# Group 3: Nguyễn Đức Duy Nguyễn Công Hoàng Trần Quốc Bảo

Project X

## Programming Challenges

#### 1. Charge Account Validation Write a program that lets the user enter a charge account number. The program should determine if the number is valid by checking for it in the following list:

#### 5658845 8080152 1005231 4520125 4562555 6545231 7895122 5552012

#### 3852085 8777541 5050552 7576651 8451277 7825877 7881200 1302850

#### 1250255 4581002

#### The list of numbers above should be initialized in a single-dimensional array. A simple linear search should be used to locate the number entered by the user. If the user enters a number that is in the array, the program should display a message saying that the number is valid. If the user enters a number that is not in the array, the program should display a message indicating that the number is invalid. (1)

**#include <iostream>**

**using namespace** std;

**const int** ACCOUNTS[] = {

5658845, 4520125, 7895122, 8777541, 8451277, 1302850, 8080152, 4562555, 5552012, 5050552, 7825877, 1250255, 1005231, 6545231, 3852085, 7576651, 7881200, 4581002};

**int** main() {

**int** c;

cout << "Enter your charge account number:\t";

cin >> c;

**for** (**int** i = 0; i < 18; i++) {

**if** (ACCOUNTS[i] == c) {

cout << "The number is valid" << endl;

**return** 0;

}

}

cout << "The number is invalid" << endl;

**return** 0;

}

#### 2. Lottery Winners lottery ticket buyer purchases 10 tickets a week, always playing the same 10 5-digit “lucky” combinations. Write a program that initializes an array or a vector with these numbers and then lets the player enter this week’s winning 5-digit number. The program should perform a linear search through the list of the player’s numbers and report whether or not one of the tickets is a winner this week. Here are the numbers: (2)

#### 13579 62483 26791 77777 26792

#### 79422 33445 85647 55555 93121

**#include <iostream>  
const int** TICKETS[] = {

13579, 26791, 26792, 33445, 55555, 62483, 77777, 79422, 85647, 93121};

**using namespace** std;

**bool** isWinner(**int** \_assert) {

**for** (**int** t : TICKETS) {

**if** (\_assert == t)

**return true**;

}

**return false**;

}

**int** main() {

**const int** TRIES = 3;

**for** (**int** i = 0; i < TRIES; i++) {

cout << "Enter this week\'s winning ticket:\t";

**int** temp;

cin >> temp;

**if** (isWinner(temp)) {

cout << "You won!\n";

}

}

**return** 0;

}

## Case Study

#include <iostream>  
#include <iomanip>

#include <cmath>

#include <string>

**using namespace** std;

**const int** PRODUCTS = 9;

**enum** SORTED\_TYPE {

*TITLE*, *ID*, *PRICE*

};

SORTED\_TYPE currentlySorted = *ID*;

string PRODUCT\_TITLE[PRODUCTS] = {

"Six Steps to Leadership",

"Six Steps to Leadership",

"The Road to Excellence",

"Seven Lessons of Quality",

"Seven Lessons of Quality",

"Seven Lessons of Quality",

"Teams Are Made, Not Born",

"Leadership for the Future",

"Leadership for the Future"

};

string PRODUCT\_DESCRIPTION[PRODUCTS] = {

"Book",

"Audio CD",

"DVD",

"Book",

"Audio CD",

"DVD",

"Book",

"Book",

"Audio CD"

};

**int** PRODUCT\_NUMBER[PRODUCTS] = {

914, 915, 916, 917, 918,

919, 920, 921, 922

};

**double** PRODUCT\_PRICE[PRODUCTS] = {

12.95, 14.95, 18.95, 16.95,

21.95, 31.95, 14.95, 14.95, 16.95

};

**void** printAllProducts() {

cout << setw(30) << left << "Product Title"

<< setw(25) << "Product Description"

<< setw(20) << "Product Number"

<< setw(15) << "Unit Price"

<< endl;

**for** (**int** i = 0; i < PRODUCTS; i++) {

cout << setw(30) << left << PRODUCT\_TITLE[i]

<< setw(25) << PRODUCT\_DESCRIPTION[i]

<< setw(20) << PRODUCT\_NUMBER[i] << '$'

