|  |  |
| --- | --- |
| **Wydział Elektroniki i Technik Informacyjnych Politechniki Warszawskiej**  Programowanie Obiektowe PROE | **Zadanie programistyczne 1**  **Grupa: 1E1**  **Wykonał:**  **Konrad Domian**  **Nr albumu 304148** |
| **Warszawa 04.05.2020** | Prowadzący:  mgr.Inż. Bączyk Marcin |

**1.Temat zadania**

Tematem zadania było stworzenie aplikacji „symulującej” sklepową kasę.

**2.Wstepna Specyfikacja Programu**

1.Jaki jest interfejs?

W pierwszej wersji program sterowany jest z zakresu terminalu, po uruchomieniu programu wyświetla się informacja o poprawnym bądź niewczytaniu listy produktów, następnie wyświetlane jest menu kasy w którym wymienione są funkcje:

-sprzedaż

-wymiana

-dodaj produkt

-sprawdź kod

-menadżer listy produktów

-zakończ sprzedaż

2.Jaki jest wynik działania programu?

Po wejściu w zakładkę sprzedaż program wyświetla informacje o poleceniu wprowadzenia kodu produktu, następnie prosi o ilość tego produkty i na samym końcu prosi o potwierdzenie poprawności wpisanych danych.

Po wejściu w zakładkę wymiana program wyświetla informacje o poleceniu wprowadzeniu numeru transakcji następnie podaje informacje o tym czy transakcja została odnaleziona, jeżeli tak to program w pliku dailyRaport zeruje wybrana wartość transakcji.

Po wejściu w zakładkę dodaj produkt do pliku produkty dodawana jest nowa pozycja

Po wejściu w zakładkę sprawdź kod wyświetla się informacja o wpisaniu kodu produktu i następnie wyświetlane są dane o produkcie i czy produkt o podanym kodzie istnieje.

Po wejściu w zakładkę menadżer listy produktów wyświetlana jest menu menadżera, w którym zawarte są polecenia wyświetl listę produktów, dodaj produkt do listy, usuń produkt z listy i wróć do głównego menu

Po wejściu wyświetl listę produktów wyświetlana jest całą zawartość pliku listOfProduct

Po wejściu w zakładkę usuń produkt wyświetlana jest informacja o poleceniu wprowadzenia kodu produktu następnie informacja o znalezieniu produktu, jeżeli produkt został odnaleziony wyświetlana jest informacja o produkcie wraz z poleceniem o potwierdzenie usunięcia produktu z pliku produkty

Po wejściu w zakładkę wróć do menu głównego uruchamia się destruktor klasy ProductListMenadzer.

3.Obiekty:

Produkt:

-id produktu (int)

-nazwa produktu (string)

-cena (int)

-stawka podatku VAT (jako PTU) (char)

-wyświetl produkt

Bill:

-Numer Rachunku (int)

-Łączny podatek Vat (double)

-Godzina wydania rachunku (string)

-Dodaj produkt do rachunku

-Wydrukuj Rachunek

-Wyświetl Menu Tworzenia Rachunku

-Pobierz Dane o wyborze submenu od użytkownika

-Zlicz łączny podatek Vat z rachunku

-Wydrukuj Rachunek dla klienta

DailyRaport:

-Nazwa dokumentu (string)

-łączny Podatek (double)

-Utarg (double)

-Vector Rachunków (vector<Bill>)

-Plik Raportu Dziennego(ofstream)

-Dodaj Rachunek do Raportu dziennego;

-Usuń Rachunek z Raportu dziennego

-Znajdź Rachunek

MainMenu:

-Opcje wyboru funkcji menu

MainMenuPrinter:

-wyświetl menu w wybranym przez programistę języku (polski/angielski)

MenuUserChooser:

-Pobiera od użytkownika dane o wyborze funkcji

ProductListManager:

-Dodaj Produkt do listy Produktów

-usuń Produkt z listy Produktów

-Znajdź Produkt po ID produktu

-Wyświetl Wszystkie produkty znajdujące się na liście

-Sprawdź poprawność wpisania stawki Vat

MenuProduktListManager:

