**Treść zadania:**

**2.** Stwórz program, który będzie katalogował informacje o minerałach. Minerały poukładane są w działach o nazwach odpowiadających państwom, w których występują. Należy interakcyjnie wczytać elementy listy, której każda pozycja zawiera:

• nazwę minerału

• skład chemiczny

• nazwę działu

• numer identyfikacyjny

Dane nie mogą się powtarzać. Liczba katalogów jak i rekordów w każdym katalogu nie jest ograniczona (jest zmieniana w sposób dynamiczny w trakcie tworzenia katalogu). Nie można wprowadzać liczby porządkowej do podwójnego numerowania elementów listy. Program powinien reagować i być odporny na nielogiczne wprowadzanie danych. Program powinien umożliwiać użytkownikowi korektę wprowadzanych danych. Program powinien umożliwiać wyświetlenie danych w odpowiednich porządkach, a) według alfabetycznej kolejności działów, a w ramach danego działu według malejących numerów; b) według alfabetycznej kolejności działów, a w ramach danego działu według listy alfabetycznej nazw minerałów. Posortowaną listę zawierającą wszystkie pozycje wyświetlać w sposób czytelny dla użytkownika. Program powinien zawierać proste menu, w którym będzie można:

Wyjść z programu (przed zakończeniem programu należy poprosić użytkownika o potwierdzenie tej operacji)

• Powrócić do wprowadzania danych

• Poprawić wprowadzony element listy

• Wyświetlić posortowaną listę na dwa sposoby

Program powinien zawierać funkcję, dzięki której od początku będzie zawierał część danych, tak aby nie wprowadzać wszystkich pozycji w trakcie oddawania zadania na laboratorium

**3.** Dodanie do projektu drugiego następujących funkcjonalności:

-Zapisywanie informacji o minerałach do pliku.

-Wczytywanie informacji o minerałach z pliku.

Zapisywanie i wczytywanie ma działać dla trzech formatów: JSON, XML oraz txt, gdzie plik txt powinien mieć takie samo formatowanie jak postać wyświetlana dla użytkownika w programie (czyli postać czytelna dla człowieka). Dodatkowo kluczowym wymaganiem jest to, by wczytywanie mogło się odbywać nawet wtedy, kiedy są w plikach błędne wartości (np. powtarzające się id, albo cyfra w nazwie kraju), przy czym należy opracować odpowiednie mechanizmy radzenia sobie z takimi błędami (np. nadanie innego id, wraz z powiadomieniem o błędzie).

**Dokumentacja**:

Program przechowuje dane w formie listy list. Pierwsza lista jest listą działów, a każdy element tej listy zawiera wskaźnik do listy elementów znajdujących się w tym dziale

**Struktury:**

* *Record*   
  - przechowuje informacje o danym minerale oraz wskaźnik do następnego elementu listy
* *Section*   
  - przechowuje nazwę kraju, listę elementów które w nim występują oraz wskaźnik do następnego elementu listy

**Pliki:**

* list.c - funkcja main i główne funkcje programu
* assist.c - funkcje pomocnicze
* fileSupport.c - funkcje zapisujące i wczytujące dane z plików

**Funkcje:**

**list.c:**

* *unsigned int avblId(Section \*head)*- znajduje numer ID większy od wszystkich do tej pory użytych
* *bool inputRecord(Record \*m)*- prosi użytkownika o podanie wszystkich informacji o minerale i wywołuje funkcje wczytujące
* *Record fill(Record mineral, char name[], char country[], char formula[], unsigned int id)*- uzupełnia informacje przechowywane w elemencie jeżeli zostały one podane w funkcji *mockInput*
* *Record \*findPreviousRA(char toInsert[], Record \*head, bool \*makeNew)*- zwraca wskaźnik na element najpóźniejszy w kolejności alfabetycznej który jest wcześniejszy niż ten który ma zostać dodany, jeżeli napotka element o tej samej nazwie, modyfikuje zmienną *makeNew* aby informowała że nie należy tworzyć nowego elementu
* *Section \*findPreviousS(char toInsert[SIZE], Section \*head, bool \*makeNew)*   
  - analogicznie do *findPreviousRA*
* *void placeR(Record mineral, Record \*prevA, Record \*prevI) -* umieszcza dane w odpowiednim miejscu listy
* *bool insert(Record mineral, Section \*\*head)  
  -* dodaje nowy element do listy
* *void mockInput(Section \*\*head)*    
  - dodaje elementy, które będą znajdować się w liście przed ingerencją użytkownika
* *bool alter(Section \*\*head)   
  -* pozwala użytkownikowi na wprowadzenie numeru identyfikacyjnego elementu który chce poprawić, wypisuje wybrany element i usuwa go. Następnie wywołuje funkcję *insert* która dodaje poprawiony element
* *void print(Section \*\*head, Mode m)*  
  - wypisuje elementy listy w dwóch trybach:   
   - według alfabetycznej kolejności działów, a w ramach danego działu według  
  malejących numerów (ID)  
   - według alfabetycznej kolejności działów, a w ramach danego działu według listy alfabetycznej nazw minerałów (ALPHA)
* *void quit(Section \*\*head)   
  -* usuwa po kolei wszystkie elementy listy i zwalnia pamięć
* *void menu(Section \*\*head)*   
  - pozwala użytkownikowi na interakcję z programem