<< setw(14) << PRODUCT\_PRICE[i]

<< endl;

}

}

**void** parallelSwap(**int** from, **int** to) {

**if** (from == to)

**return**;

string tempTitle = PRODUCT\_TITLE[to];

string tempDesc = PRODUCT\_DESCRIPTION[to];

**int** tempNum = PRODUCT\_NUMBER[to];

**double** tempPrice = PRODUCT\_PRICE[to];

PRODUCT\_TITLE[to] = PRODUCT\_TITLE[from];

PRODUCT\_DESCRIPTION[to] = PRODUCT\_DESCRIPTION[from];

PRODUCT\_NUMBER[to] = PRODUCT\_NUMBER[from];

PRODUCT\_PRICE[to] = PRODUCT\_PRICE[from];

PRODUCT\_TITLE[from] = tempTitle;

PRODUCT\_DESCRIPTION[from] = tempDesc;

PRODUCT\_NUMBER[from] = tempNum;

PRODUCT\_PRICE[from] = tempPrice;

}

**void** sortByPrice() {

**for** (**int** i = 0; i < PRODUCTS; i++) {

**double** temp = PRODUCT\_PRICE[i];

**int** k = i;

**for** (**int** j = i; j < PRODUCTS; j++) {

**if** (PRODUCT\_PRICE[j] < temp) {

temp = PRODUCT\_PRICE[j];

k = j;

}

}

parallelSwap(i, k);

}

currentlySorted = *PRICE*;

}

**void** sortByName() {

**for** (**int** i = 0; i < PRODUCTS; i++) {

string temp = PRODUCT\_TITLE[i];

**int** k = i;

**for** (**int** j = i; j < PRODUCTS; j++) {

**if** (PRODUCT\_TITLE[j] < temp) {

temp = PRODUCT\_TITLE[j];

k = j;

}

}

parallelSwap(i, k);

}

currentlySorted = *TITLE*;

}

**void** sortByNumber() {

**for** (**int** i = 0; i < PRODUCTS; i++) {

**int** temp = PRODUCT\_NUMBER[i];

**int** k = i;

**for** (**int** j = i; j < PRODUCTS; j++) {

**if** (PRODUCT\_NUMBER[j] < temp) {

temp = PRODUCT\_NUMBER[j];

k = j;

}

}

parallelSwap(i, k);

}

currentlySorted = *ID*;

}

**int** binarySearch(**int** productID) {

**int** leftBound = 0, rightBound = PRODUCTS - 1,

cursor = (leftBound + rightBound) / 2;

**for** (;;) {

**if** (rightBound - leftBound <= 1

&& PRODUCT\_NUMBER[cursor] != productID)

**return** -1;

**if** (PRODUCT\_NUMBER[cursor] == productID) **return** cursor;

**else if** (PRODUCT\_NUMBER[cursor] > productID) {

rightBound = cursor;

cursor = (leftBound + rightBound) / 2;

} **else** {

leftBound = cursor;

cursor = **static\_cast**<**int**>(

ceil((leftBound + rightBound) / 2.0));

}

}

}

**int** binarySearch(**const** string &title) {

**int** leftBound = 0, rightBound = PRODUCTS - 1,

cursor = (leftBound + rightBound) / 2;

**for** (;;) {

**if** (rightBound - leftBound <= 1

&& PRODUCT\_TITLE[cursor] != title)

**return** -1;

**if** (PRODUCT\_TITLE[cursor] == title) **return** cursor;

**else if** (PRODUCT\_TITLE[cursor] > title) {

rightBound = cursor;

cursor = (leftBound + rightBound) / 2;

} **else** {

leftBound = cursor;

cursor = **static\_cast**<**int**>(

ceil((leftBound + rightBound) / 2.0));

}

}

}

**int** binarySearch(**double** price) {

**int** leftBound = 0, rightBound = PRODUCTS - 1,

cursor = (leftBound + rightBound) / 2;

**for** (;;) {

**if** (rightBound - leftBound <= 1

&& PRODUCT\_PRICE[cursor] != price)

**return** -1;

**if** (PRODUCT\_PRICE[cursor] == price) **return** cursor;

**else if** (PRODUCT\_PRICE[cursor] > price) {

rightBound = cursor;

cursor = (leftBound + rightBound) / 2;

