Ce Am Nevoie YURR

Setters

- Pentru float*/int*

```
void setVotes(int noOfCandidates, int* votes) {
    if (votes == nullptr) throw "Number of votes needs to have at least 1 value";
    else {
        for (int i=0; i < noOfCandidates; i++) {
            this->votes[i] = votes[i];
        }
    }
}
```

- Pentru string*

```
void setCapturedPhotos(string* capturedPhotos, int noPhotos) {
    if (capturedPhotos == nullptr || noPhotos == 0) throw "Array must have at least one value";
    else {
        delete[] this->capturedPhotos;
        this->capturedPhotos = new string[noPhotos];
        this->noPhotos = noPhotos;
        for (int i = 0; i < noPhotos; i++) {
            this->capturedPhotos[i] = capturedPhotos[i];
        }
    }
}
```

- Pentru char*

```
void setName(const char* name) {
    if (name == nullptr || strlen(name) < 3) throw "Name must be longer";
    delete[] this->name;
    this->name = new char[strlen(name) + 1];
    strcpy_s(this->name, strlen(name) + 1, name);
}
```

Getters

- Pentru float*/int*

```
int* getVotes()  {
    int* copy = new int[this->noOfCandidates];
    for (int i = 0; i < noOfCandidates; i++) {
        copy[i] = votes[i];
    }
    return copy;
}
```

- Pentru string*

```
string* getCapturedPhotos() const{
    if (capturedPhotos == nullptr || noPhotos==0) return nullptr;
    else {
        string* copy = new string[noPhotos];
        for (int i = 0; i < noPhotos; i++) {
            copy[i] = capturedPhotos[i];
        }
        return copy;
    }
}
```

- Pentru char*

```
char* getName() {
    if (name == nullptr) return nullptr;
    else {
        char* copy = new char[strlen(this->name) + 1];
        strcpy_s(copy, strlen(this->name) + 1, name);
        return copy;
    }
}
```

Constructor

-const se initializeaza sus, la fel ca si celelalte attribute normale

-pt pointer folosesti metoda set

Destructor

In destructor stergem toate variabilele care sunt de tip pointer;

Copy contructor

- Pentru float*/int*

```cpp
Election& operator=(const Election& other) {
    if (this != &other) {
        this->region = other.region;
        this->scope = other.scope;
        this->noOfCandidates = other.noOfCandidates;

        if (this->votes != nullptr) {
            delete[] this->votes;
        }

        if (other.votes != nullptr) {
            this->votes = new int[other.noOfCandidates];
            for (int i = 0; i < other.noOfCandidates; i++) {
                this->votes[i] = other.votes[i];
            }
        }
        else {
            this->votes = nullptr;
        }
    }
    return *this;
}
```

- Pentru string*

```cpp
DigitalCamera& operator=(const DigitalCamera& other) {
    if (this != &other) {

        this->brand = other.brand;
        this->type = other.type;
        this->batteryLife = other.batteryLife;
        this->hasStorageCard = other.hasStorageCard;

        if (this->capturedPhotos != nullptr) delete[] this->capturedPhotos;
        if (other.capturedPhotos != nullptr) {
            this->capturedPhotos = new string[other.noPhotos];
            this->noPhotos = other.noPhotos;
            for (int i = 0; i < other.noPhotos; i++) {
                this->capturedPhotos[i] = other.capturedPhotos[i];
            }
        }
        else this->capturedPhotos = nullptr;
    }
    return *this;
}
```

- Pentru char*

```cpp
ElectionCandidate& operator=(const ElectionCandidate& newCandidate) {
    if (this != &newCandidate) {
        delete[] name;

        this->party = newCandidate.party;
        this->votes = newCandidate.votes;

        if (newCandidate.name != nullptr) {
            this->name = new char[strlen(newCandidate.name) + 1];
            strcpy_s(this->name, strlen(newCandidate.name) + 1, newCandidate.name);
        }
        else {
            this->name = nullptr;
        }
    }
    return *this;
}
```

