Střední průmyslová škola elektrotechnická Havířov	**PRG**	**Třída: 4B** **Skupina:** 1

Spojový seznam - Databáze cpp

Den: 05.03.2025

Jméno učitele: Božena Ralbovská

Jméno: Vojtěch Lisztwan

Známka: snad jedna

1. Zadání

- navrhněte třídy pro uzel a seznam
- záznam bude mít minimálně 4 položky (1 číslo, 1 řetězec, …).
- pro ovládání databáze navrhněte přívětivou uživatelskou nabídku

Pro následující akce v seznamu vytvořte vlastní metody:

- konstruktor, destruktor
- načtení údajů ze souboru (použijte textový soubor)
- uložení údajů do souboru
- přidání záznamu do seznamu na začátek
- přidání záznamu do seznamu na konec
- výpis všech záznamů
- vyhledání záznamu podle zvolené položky
- řazení záznamů (podle čísla i podle řetězce)
- odstranění vybraného záznamu podle čísla
- oprava údajů
- určení minima, maxima
- 2 výpočtové funkce např. součet, průměr, počet podle kritérií … (smysluplné vzhledem k datům)

2. Teoretický rozbor - analýza

Struktura kódu

- ### Struktura kódu
  - Kód je rozdělen do dvou tříd:
  - Node – představuje jednotlivý uzel v seznamu (představuje jedno pivo).
  - List – reprezentuje obousměrně vázaný seznam obsahující objekty třídy Node.
  - Každá třída obsahuje konstruktory, settery, gettery a další metody pro správu dat.
- ### Třída Node (Uzel seznamu)
  - Tato třída slouží jako základní stavební prvek seznamu. Každý objekt Node uchovává informace o pivu a obsahuje:
  - Atributy:
    - int number – unikátní ID piva.
    - string name – název piva.
    - double price – cena piva.
    - int ibu – hodnota hořkosti IBU.
    - Node* prev, *next – ukazatele na předchozí a následující uzel seznamu.
  - Konstruktory:
    - Bezparametrický konstruktor a přetížené verze s různými parametry pro inicializaci dat.
  - Settery a gettery:
    - setNumber(), setName(), setPrice(), setIbu() pro nastavení hodnot atributů.
    - getNumber(), getName(), getPrice(), getIbu() pro získání hodnot.
  - Ošetření hodnot:
    - setPrice() kontroluje rozsah ceny.
    - setIbu() kontroluje rozsah hořkosti IBU a vypisuje chybovou hlášku.
    - Přetížení operátoru << pro výpis obsahu uzlu na standardní výstup.
- ### Třída List (Dvojitě vázaný seznam piv)
  - Tato třída představuje kolekci objektů Node. Obsahuje:
  - Atributy:
    - Node* first – ukazatel na první prvek seznamu.
  - Konstruktory:

- Bezparametrický konstruktor inicializuje seznam jako prázdný.
- Konstruktor přijímající první uzel.

- Destruktor:
  - Uvolňuje paměť pro všechny uzly seznamu při jeho odstranění.
- Metody pro přidávání uzlů:
  - addNodeToStart() – přidává nový uzel na začátek seznamu.
  - addNodeToEnd() – přidává nový uzel na konec seznamu.
- Metody pro vyhledávání uzlů:
  - findById(), findByName(), findByIbu(), findByPrice() – umožňují vyhledávání piv podle různých kritérií.
- Metody pro práci se soubory:
  - loadFromFile() – načítání seznamu ze souboru.
  - saveToFile() – ukládání seznamu do souboru.
- Metody pro úpravu seznamu:
  - remove(Node* n) – odstranění uzlu ze seznamu.
  - edit(Node* n) – úprava existujícího uzlu.
- Metody pro řazení seznamu:
  - sortByNumber(), sortByPrice(), sortByName(), sortByIbu() – různé způsoby řazení seznamu.
  - Pomocná metoda zamena(Node* a, Node* b) – umožňuje výměnu dvou uzlů při řazení.
- Přetížení operátoru << pro výpis celého seznamu.

- **Objektově orientovaný přístup**
  - Kód využívá základní principy OOP:
  - Zapouzdření: Data jsou uložena v soukromých proměnných a lze k nim přistupovat pouze přes metody třídy.

3. Postup - popis vlastních funkcí

Přidávání nových uzlů

`addNodeToStart(Node n)`: Přidá nový uzel na začátek seznamu. Pokud seznam není prázdný, propojí nový uzel s původním prvním uzlem.

`addNodeToEnd(Node n)`: Přidá nový uzel na konec seznamu. Pokud seznam není prázdný, propojí nový uzel s posledním uzlem.

Vyhledávání piv

`findById(int cislo)`: Prochází seznam a vrací uzel s odpovídajícím ID.

`findByName(string nazev)`: Prochází seznam a vrací první uzel s odpovídajícím názvem.

`findByIbu(int ibucko)`: Vrací uzel, který odpovídá zadané hořkosti IBU.

`findByPrice(double cena)`: Vrací uzel s odpovídající cenou.

Řazení seznamu

`sortByNumber()`: Seřadí seznam podle ID piv.

`sortByPrice()`: Seřadí seznam podle ceny.

`sortByName()`: Seřadí seznam podle názvu.

`sortByIbu()`: Seřadí seznam podle hořkosti IBU.