} **else** {

leftBound = cursor;

cursor = **static\_cast**<**int**>(

ceil((leftBound + rightBound) / 2.0));

}

}

}

**void** printProduct(**int** productID) {

**int** index = binarySearch(productID);

**if** (index == -1) {

cout << "Unable to find Product with id " << productID << endl;

} **else** {

cout << setw(30) << left << "Product Title"

<< setw(25) << "Product Description"

<< setw(20) << "Product Number"

<< setw(15) << "Unit Price"

<< endl;

cout << setw(30) << left << PRODUCT\_TITLE[index]

<< setw(25) << PRODUCT\_DESCRIPTION[index]

<< setw(20) << PRODUCT\_NUMBER[index] << '$'

<< setw(14) << PRODUCT\_PRICE[index]

<< endl;

}

}

**void** printProduct(**double** price) {

**int** index = binarySearch(price);

**if** (index == -1) {

cout << "Unable to find Product with price $" << price << endl;

} **else** {

cout << setw(30) << left << "Product Title"

<< setw(25) << "Product Description"

<< setw(20) << "Product Number"

<< setw(15) << "Unit Price"

<< endl;

cout << setw(30) << left << PRODUCT\_TITLE[index]

<< setw(25) << PRODUCT\_DESCRIPTION[index]

<< setw(20) << PRODUCT\_NUMBER[index] << '$'

<< setw(14) << PRODUCT\_PRICE[index]

<< endl;

}

}

**void** printProduct(**const** string &title) {

**int** index = binarySearch(title);

**if** (index == -1) {

cout << "Unable to find Product with title " << title << endl;

} **else** {

cout << setw(30) << left << "Product Title"

<< setw(25) << "Product Description"

<< setw(20) << "Product Number"

<< setw(15) << "Unit Price"

<< endl;

cout << setw(30) << left << PRODUCT\_TITLE[index]

<< setw(25) << PRODUCT\_DESCRIPTION[index]

<< setw(20) << PRODUCT\_NUMBER[index] << '$'

<< setw(14) << PRODUCT\_PRICE[index]

<< endl;

}

}

**void** getMenuChoice(**int** &choice) {

cout << "\n\n\n\t\tBookstore Manager\t\t\n";

cout << "\t1. Print All Products.\n";

cout << "\t2. Search a product by id.\n";

cout << "\t3. Search a product by title.\n";

cout << "\t4. Search a product by price.\n";

cout << "\t5. Sort products by title.\n";

cout << "\t6. Sort products by id.\n";

cout << "\t7. Sort products by price.\n";

cout << "\t8. Exit program.\n\n";

cout << "Your choice:\t";

cin >> choice;

}

**int** main() {

**int** choice;

**do** {

getMenuChoice(choice);

// Each case content is wrapped inside a pair of brackets

// because we want each case variable to be independent and

// to be deleted in stack memory after each iteration.

**switch** (choice) {

**case** 1:

printAllProducts();

**break**;

**case** 2: {

// if not surrounded by a bracket

// compiler will throw an error

**int** id;

cout << "\n\nWhich Product do you want to search (id):\t";

cin >> id;

**if** (currentlySorted != *ID*) sortByNumber();

printProduct(id);

**break**;

}

**case** 3: {

string name;

cout << "\n\nWhich Product do you want to search (name):\t";

cin.ignore(256, '\n');

getline(cin, name);

**if** (currentlySorted != *TITLE*) sortByName();

printProduct(name);

**break**;

}

**case** 4: {

**double** p;

cout << "\n\nWhich Product do you want to search (price):\t";

cin >> p;

**if** (currentlySorted != *PRICE*) sortByPrice();

printProduct(p);

**break**;

}

**case** 5:

sortByName();

cout << "Sorted!\n";

**break**;

**case** 6:

sortByNumber();

cout << "Sorted!\n";

**break**;

**case** 7:

sortByPrice();

cout << "Sorted!\n";

**break**;

**case** 8:

cout << "Quitting!\n";

**break**;

**default**:

cout << "Something went wrong!\n" << endl;

}

} **while** (choice != 8);

**return** 0;

}