**Course Project ADT**

**Lyuboslav Gigov**

Abstract Data Types for Bookstore Project:

* Abstract base class *Book* with the following pure virtual functions:

virtual int *Search*(SearchCriteria s1) = 0; // Estimates if the user is likely to enjoy the book //

virtual void *DisplayGenre*() = 0; // Displays the genre of current object (book) //

The class has these data members:

string name; // Name of the book //

string author; // Author of the book //

double price; // Price of the book //

int grade; // Estimated compatibility of book (0-100) with user’s preferences //

Axioms:

1. price > 0.
2. *Search()* should return an integer from 0-100.
3. It follows that 0 <= grade <= 100.

* The class *Fiction* inherits from *Book*. It doesn’t provide an implementation to any of the abstract functions and thus it is also abstract.
* The class *ScienceFiction* inherits from *Fiction* and has the following methods:

virtual int *Search*(SearchCriteria s2); // Unique implementation based on the genre //

virtual void *DisplayGenre*(); // Displays “Science Fiction” //

* The class *Fantasy* inherits from *Fiction* and overrides the following methods:

virtual int *Search*(SearchCriteria s3);

virtual void *DisplayGenre*(); // Displays “Fantasy” //

* *Romance* inherits from *Fiction* and, just like the above classes, it overrides the two polymorphic functions of the abstract base class.
* The class *NonFiction* inherits from *Book,* but it remains an abstract class because it provides no implementation of the functions.
* The classes *Biographies* and *Encylopedias* inherit from *NonFiction* and again override the two polymorphic functions.

All the non-abstract classes (*Romance, Fiction, ScienceFiction, etc.)* have constructors of this form:

*ScienceFiction / Fantasy / etc.* (string n, string a, double p);

Additionally, they have getters for their data members (name, author, price, grade) and a setter that is used to set the value of their grade.

The main feature of my project is the polymorphic *Search()* function. It takes a struct SearchCriteria as input, where the preferences and interests of the reader are stored. It has the following signature:

struct SearchCriteria {

int age; // Age of the reader //

double maxPrice; // Maximum price the user is willing to pay for a book //

Gender g; // enum (male, female, other) //

Context c; // enum (school, vacation, home, work, gift) //

PersonalInterests p; // enum (sport, science, technology, celebrities, history) //

};

The function will be called to books (objects) of different genres, and the implementation for each genre (class) will be different. One similarity between all implementations will be the price consideration: if the book exceeds the maximum price the user is willing to pay, it will immediately receive a 0. Other than that, each implementation of *Search()* will look for genre-specific patterns in the user’s input information. For example, if the user is a teenage female and wants a book for leisure at home, this will be taken into account by the *Search()* function in the *Romance* class, and hence romance books at a reasonable price will receive a high score.

Data Structures and Algorithms

The collection of books in the bookstore will be represented by a vector of pointers to type *Book*: vector<Book\*> books. When the *Search*() function “grades” all books, the collection of books will be sorted in a descending order according to their grades using the *std::sort()* function defined in the <algorithm> header file. It uses an algorithm known as Introsort, which is a hybrid between Quicksort, Heapsort, and Insertion Sort. Introsort begins with Quicksort and if the recursion depth exceeds 2logN (N – number of elements to be sorted), it switches to Heapsort so that Quicksort’s O(n^2) worst-case complexity is avoided. If N is small enough, it simply performs Insertion Sort (because for small enough N, Quicksort’s large constant factor leads to large time complexity). The result is that the *std::sort()* function has best, average, and worst case performance of O(NlogN).