Initializari

- Static - sunt definite in clasa, pot fi initializate doar in afara clasei
- Const – sunt definite in clasa, dar initializate doar in constructor
- Static const – sunt definite in clasa, pot fi initializate in clasa sau in afara
- Variabila pointer – sunt initializa cu nullptr, indifferent de tip(de ex si string* si char* si float* sunt initializate toate cu nullptr)
- Enum – sunt definite inafara clasei, iar atributul de tip enum este initializat cu prima valoare a enumeratiei
- Restul de tipuri de variabile:
    - Boolean – false
    - Int – 0;
    - Float -0.0 f; (trebuie specificat ca este variabila tip float)
    - Double – 0.0;
    - String – “ ”;

Toti operatorii

<< (ostream operator)

```
friend ostream& operator<<(ostream& os, const Book& book) {
    os << "ISBN: " << book.isbn << " | "
        << "Title: " << book.title << " | "
        << "Genre: " << (book.genre == FICTION ? "Fiction" : book.genre == NON_FICTION ? "Non-Fiction" : "Science Fiction") << " | "
        << "Has Audiobook: " << (book.hasAudiobook ? "Yes" : "No") << " | "
        << "Author: " << book.author << " | "
        << "Pages: " << book.noOfPages;
    return os;
}
```

>> (istream operator)

```cpp
friend istream& operator>>(istream& in, Book& book) {
    string genreInput, audiobookInput;

    // Read ISBN
    cout << "Enter ISBN: ";
    in >> book.isbn;
    in.ignore();  // Clear newline after ISBN

    // Read Title
    cout << "Enter Title: ";
    getline(in, book.title);

    // Read Genre
    cout << "Enter Genre (Fiction/Non-Fiction/Science Fiction): ";
    getline(in, genreInput);
    if (genreInput == "Fiction") book.genre = FICTION;
    else if (genreInput == "Non-Fiction") book.genre = NON_FICTION;
    else book.genre = SCIENCE_FICTION;

    // Read Audiobook availability
    cout << "Has Audiobook (Yes/No): ";
    getline(in, audiobookInput);
    book.hasAudiobook = (audiobookInput == "Yes");

    // Read Author
    cout << "Enter Author: ";
    getline(in, book.author);

    // Read Number of Pages
    cout << "Enter Number of Pages: ";
    in >> book.noOfPages;

    return in;
}
```

==

```cpp
bool Book::operator==(const Book& newbook) {
    if (this == &newbook) return true;
    if (this->title == nullptr || newbook.title == nullptr) return false;
    if (strcmp(this->title, newbook.title) != 0) return false;
    if (this->hasAudiobook != newbook.hasAudiobook) return false;
    if (this->genre != newbook.genre) return false;
    if (this->author != newbook.author) return false;
    if (this->isbn != newbook.isbn) return false;
    if (this->noOfPages != newbook.noOfPages) return false;

    return true;
}
```

String and const attributes support direct comparison, while the char* doesn't and must check equality with strcmp.

++ prefix

```
Book& Book::operator++() {
    ++noOfPages;
    return *this;
}
```

++ postfix

```
Book& Book::operator++(int) {
    Book copy = *this;
    this->noOfPages += 1;
    return copy;
}
```

-- prefix

```
Book& Book::operator--() {
    if (noOfPages > 0) --noOfPages;
    else cout << "Number of pages cannot be less than zero." << endl;
    return *this;
}
```

-- postfix

```
Book& Book::operator--(int) {
    Book copy = *this;
    if (noOfPages > 0) this->noOfPages -= 1;
    else cout << "Number of pages cannot be less than zero." << endl;
    return copy;
}
```

+

```
Book operator+(const Book& newbook) const {
    Book copy = *this;
    copy.noOfPages += newbook.noOfPages;
    return copy;
}
```

-

```
Book operator-(const Book& newbook) const {
    Book copy = *this;
    copy.noOfPages -= newbook.noOfPages;
    return copy;
}
```

*

```
Book operator*(const Book& newbook) const {
    Book copy = *this;
    copy.noOfPages *= newbook.noOfPages;
    return copy;
}
```

/

```
Book operator/(const Book& newbook) const {
    Book copy = *this;
    copy.noOfPages /= newbook.noOfPages;
    return copy;
}
```

!

```cpp
bool operator!(const Book& newbook) const {
    return noOfPages != newbook.noOfPages;
}
```