`zamena(Node* a, Node* b)`: Pomocná metoda pro výměnu dvou uzlů při řazení.

Práce se soubory

`loadFromFile()`: Načte seznam piv ze souboru a vytvoří odpovídající uzly.

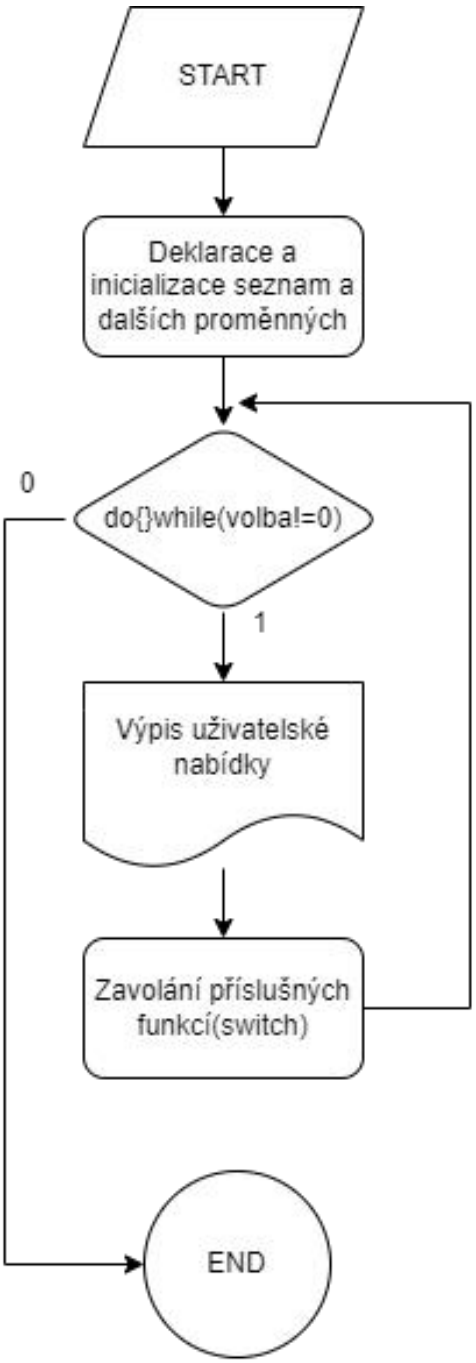
`saveToFile()`: Uloží aktuální seznam piv do souboru.

Úpravy a mazání uzlů

`edit(Node* n)`: Umožní uživateli změnit údaje existujícího piva.

`remove(Node* n)`: Odstraní pivo ze seznamu a uvolní paměť.