**assist.c:**

* *void cleanBuff(void)*- usuwa znaki które nie zostały użyte z bufora
* *bool legitChar(char a)*- zwraca prawdę, jeżeli znak jest literą, myślnikiem lub apostrofem
* *bool legitFileName(char a)*  
  - zwraca prawdę, jeżeli znak nie jest ukośnikiem lub spacją
* *int inputStr(char arr[], char comment[])*   
  - prosi użytkownika o podanie napisu i wczytuje go do tablicy podanej jako argument, zwraca 0 jeśli wczytywanie się nie powiodło, 1 jeżeli wczytywanie się powiodło i zostało przerwane w inny sposób niż przez znak nowej linii, 2 jeżeli wczytywanie się powiodło i zostało zakończone przez znak nowej linii
* *int finputStr(char arr[], bool allowNumbers, FILE\* filePointer)   
  -* wczytuje napis z pliku analogicznie do funkcji *finputStr*
* *bool inputInt(unsigned int \*n, char comment[])*- prosi użytkownika o podanie wartości int i zapisuje ją, modyfikując wskaźnik podany jako argument
* *void \*checkAlloc(void \*p)*   
  - jako argument dostaje wskaźnik który wcześniej został zadeklarowany przy pomocy funkcji malloc i sprawdza czy alokacja powiodła się
* *bool format(char a[])*- modyfikuje podaną tablice znaków tak, aby zawierała wyłącznie małe litery alfabetu
* *bool checkPointer(FILE\* const filePointer)*- zwraca prawdę jeżeli uchwyt do pliku działa poprawnie
* *void space(FILE\* filePointer, int n)*- wypisuje n spacji do pliku
* *char skip(FILE\* filePointer)*- podczas wczytywania danych z pliku, pomija puste wiersze; zwraca pierwszy napotkany niepusty znak

**fileSupport.c:**

* *void saveTxt(char filename[], Section \*\*head, Mode m)*- zapisuje dane do pliku txt
* *void saveJSON(char filename[], Section \*\*head)*- zapisuje dane do pliku JSON
* *void saveXML(char filename[], Section \*\*head)*- zapisuje dane do pliku XML
* *bool loadTxt(char filename[], Section \*\*head)*- wczytuje dane z pliku txt do bazy, pomijając te, które już się w niej znajdują
* *bool readChar(char a, FILE\* const filePointer)*- wczytuje znak z pliku i sprawdza czy jest zgodny z formatem pliku
* *bool readElementStrJSON(char key[], char element[], FILE\* const filePointer)  
   -* wczytuje linię zawierającą klucz oraz napis
* *bool readElementIntJSON(char key[], unsigned int \*element, FILE\* const filePointer)*- wczytuje linię zawierająco klucz oraz liczbę
* *bool loadJSON(char filename[], Section \*\*head)*- wczytuje dane z pliku txt do bazy, pomijając te, które już się w niej znajdują
* *bool readKeyXML(char key[], FILE\* const filePointer)*- wczytuje klucz i sprawdza czy jest on zgodny z formatem pliku
* *bool readElementStrXML(char key[], char element[], FILE\* const filePointer)*wczytuje linię zawierającą klucz oraz napis
* *bool readElementIntXML(char key[], unsigned int \*element, FILE\* const filePointer)*- wczytuje linię zawierającą klucz oraz liczbę
* *bool loadXML(char filename[], Section \*\*head)*- wczytuje dane z pliku txt do bazy, pomijając te, które już się w niej znajdują

**Działanie programu**

Po uruchomieniu programu zostaje uruchomiona funkcja *menu*, która pozwala użytkownikowi na wprowadzenie następujących instrukcji:

**1** - tworzy nowy element listy. Użytkownik podaje kolejno nazwę, skład chemiczny, kraj występowania oraz numer identyfikacyjny minerału. Jeżeli podane dane będą niepoprawne, program usunie niedozwolone znaki z napisu lub nada inny numer ID

**2** - poprawia dane które już znajdują się w bazie. Prosi użytkownika o podanie numeru elementu który ma zostać poprawiony, następnie wczytuje poprawione dane w taki sam sposób jak przy dodawaniu nowego elementu

**3** - wypisuje wszystkie elementy według alfabetycznej kolejności działów, a w ramach danego działu według listy alfabetycznej nazw minerałów

**4** - wypisuje wszystkie elementy listy według alfabetycznej kolejności działów, a w ramach danego działu według malejących numerów ID

**5** - zapisuje dane zgromadzone w bazie do pliku. Prosi użytkownika o wybór formatu pliku oraz podanie nazwy

**6** - wczytuje dane z pliku. Prosi użytkownika o podanie formatu pliku oraz nazwy

**7** - kończy program po uprzednim potwierdzeniu operacji