+=,-=,*=,/=

```cpp
// += operator (Add pages)
Book& operator+=(int pages) {
    this->noOfPages += pages;  // Add the number of pages
    return *this;  // Return the current object
}

// -= operator (Subtract pages)
Book& operator-=(int pages) {
    this->noOfPages -= pages;  // Subtract the number of pages
    return *this;  // Return the current object
}

// *= operator (Multiply pages by a multiplier)
Book& operator*=(int multiplier) {
    this->noOfPages *= multiplier;  // Multiply noOfPages by the multiplier
    return *this;  // Return the current object
}

// /= operator (Divide pages by a divisor)
Book& operator/=(int divisor) {
    if (divisor == 0) {
        std::cerr << "Error: Division by zero is not allowed." << std::endl;
        return *this; // Return the original object if division by zero
    }
    this->noOfPages /= divisor;  // Divide noOfPages by the divisor
    return *this;  // Return the current object
}
```

!=

```cpp
bool Book::operator!=(const Book& newbook)
{
    return !(*this == newbook);
}
```

[] index

```cpp
char& operator[](int index) {
    if (title == nullptr || index < 0 || index >= strlen(title)) {
        cout << "Index out of bounds or title is null!" << endl;
        exit(0);
    }
    return title[index];
}
```

() cast

```cpp
operator std::string() const {
    std::stringstream ss;
    ss << "ISBN: " << this->isbn << " | "
       << "Title: " << (this->title ? this->title : "No Title") << " | "
       << "Author: " << this->author << " | "
       << "Pages: " << this->noOfPages;
    return ss.str();
}
```

Title was considered char*, author is string and noOfPages is int.

<

```cpp
bool Book::operator<(const Book& newbook) {
    return noOfPages < newbook.noOfPages;
}
```

>

```cpp
bool Book::operator>(const Book& newbook) {
    return noOfPages > newbook.noOfPages;
}
```

<=

```cpp
bool Book::operator<=(const Book& newbook) {
    return noOfPages <= newbook.noOfPages;

}
```

>=

```
bool Book::operator>=(const Book& newbook) {
    return noOfPages >= newbook.noOfPages;
}
```

=

```
Book& Book::operator=(const Book& other) {
    if (this != &other) {
        if (this->title != nullptr) {
            delete[] this->title;
        }
        this->title = new char[strlen(other.title) + 1];
        strcpy(this->title, other.title);

        this->isbn = other.isbn;
        this->genre = other.genre;
        this->hasAudiobook = other.hasAudiobook;
        this->author = other.author;
        this->noOfPages = other.noOfPages;
    }
    return *this;
}
```

```cpp
// Adding a scalar to noOfPages
Book operator+(int multiplier) const {
    Book copy = *this;
    copy.noOfPages += multiplier;  // Adding the multiplier to noOfPages
    return copy;
}

// Subtracting a scalar from noOfPages
Book operator-(int multiplier) const {
    Book copy = *this;
    copy.noOfPages -= multiplier;  // Subtracting the multiplier from noOfPages
    return copy;
}

// Multiplying noOfPages by a scalar
Book operator*(int multiplier) const {
    Book copy = *this;
    copy.noOfPages *= multiplier;  // Multiplying noOfPages by the multiplier
    return copy;
}

// Dividing noOfPages by a scalar
Book operator/(int multiplier) const {
    if (multiplier == 0) {
        std::cerr << "Error: Division by zero is not allowed." << std::endl;
        return *this; // Returning the original object if division by zero
    }

    Book copy = *this;
    copy.noOfPages /= multiplier;  // Dividing noOfPages by the multiplier
    return copy;
}
```

==, !=, <, >, <=, >= - relational operators

<<, >> - stream operators

+,-,*,/ - arithmetic operators

[],() – cast operators

++,-- - increment and decrement operators

+=, -=, *=, /= compound assignment operators

Relational operators return bool because they are used for comparisons and logical conditions.

Stream operators, arithmetic operators, cast operators, increment/decrement operators, and compound assignment operators return the object itself (or a modified version of the object) for further use or chaining.

Additional notes

- delete is used to free memory allocated for a single object that was created using new.
- delete[] is used to free memory allocated for an array of objects created with new[].