3. Vývojový diagram



4. program + blokový komentář

main.cpp

```
#include <iostream>
```

```cpp
#include "objekty.h"  // Zahrnuje externí knihovnu, která obsahuje definice tříd a funkcí pro práci se seznamem piv

using namespace std;

// Funkce pro zobrazení hlavního menu
void showMenu() {
    cout << "\n===== Pivni seznam =====" << endl;
    cout << "1. Pridat pivo na zacatek" << endl;
    cout << "2. Pridat pivo na konec" << endl;
    cout << "3. Vypsat seznam" << endl;
    cout << "4. Ulozit do souboru" << endl;
    cout << "5. Nacist ze souboru" << endl;
    cout << "6. Najit nejmensi cenu" << endl;
    cout << "7. Najit nejvetsi cenu" << endl;
    cout << "8. Vypocitat prumernou hodnotu ceny piva" << endl;
    cout << "9. Seradit podle ID" << endl;
    cout << "10. Seradit podle ceny" << endl;
    cout << "11. Seradit podle nazvu" << endl;
    cout << "12. Seradit podle IBU" << endl;
    cout << "13. Upravit pivo" << endl;
    cout << "14. Odstranit pivo" << endl;
    cout << "15. Hledat pivo" << endl;
    cout << "0. Konec" << endl;
    cout << "Vyber moznost: ";  // Vyzve uživatele k zadání volby z menu
}

// Funkce pro čtení nového piva zadaného uživatelem
Node readNew(){
    int cislo, ibucko;
    string nazev;
    double cena;
```

```
    // Získání údajů o novém pivu od uživatele
    cout << "Zadej ID piva: ";
    cin >> cislo;
    cout << "Zadej nazev piva: ";
    cin.ignore();  // Vyprázdní buffer
    getline(cin, nazev);  // Načte název piva
    cout << "Zadejte cenu piva: ";
    cin >> cena;
    cout << "Zadejte IBU piva: ";
    cin >> ibucko;
    return Node(cislo, nazev, cena, ibucko);  // Vytvoří a vrátí nový objekt typu Node (pivo)
}

int main() {
    List seznam;  // Vytvoří instanci třídy List pro uchování seznamu piv
    int volba, cislo;
    string nazev;
    Node* p = new Node();  // Vytvoří ukazatel na nový objekt Node (pivo)
    do {
        showMenu();  // Zobrazí hlavní menu
        cin >> volba;  // Načte volbu uživatele
        switch (volba) {
        case 1:
            seznam.addNodeToStart(readNew());  // Přidá nové pivo na začátek seznamu
            break;
        case 2:
            seznam.addNodeToEnd(readNew());  // Přidá nové pivo na konec seznamu
            break;
        case 3:
            cout << seznam;  // Vypíše celý seznam piv
            break;
        case 4:
```

```
            seznam.saveToFile();  // Uloží seznam piv do souboru
            break;
        case 5:
            seznam.loadFromFile();  // Načte seznam piv ze souboru
            break;
        case 6:
            p = seznam.min();  // Najde pivo s nejmenší cenou
            cout << "Nejmensi cena:" << endl;
            cout << *p;  // Vypíše pivo s nejmenší cenou
            break;
        case 7:
            p = seznam.max();  // Najde pivo s největší cenou
            cout << "Nejvetsi cena:" << endl;
            cout << *p;  // Vypíše pivo s největší cenou
            break;
        case 8:
            cout << "Prumerna cena: " << seznam.average() << endl;  // Vypočítá a vypíše průměrnou cenu piv
            break;
        case 9:
            seznam.sortByNumber();  // Seřadí seznam podle ID piva
            cout << "Seznam serazen podle ID." << endl;
            break;
        case 10:
            seznam.sortByPrice();  // Seřadí seznam podle ceny piva
            cout << "Seznam serazen podle ceny." << endl;
            break;
        case 11:
            seznam.sortByName();  // Seřadí seznam podle názvu piva
            cout << "Seznam serazen podle jmena." << endl;
            break;
        case 12:
            seznam.sortByIbu();  // Seřadí seznam podle IBU piva
```

```
        cout << "Seznam serazen podle IBU." << endl;
        break;
    case 13: {
        cout << "Zadej ID piva k uprave: ";
        cin >> cislo;
        Node* pivo = seznam.findById(cislo);  // Najde pivo podle ID
        if (pivo) seznam.edit(pivo);  // Umožní úpravu piva, pokud je nalezeno
        else cout << "Pivo nenalezeno." << endl;
        break;
    }
    case 14: {
        cout << "Zadej ID piva k odstraneni: ";
        cin >> cislo;
        Node* pivo = seznam.getFirst();  // Začne hledat pivo od prvního
        while (pivo && pivo->getNumber() != cislo) {
            pivo = pivo->getNextNode();  // Prohledá seznam, dokud nenajde požadované pivo
        }
        if (pivo) seznam.remove(pivo);  // Odstraní pivo, pokud je nalezeno
        else cout << "Pivo nenalezeno." << endl;
        break;
    }
case 15:
    seznam.find();  // Funkce pro hledání piva v seznamu
    break;
case 0:
    cout << "Ukonceni programu." << endl;
    cout << "prejete si ulozit databazi do souboru? 0/1" << endl;
    int volbicka;
    cin >> volbicka;
    if (volbicka == 1) {
        seznam.saveToFile();  // Uloží seznam do souboru při ukončení programu
        return 0;
```

```
            }
            break;
        default:
            cout << "Neplatna volba, zkus to znovu." << endl;  // Pokud je volba neplatná, uživatel je informován
        }
        system("pause");  // Pauza před opětovným zobrazením menu
        system("cls");  // Vyčistí obrazovku
    } while (volba != 0);  // Program pokračuje, dokud uživatel nezvolí možnost 0 pro ukončení

    return 0;  // Konec programu
}
```