-Wyświetla Menu

-Pobiera od użytkownika dane o wyborze pozycji w menu

Sale:

-Wyświetl Menu Sprzedaży

-Pobierz dane od użytkownika o wyborze pozycji w menu

-Wykonaj zwrot Rachunku

TakeFromUserData:

-pobiera od użytkownika dowolne dane

TakeTime:

-godzina (char array)

-Dzień (char array)

-pobiera z urządzenia aktualna godzinę, dzień

4. Przypadki użycia

Tworzone są obiekty DailyRaport, ProductListManager. Które odpowiadają za otwarcie pliku ProductList.txt i stworzenie pliku raportu dziennego. Następnie tworzony jest obiekt MainMenu i wywoływana jest metoda menuChooser. Metoda ta pobiera od użytkownika dane odnośnie do jakiej opcji w menu chce użyć:

Jeżeli wybierze funkcje zamknij program to uruchamia ją się poszczególne destruktory klas DailyRaport, ProductListManager, menuChooser, MainMenu. W destruktorach klas DailyRaport zapisywane są dane z pamięci ram do pliku DailyRaport i zamykany jest ten plik. Za to w destruktorze ProductListManager zapisywane z pamięci ram jest vector produktów i też zamykany jest ten plik.

Jeżeli jednak użytkownik wybierze zakładkę sprzedaż to stworzy się obiekt Sale następnie wywoła się metoda displayMenu wraz z metodą menuChooser. Metoda menuChooser ponownie poprosi użytkownika o podanie jej wyboru która z opcji wyświetloną przez metodę displayMenu użytkownik chce użyć. Jeżeli użytkownik wybierze zakładkę stwórz nowy rachunek do zostanie stworzony obiekt Bill a następnie zostanie wywołane metody displayBillMenu, billMenuChooser, taxesCounter i printBillForCoustomer. A jeżeli jednak użytkownik wybierze opcje wróć to menu to zostanie zniszczony obiekt Sale.

Jeżeli wybierze zwrot to zostanie stworzony obiekt Sale i wywołana metoda replacement.

Jeżeli wybierze opcje znajdź produkt to zostanie wywołana metoda obiektu productListMenadzer findProduct().

Jeżeli wybierze opcje Menadżer listy produktów to zostanie stworzony obiekt ProductManagementMenu i wywołane metody displayMenu i userChooser. Metoda userChooser pobiera od użytkownika dane odnośnie do której funkcji menadżera chce użyć i jeżeli wybierze:

Wyświetl list Produktów to zostanie wywołana metoda displayAllProduct

Dodaj Produkt to zostanie wywołana metoda addProductToList

Usuń Produkt to zostanie wywołana metoda removeProductFromList

Wróć do Menu Głównego to zostanie wywołany destruktor klasy productManagemtntMenu

5. Statyczna analiza typów i interfejsów poszczególnych modułów

Bill:

Zaprzyjaźniona z klasą DailyRaport

DailyRaport:

Metoda findBill zwraca zmienna bool odpowiadającą za to czy dany rachunek został znaleziony

MainMenuPrinter:

Zaprzyjaźniona z klasą MainMenu

ProductListManager:

Zaprzyjaźniona z klasą Bill

Metoda findProduct zwraca zmienną typu bool czy produkt został odnaleziony

Metoda corectPTU zwraca zmienna typu bool, jeżeli produkt ma prawidłowo o kreśloną stawkę VAT

