6. (3 pkt) Narysować schemat blokowy który dla podanej liczby wypisuje jej binarną reprezentację



7. (8 pkt) Uzupełnić kod klasy w miejscach komentarzy:

```
class IPv4 {
private:
	// Metoda powinna ograniczyć wartość parametru 'octet' do przedziału 0-255
	// Można zastosować dowolną „zdroworozsądkową” technikę
	int sanitize(int octet) {
		return octet < 0 ? 0 : octet & 255;
	}
public:
	unsigned int octets[4] = { 0,0,0,0 };
	IPv4(int first, int second, int third, int fourth){
		octets[0] = sanitize(first);
		octets[1] = sanitize(second);
		octets[2] = sanitize(third);
		octets[3] = sanitize(fourth);
	};

	// Metoda powinna zwrócić nowy obiekt IPv4 reprezentujący adres rozgłoszeniowy
	// DLA CHĘTNYCH – bonusowe punkty
	IPv4 getBroadcastAddress(IPv4 & mask) {
		return IPv4(
			octets[0] | (~mask.octets[0] & 255),
			octets[1] | (~mask.octets[1] & 255),
			octets[2] | (~mask.octets[2] & 255),
			octets[3] | (~mask.octets[3] & 255)
		);
	};
	// Metoda powinna zwrócić nowy obiekt IPv4 reprezentujący adres bramy domyślnej
	// wg. tego obiektu
	IPv4 getGatewayAddress(IPv4 & mask) {
		return IPv4(
			octets[0] & mask.octets[0],
			octets[1] & mask.octets[1],
			octets[2] & mask.octets[2],
			(octets[3] & mask.octets[3]) + 1
		);
	};

	void print() {
		std::cout << octets[0] << "." << octets[1] << "."
			<< octets[2] << "." << octets[3] << std::endl;
	}
};

int main()
{
	IPv4 myIp = IPv4(192, 168, 1, 100);
	IPv4 mask = IPv4(255, 255, 255, 0);
	myIp.print();
	myIp.getBroadcastAddress(mask).print();
	myIp.getGatewayAddress(mask).print();
}
```