objekty.h

```
#ifndef OBJEKTY_H
#define OBJEKTY_H

#include <iostream>  // Pro vstup/výstup
#include <fstream>   // Pro práci se soubory
#include <sstream>   // Pro práci se stringy a streamy
#include <string.h>  // Knihovna pro práci s řetězci (doporučováno spíše <string>)

using namespace std; // Používáme standardní jmenný prostor

// Třída reprezentující jedno pivo v seznamu
class Node {
private:
    int number; // ID piva
    string name; // Název piva
    double price; // Cena piva
```

```
    int ibu; // IBU (hořkost piva)
    Node *prev, *next; // Ukazatele na předchozí a následující pivo v seznamu (dvojlinkední seznam)

public:
    // Konstruktor pro inicializaci uzlu
    Node();
    Node(int cislo); // Konstruktor pouze s ID
    Node(int cislo, string nazev); // Konstruktor s ID a názvem
    Node(int cislo, string nazev, double cena); // Konstruktor s ID, názvem a cenou
    Node(int cislo, string nazev, double cena, int ibucko); // Konstruktor s ID, názvem, cenou a IBU

    // Settery pro nastavení hodnot atributů
    void setNumber(int c);
    void setName(string nazev);
    void setPrevNode(Node *p); // Nastaví ukazatel na předchozí uzel
    void setNextNode(Node *n); // Nastaví ukazatel na následující uzel

    // Setter pro cenu
    void setPrice(double cena) {
        if(cena > 0 && cena < 10000) { // Kontrola rozsahu ceny
            price = cena; // Nastavení ceny, pokud je v platném rozsahu
        }
    }

    // Setter pro IBU (hořkost piva)
    void setIbu(int ibucko) {
        if(ibucko <= 0 || ibucko > 100) { // Kontrola rozsahu IBU
            system("cls"); // Vymaže obrazovku
            cout << "Ibu musi byt v rozsahu 0 az 100" << endl; // Chybové hlášení
            system("pause"); // Pauza pro uživatele
            return;
        } else {
```

```
            ibu = ibucko; // Nastavení hodnoty IBU, pokud je platná
        }
    }

    // Gettery pro získání hodnot atributů
    double getPrice() { return price; } // Vrací cenu piva
    int getIbu() { return ibu; } // Vrací IBU piva
    Node* getPrevNode(); // Vrací ukazatel na předchozí uzel
    Node* getNextNode(); // Vrací ukazatel na následující uzel
    int getNumber(); // Vrací ID piva
    string getName(); // Vrací název piva

    // Přetížení operátoru << pro výpis objektu Node (pro výstup na obrazovku)
    friend ostream& operator<<(ostream& out, Node& n);
};

// Třída reprezentující seznam piv
class List {
    Node* first = nullptr; // Ukazatel na první prvek seznamu

public:
    // Konstruktor pro inicializaci seznamu
    List() {
        first = nullptr; // Inicializace jako prázdný seznam
    }
    List(Node* n) {
        first = n; // Konstruktor pro seznam s prvním uzlem
    }

    // Destruktor
    ~List() {
        Node* current = first;
```

```
    while (current != nullptr) {
        Node* nextNode = current->getNextNode(); // Uloží ukazatel na další uzel
        delete current; // Uvolní paměť pro aktuální uzel
        current = nextNode; // Pokračuje na další uzel
    }
}

// Getter pro získání prvního prvku seznamu
Node* getFirst() { return first; }

// Metody pro přidání nového piva do seznamu
void addNodeToStart(Node n); // Přidá nové pivo na začátek seznamu
void addNodeToStart2(int c, string nazev, double cena, int ibu); // Alternativní metoda pro přidání na začátek
void addNodeToEnd(Node n); // Přidá nové pivo na konec seznamu

// Přetížení operátoru << pro výpis celého seznamu piv
friend ostream& operator<<(ostream& out, List& n);

// Metody pro získání informací o seznamu
double average(); // Vypočítá průměrnou cenu piva v seznamu
Node* min(); // Najde pivo s nejnižší cenou
Node* max(); // Najde pivo s nejvyšší cenou

// Hledání piva v seznamu podle různých kritérií
Node* findById(int cislo); // Najde pivo podle ID
Node* findByName(string nazev); // Najde pivo podle názvu
Node* findByIbu(int ibucko); // Najde pivo podle IBU
Node* findByPrice(double cena); // Najde pivo podle ceny

// Metody pro práci se soubory
void loadFromFile(); // Načte seznam piv ze souboru
void saveToFile(); // Uloží seznam piv do souboru
```

```
    // Metody pro úpravu seznamu
    void remove(Node* n); // Odstraní pivo ze seznamu
    void edit(Node* n); // Umožní upravit pivo v seznamu

    // Metody pro řazení seznamu piv podle různých atributů
    void sortByNumber(); // Seřadí seznam podle ID
    void sortByPrice(); // Seřadí seznam podle ceny
    void sortByName(); // Seřadí seznam podle názvu
    void sortByIbu(); // Seřadí seznam podle IBU

    // Metoda pro hledání piva v seznamu (např. podle názvu)
    void find();

    // Pomocná metoda pro výměnu dvou uzlů v seznamu (např. při řazení)
    void zamena(Node* a, Node* b);
};

#endif
```

Objekty.cpp

```
#include "objekty.h"

// Konstruktor pro Node
Node::Node() {
    number = 0;           // Inicializace čísla piva na 0
    name = "pivo";        // Inicializace názvu piva na "pivo"
    price = 0;            // Inicializace ceny piva na 0
    next = nullptr;       // Inicializace ukazatele na další prvek seznamu jako nullptr
    prev = nullptr;       // Inicializace ukazatele na předchozí prvek seznamu jako nullptr
```

```
}

// Konstruktor pro Node s parametrem čísla piva
Node::Node(int cislo) {
    number = cislo;       // Nastavení čísla piva
    name = "pivo";        // Nastavení názvu piva na "pivo"
    price = 0;            // Cena piva je nastavena na 0
    next = nullptr;       // Inicializace ukazatele na další prvek seznamu jako nullptr
    prev = nullptr;       // Inicializace ukazatele na předchozí prvek seznamu jako nullptr
}

// Konstruktor pro Node s parametrem čísla a názvu piva
Node::Node(int cislo, string nazev) {
    number = cislo;       // Nastavení čísla piva
    name = nazev;         // Nastavení názvu piva
    price = 0;            // Cena piva je nastavena na 0
    next = nullptr;       // Inicializace ukazatele na další prvek seznamu jako nullptr
    prev = nullptr;       // Inicializace ukazatele na předchozí prvek seznamu jako nullptr
}

// Konstruktor pro Node s parametry čísla, názvu a ceny piva
Node::Node(int cislo, string nazev, double cena) {
    number = cislo;       // Nastavení čísla piva
    name = nazev;         // Nastavení názvu piva
    price = cena;         // Nastavení ceny piva
    next = nullptr;       // Inicializace ukazatele na další prvek seznamu jako nullptr
    prev = nullptr;       // Inicializace ukazatele na předchozí prvek seznamu jako nullptr
}

// Konstruktor pro Node s parametry čísla, názvu, ceny a ibu piva
Node::Node(int cislo, string nazev, double cena, int ibucko) {
    number = cislo;       // Nastavení čísla piva
```

```
    name = nazev;         // Nastavení názvu piva
    price = cena;         // Nastavení ceny piva
    next = nullptr;       // Inicializace ukazatele na další prvek seznamu jako nullptr
    prev = nullptr;       // Inicializace ukazatele na předchozí prvek seznamu jako nullptr
    ibu = ibucko;         // Nastavení hodnoty IBU (bitter units) piva
}

// Settery pro atributy Node
void Node::setNumber(int c) {
    this->number = c;     // Nastavení čísla piva
}

void Node::setName(string nazev) {
    this->name = nazev;   // Nastavení názvu piva
}

void Node::setPrevNode(Node *p) {
    this->prev = p;       // Nastavení předchozího prvku seznamu
}

void Node::setNextNode(Node *n) {
    this->next = n;       // Nastavení následujícího prvku seznamu
}

// Gettery pro atributy Node
Node* Node::getPrevNode() {
    return this->prev;    // Vrací ukazatel na předchozí prvek seznamu
}

Node* Node::getNextNode() {
    return this->next;    // Vrací ukazatel na následující prvek seznamu
}
```

```cpp
int Node::getNumber() {
    return this->number;  // Vrací číslo piva
}

string Node::getName() {
    return this->name;    // Vrací název piva
}

// Operátor << pro výpis Node do streamu
ostream& operator<<(ostream& out, Node& n) {
    out << "\t|\t" << n.getNumber() << "\t|\t" << n.getName() << "\t|\t" << n.getPrice() << "\t|\t" << n.getIbu() << endl;
    return out;
}

// Funkce pro přidání Node na začátek seznamu
void List::addNodeToStart(Node n) {
    Node* pom = new Node(n);  // Vytvoření nového uzlu
    if (findById(n.getNumber()) != nullptr) {
        cout << "Pivo s takovymto ID uz existuje" << endl; // Pokud pivo s tímto ID již existuje, nebudeme přidávat
        return;
    }
    if (this->first == nullptr) {  // Pokud je seznam prázdný
        first = pom;
    } else {
        pom->setNextNode(this->first);  // Nastavení nového uzlu na začátek seznamu
        this->first->setPrevNode(pom);  // Nastavení předchozího ukazatele na nový uzel
        this->first = pom;              // Nastavení začátku seznamu na nový uzel
    }
}

// Funkce pro přidání Node na začátek seznamu s parametry
void List::addNodeToStart2(int c, string nazev, double cena, int ibu) {
```

```
    Node n = Node(c, nazev, cena, ibu);  // Vytvoření uzlu
    addNodeToStart(n);  // Přidání uzlu na začátek seznamu
}

// Funkce pro přidání Node na konec seznamu
void List::addNodeToEnd(Node n) {
    if (first == nullptr) {  // Pokud je seznam prázdný
        addNodeToStart(n);    // Přidání na začátek
        return;
    }
    if (findById(n.getNumber()) != nullptr) {  // Pokud pivo s tímto ID již existuje
        cout << "Pivo s takovymto ID uz existuje" << endl;
        return;
    }
    Node *novy = new Node(n);  // Vytvoření nového uzlu
    Node *pom = first;         // Dočasný ukazatel pro procházení seznamu
    while (pom->getNextNode() != nullptr) {  // Procházení seznamu až na konec
        pom = pom->getNextNode();
    }
    pom->setNextNode(novy);     // Nastavení ukazatele posledního uzlu na nový uzel
    novy->setPrevNode(pom);     // Nastavení předchozího ukazatele nového uzlu
}

// Operátor << pro výpis List do streamu
ostream& operator<<(ostream& out, List& n) {
    Node *pom = n.getFirst();  // Dočasný ukazatel na první uzel seznamu
    if (n.getFirst() == nullptr) {  // Pokud je seznam prázdný
        out << "Seznam je prazdny." << endl;
        return out;
    }
    out << "\t|\tID\t|\tNazev piva\t|\tCena\t|\tIBU" << endl << endl;
    while (pom != nullptr) {  // Procházení seznamu a výpis jednotlivých uzlů
```

```
        out << *pom << endl;
        pom = pom->getNextNode();
    }
    return out;
}

// Funkce pro výpočet průměrné ceny piva v seznamu
double List::average() {
    Node *pom = new Node();  // Dočasný uzel pro procházení seznamu
    double sum = 0;          // Suma cen piv
    double count = 0;        // Počet piv
    if (first == nullptr) {  // Pokud je seznam prázdný
        cout << "Seznam je prazdny" << endl;
        return -1;
    }
    for (pom = this->getFirst(); pom != nullptr; pom = pom->getNextNode()) {  // Procházení seznamu
        count++;
        sum += pom->getPrice();  // Sčítání cen piv
    }
    return sum / count;  // Vrací průměrnou cenu
}

// Funkce pro nalezení Node s minimální cenou
Node* List::min() {
    if (first == nullptr) {  // Pokud je seznam prázdný
        cout << "Seznam je prazdny" << endl;
        return nullptr;
    }
    Node *pom = new Node();
    int min = this->first->getPrice();  // Inicializace minimální ceny
    Node *minNode = this->first;        // Inicializace uzlu s minimální cenou
    for (pom = this->getFirst(); pom != nullptr; pom = pom->getNextNode()) {  // Procházení seznamu
```

```
        if (pom->getPrice() < min) {  // Pokud je cena uzlu menší než aktuální minimum
            min = pom->getPrice();
            minNode = pom;  // Nastavení uzlu s novým minimem
        }
    }
    return minNode;  // Vrací uzel s minimální cenou
}

// Funkce pro nalezení Node s maximální cenou
Node* List::max() {
    if (first == nullptr) {  // Pokud je seznam prázdný
        cout << "Seznam je prazdny" << endl;
        return nullptr;
    }
    Node *pom = new Node();
    int max = this->first->getPrice();  // Inicializace maximální ceny
    Node *maxNode = this->first;        // Inicializace uzlu s maximální cenou
    for (pom = this->getFirst(); pom != nullptr; pom = pom->getNextNode()) {  // Procházení seznamu
        if (pom->getPrice() > max) {  // Pokud je cena uzlu větší než aktuální maximum
            max = pom->getPrice();
            maxNode = pom;  // Nastavení uzlu s novým maximem
        }
    }
    return maxNode;  // Vrací uzel s maximální cenou
}

// Funkce pro nalezení Node podle ID
Node* List::findById(int cislo) {
    Node *pom = first;
    while (pom != nullptr) {  // Procházení seznamu
        if (pom->getNumber() == cislo) return pom;  // Pokud nalezne uzel s odpovídajícím ID
        pom = pom->getNextNode();
```

```
    }
    return nullptr;  // Pokud není nalezeno
}

// Funkce pro nalezení Node podle názvu
Node* List::findByName(string nazev) {
    Node *pom = first;
    while (pom != nullptr) {  // Procházení seznamu
        if (pom->getName() == nazev) return pom;  // Pokud nalezne uzel s odpovídajícím názvem
        pom = pom->getNextNode();
    }
    return nullptr;  // Pokud není nalezeno
}

// Funkce pro nalezení Node podle IBU
Node* List::findByIbu(int ibucko) {
    Node *pom = first;
    while (pom != nullptr) {  // Procházení seznamu
        if (pom->getIbu() == ibucko) return pom;  // Pokud nalezne uzel s odpovídajícím IBU
        pom = pom->getNextNode();
    }
    return nullptr;  // Pokud není nalezeno
}

// Funkce pro nalezení Node podle ceny
Node* List::findByPrice(double cena) {
    Node *pom = first;
    while (pom != nullptr) {  // Procházení seznamu
        if (pom->getPrice() == cena) return pom;  // Pokud nalezne uzel s odpovídající cenou
        pom = pom->getNextNode();
    }
    return nullptr;  // Pokud není nalezeno
```

```
}

// Funkce pro načtení piv ze souboru
void List::loadFromFile() {
    ifstream file;
    file.open("pivecka.txt");  // Otevření souboru
    if (!file.is_open()) {  // Pokud se soubor nepodařilo otevřít
        cout << "soubor se nepodarilo otevrit" << endl;
        return;
    }
    string radek;  // Dočasná proměnná pro čtení řádků ze souboru
    string cislo, nazev, cena, ibucko;
    double cen;
    int c;
    int ibuc;

    // Vymazání stávajícího seznamu
    while (first != nullptr) {
        Node* temp = first;
        first = first->getNextNode();
        delete temp;
        temp = nullptr;
    }
    this->first = nullptr;

    Node x;  // Dočasný uzel pro vytvoření nových piv
    // Načítání piv z každého řádku souboru
    while (getline(file, radek)) {
        stringstream radecek = stringstream(radek);
        getline(radecek, cislo, '$');  // Čtení ID piva
        c = stoi(cislo);
        getline(radecek, nazev, '$');  // Čtení názvu piva
```

```
        getline(radecek, cena, '$');  // Čtení ceny piva
        getline(radecek, ibucko, '$');  // Čtení IBU
        cen = stod(cena);
        ibuc = stoi(ibucko);
        x = Node(c, nazev, cen, ibuc);  // Vytvoření uzlu piva
        addNodeToEnd(x);  // Přidání uzlu do seznamu
    }
    cout << "Uspesne nacteno ze souboru pivecka.txt" << endl;
}

// Funkce pro uložení piv do souboru
void List::saveToFile() {
    ofstream file;
    file.open("pivecka.txt");  // Otevření souboru pro zápis
    if (!file.is_open()) {  // Pokud se soubor nepodařilo otevřít
        cout << "soubor se nepodarilo otevrit" << endl;
        return;
    }
    Node *pom = new Node();
    for (pom = this->getFirst(); pom != nullptr; pom = pom->getNextNode()) {  // Procházení seznamu
        file << pom->getNumber() << "$" << pom->getName() << "$" << pom->getPrice() << "$" << pom->getIbu() << "$" << endl;
    }
    cout << "Saved to file pivecka.txt" << endl;  // Informace o úspěšném uložení
    file.close();  // Zavření souboru
}

// Funkce pro odstranění uzlu ze seznamu
void List::remove(Node *n) {
    // Případ, kdy uzel k odstranění je první uzel
    if (n == first) {
        first = first->getNextNode();  // Nastavíme první uzel na následující uzel
        delete n;                      // Smažeme aktuální uzel
```

```
        n = nullptr;                    // Nastavíme ukazatel na nullptr (dobrá praxe, ale není nutné)
        return;
    }
    // Případ, kdy uzel je poslední
    if (n->getNextNode() == nullptr) {
        n->getPrevNode()->setNextNode(nullptr); // Nastavíme ukazatel na následující uzel předchozího uzlu na nullptr
        delete n;                               // Smažeme aktuální uzel
        n = nullptr;                            // Nastavíme ukazatel na nullptr
        return;
    }

    // Případ, kdy uzel je někde uprostřed
    Node *pre;
    Node *nex;
    pre = n->getPrevNode();          // Získáme předchozí uzel
    nex = n->getNextNode();          // Získáme následující uzel
    pre->setNextNode(nex);           // Nastavíme ukazatel na následující uzel předchozího uzlu
    nex->setPrevNode(pre);           // Nastavíme ukazatel na předchozí uzel následujícího uzlu
}

// Funkce pro úpravu detailů piva (uzlu)
void List::edit(Node *n) {
    double cena;        // Cena piva
    string nam;         // Název piva
    int ibucko;         // IBU (International Bitterness Units)
    int volba;          // Uživatelská volba pro to, co upravit

    do {
        system("cls");  // Vyčistí obrazovku (Windows)
        cout << "--------------------------------" << endl;
        cout << "Zadejte co byste radi upravili" << endl; // Výzva pro volbu
        cout << "0. Zpet" << endl;
```

```
        cout << "1. Nazev piva" << endl;
        cout << "2. Nova cena" << endl;
        cout << "3. Nove ibu" << endl;
        cin >> volba;    // Získáme uživatelský vstup pro volbu

        switch (volba) {
            case 1:  // Upravit název piva
            system("cls");
            cout << "Zadejte novy nazev piva" << endl;
            cin.ignore();  // Vyčistí vstupní buffer
            getline(cin, nam);  // Získáme nový název piva
            n->setName(nam);    // Aktualizujeme název
            break;
            case 2:  // Upravit cenu piva
            cout << "Zadejte novou cenu piva" << endl;
            cin >> cena;      // Získáme novou cenu piva
            n->setPrice(cena); // Aktualizujeme cenu
            break;
            case 3:  // Upravit IBU
            cout << "Zadejte nove ibu piva" << endl;
            cin >> ibucko;    // Získáme novou hodnotu IBU
            n->setIbu(ibucko); // Aktualizujeme IBU
            break;
            case 0:  // Ukončit
            break;
        default:
            cout << "zkus to znovu";  // Pokud je volba neplatná
            break;
        }
    } while (volba != 0);  // Opakujeme, dokud uživatel nevybere možnost ukončení
}
```

```
// Funkce pro seřazení seznamu podle čísla u uzlů
void List::sortByNumber() {
    if (first == nullptr || first->getNextNode() == nullptr) return; // Pokud je seznam prázdný nebo obsahuje pouze jeden uzel

    bool swapped;
    do {
        swapped = false;
        Node* pom = first; // Dočasný ukazatel pro procházení seznamu

        while (pom->getNextNode() != nullptr) {
            if (pom->getNumber() > pom->getNextNode()->getNumber()) {
                zamena(pom, pom->getNextNode()); // Vyměníme uzly, pokud je číslo aktuálního uzlu větší než číslo následujícího
                swapped = true; // Označíme, že došlo k výměně
            } else {
                pom = pom->getNextNode(); // Posuneme se na další uzel, pokud nedošlo k výměně
            }
        }
    } while (swapped);  // Opakujeme, dokud nedojde k žádné výměně
}

// Funkce pro seřazení seznamu podle ceny u uzlů
void List::sortByPrice() {
    if (first == nullptr || first->getNextNode() == nullptr) return; // Pokud je seznam prázdný nebo obsahuje pouze jeden uzel

    bool swapped;
    do {
        swapped = false;
        Node* pom = first;

        while (pom->getNextNode() != nullptr) {
            if (pom->getPrice() > pom->getNextNode()->getPrice()) {
```

```
                zamena(pom, pom->getNextNode()); // Vyměníme uzly, pokud je cena aktuálního uzlu větší než cena následujícího
                swapped = true;  // Označíme, že došlo k výměně
            } else {
                pom = pom->getNextNode();  // Posuneme se na další uzel, pokud nedošlo k výměně
            }
        }
    } while (swapped);  // Opakujeme, dokud nedojde k žádné výměně
}

// Funkce pro seřazení seznamu podle názvu u uzlů
void List::sortByName() {
    if (first == nullptr || first->getNextNode() == nullptr) return; // Pokud je seznam prázdný nebo obsahuje pouze jeden uzel

    bool swapped;
    do {
        swapped = false;
        Node* pom = first;

        while (pom->getNextNode() != nullptr) {
            if (pom->getName() > pom->getNextNode()->getName()) {
                zamena(pom, pom->getNextNode()); // Vyměníme uzly, pokud je název aktuálního uzlu lexikograficky větší než
název následujícího
                swapped = true;  // Označíme, že došlo k výměně
            } else {
                pom = pom->getNextNode();  // Posuneme se na další uzel, pokud nedošlo k výměně
            }
        }
    } while (swapped);  // Opakujeme, dokud nedojde k žádné výměně
}

// Funkce pro seřazení seznamu podle IBU (hořkosti) u uzlů
void List::sortByIbu() {
```

```
    if (first == nullptr || first->getNextNode() == nullptr) return; // Pokud je seznam prázdný nebo obsahuje pouze jeden uzel

    bool swapped;
    do {
        swapped = false;
        Node* pom = first;

        while (pom->getNextNode() != nullptr) {
            if (pom->getIbu() > pom->getNextNode()->getIbu()) {
                zamena(pom, pom->getNextNode()); // Vyměníme uzly, pokud je IBU aktuálního uzlu větší než IBU následujícího
                swapped = true;  // Označíme, že došlo k výměně
            } else {
                pom = pom->getNextNode();  // Posuneme se na další uzel, pokud nedošlo k výměně
            }
        }
    } while (swapped);  // Opakujeme, dokud nedojde k žádné výměně
}

// Funkce pro výměnu dat mezi dvěma uzly
void List::zamena(Node* a, Node* b) {
    // Dočasně uložíme data prvního uzlu
    int numberFirst = a->getNumber();
    string nameFirst = a->getName();
    double priceFirst = a->getPrice();
    int firstIbu = a->getIbu();

    // Vyměníme data mezi dvěma uzly
    a->setName(b->getName());
    a->setNumber(b->getNumber());
    a->setPrice(b->getPrice());
    a->setIbu(b->getIbu());
```

```
    b->setPrice(priceFirst);
    b->setName(nameFirst);
    b->setNumber(numberFirst);
    b->setIbu(firstIbu);
}

// Funkce pro hledání uzlu podle různých kritérií
void List::find() {
    double cena;        // Cena piva
    string nam;         // Název piva
    int ibucko;         // Hodnota IBU
    int volba;          // Uživatelská volba pro kritéria hledání
    int idecko;         // ID piva pro hledání
    Node *x = new Node();  // Dočasný uzel pro uložení výsledku

    do {
        system("cls");  // Vyčistí obrazovku
        cout << "--------------------------------" << endl;
        cout << "Podle ceho chcete hledat" << endl;  // Zeptáme se uživatele, podle čeho hledat
        cout << "0. Zpet" << endl;
        cout << "1. ID piva" << endl;
        cout << "2. Nazev" << endl;
        cout << "3. Cena" << endl;
        cout << "4. IBU" << endl;
        cin >> volba;  // Získáme uživatelský vstup pro kritérium hledání

        x = nullptr;  // Resetujeme dočasný uzel
        switch (volba) {
            case 1:  // Hledat podle ID
            system("cls");
            cout << "Zadejte ID piva" << endl;
            cin >> idecko;
```

```
        x = findById(idecko);  // Zavoláme funkci pro hledání podle ID
        break;
        case 2:  // Hledat podle názvu
        system("cls");
        cout << "Zadejte nazev piva" << endl;
        cin.ignore();  // Vyčistí vstupní buffer
        getline(cin, nam);  // Získáme název piva
        x = findByName(nam);  // Zavoláme funkci pro hledání podle názvu
        break;
        case 3:  // Hledat podle ceny
        cout << "Zadejte cenu piva" << endl;
        cin >> cena;
        x = findByPrice(cena);  // Zavoláme funkci pro hledání podle ceny
        break;
        case 4:  // Hledat podle IBU
        cout << "Zadejte IBU piva" << endl;
        cin >> ibucko;
        x = findByIbu(ibucko);  // Zavoláme funkci pro hledání podle IBU
        break;
        case 0:  // Ukončit
        delete x;  // Uvolníme dočasný uzel
        x = nullptr;
        return;
        break;
    default:
        cout << "zkus to znovu" << endl;  // Pokud je volba neplatná
        break;
    }

    if (x != nullptr) cout << *x << endl;  // Pokud byl uzel nalezen, vypíšeme ho
    else cout << "nenalezeno" << endl;  // Pokud nebyl uzel nalezen
```

```
        system("pause");  // Pauza před pokračováním

    } while (volba != 0);  // Opakujeme, dokud uživatel nevybere možnost ukončení

}
```

5. Zhodnocení

Silné stránky:

- Projekt implementuje všechny požadované funkce.
- Použití dvojitě propojeného seznamu je vhodné pro daný účel.
- Kód je relativně přehledný a dobře komentovaný.
- Práce se soubory.

Slabé stránky:

- Uživatelské rozhraní je velmi jednoduché a postrádá některé pokročilejší funkce (např. validace vstupu).
- Chybí ošetření některých chybových stavů (např. neexistující soubor).
- Některé části kódu by mohly být optimalizovány pro lepší výkon.
- Není implementována možnost filtrování.

Možná vylepšení:

- Implementace grafického uživatelského rozhraní.
- Přidání validace vstupních dat.
- Ošetření chybových stavů a přidání robustnějšího zpracování chyb.
- Optimalizace kódu pro lepší výkon.
- Implementace filtrování.
- Implementace vyhledávání podle více kritérií najednou.

Závěr

Zadání se podařilo splnit víceméně bez problémů. Jednak jsou zde věci, které by šlo vylepšit. Nicméně jsem z výsledkem spokojen. Program by šlo vylepšit o lepší informování uživatele o průběhu některých operací, například načítání ze souboru. Také by šlo lépe ošetřit chybové stavy. Nicméně program je celkem robustní a celkem dobře rozšiřitelný.