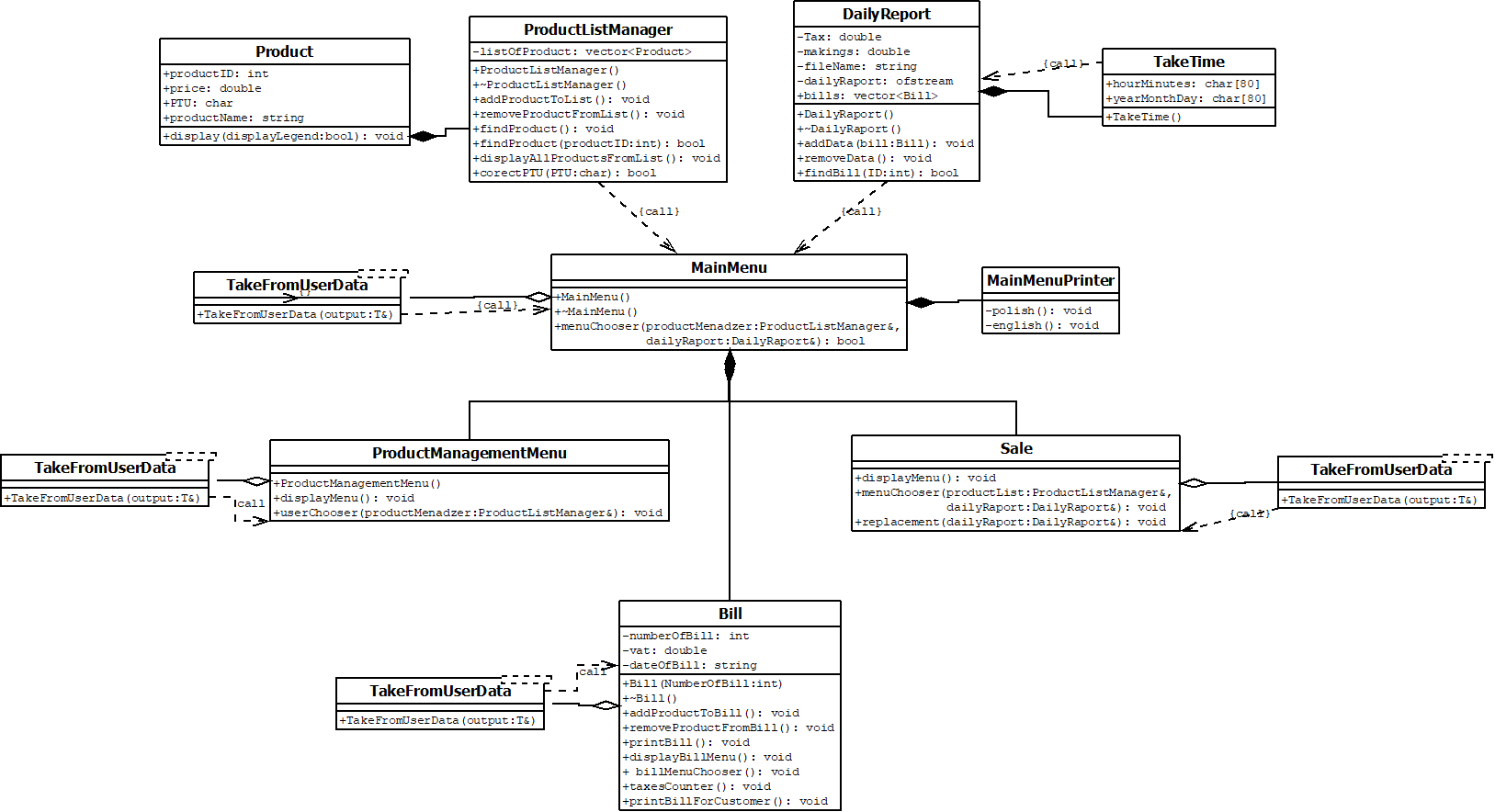
Sale:

Metoda menuChooser pobiera referencje do ProductListManager i do DailyRaport

Metoda replacement pobiera referencje do DailyRaport

TakeFromUserData:

Jest to template klasy

**2.Diagram obiektów**

**3.Opis najciekawszych fragmentów programu**

Pobranie danych z pliku za pomocą konstruktora klasy ProductListManager:

|  |
| --- |
| ProductListManager::ProductListManager()  {  std::string text;  std::ifstream productsListFile("productsList.txt", std::ios::in);  if (productsListFile.is\_open())  {  int ID = 1;  while (getline(productsListFile, text))  {  Product readProduct;  readProduct.productID = ID++;  std::istringstream iss(text);  std::vector<std::string> tokens;  copy(std::istream\_iterator<std::string>(iss), std::istream\_iterator<std::string>(), back\_inserter(tokens));  readProduct.price=std::stod(tokens[0]);  readProduct.PTU = tokens[1][0];  readProduct.productName = tokens[2];  listOfProduct.push\_back(readProduct);  }  }  else  {  std::cout << "Error cannot read a Products File" << std::endl;  exit(0);  }  productsListFile.close();  } |

W pierwszych liniach kodu otwieramy plik o nazwie productsList, następnie przez pętle while wczytujemy linia po linii aż do końca dokumentu dodatkowo zliczając każdą linie, ponieważ jak możemy zauważyć w następnej linii zaimplementowałem taką własność, że ID produktu odpowiada linia w dokumencie. Następnie tekst jest dzielony na mniejsze fragmenty po czym odpowiednie fragmenty są przypisywane do odpowiednich zmiennych.

Dodawanie nowego produktu do listy produktów:

|  |
| --- |
| void ProductListManager::addProductToList()  {  system("cls");  char PTU;  Product newProduct;  std::cout << "Type Price of the Product: ";  TakeFromUserData priceOfTheProduct(newProduct.price);\  for (;;)  {  std::cout << "Type PTU of the Productu: ";  TakeFromUserData PTUOfTheProduct(PTU);  if (corectPTU(PTU))  break;  else  std::cout << "HelpDesk! PTU Table: A-23% B-8% C-5% D-0% E-NULL\nYou can type only A,B,C,D,E " << std::endl;  }  newProduct.PTU = PTU;  std::cout << "Type nameOfTheProduct: ";  TakeFromUserData nameOfTheProduct(newProduct.productName);  newProduct.productID = listOfProduct.size() + 1;  std::cout << "Add ?" << std::endl;  newProduct.display(true);  std::cout << "(y/n) : ";  char yesOrNo;  TakeFromUserData userChoice(yesOrNo);  for (int i = 0; i < 1; i++)  {  switch (yesOrNo)  {  case 'y':  listOfProduct.push\_back(newProduct);  break;  case 'n':  break;  default:  std::cout << "only (t/n)" << std::endl;  TakeFromUserData userChoice(yesOrNo);  --i;  break;  }  }  } |

W pierwszej linii tekstu czyszczony jest terminal, aby nie przeszkodził odbiorcy w przeczytaniu danych jakie będą wyświetlane. Następnie jest tworzony obiekt TakeFromUserData, aby w sposób bezpieczny pobrać od użytkownika na pewno te dane na której nam zależy w tym przypadku jest to cena. Następnie jest wywoływana pętla for która wymusza od użytkownika prawidłowe podanie danych. Na samym końcu użytkownik jest proszony o potwierdzenie dodania produktu.

Destruktor klasy DailyRaport :

|  |
| --- |
| DailyRaport::~DailyRaport()  {  TakeTime time;  dailyRaport << "Report Day " << time.yearMonthDay<<"\n\n";  for (auto element : bills)  {  dailyRaport << "\n" <<"Bill Number: " << element.numberOfBill << "\n" <<"Date of Bill: " << element.dateOfBill << "\n"<<"Produts List:\n";  for (auto element : element.listOfProductBuyByConsumer)  {  dailyRaport << " " << element.productID << " " << element.productName << " " << element.PTU << " " << element.price <<" "<< "\n";  this->makings += element.price;  }  dailyRaport <<"Taxes :" <<std::fixed << std::setprecision(2) << element.vat << "$\n";  }  dailyRaport << "\n\n\nsummary: \n" << "makkings: " << this->makings<<"$\n all taxes:"<< this->Tax<<"$\n";  dailyRaport.close();  } |

Umieściłem ten fragment, aby zaprezentować użycie podwójnej pętli for w celu utworzenia raportu dziennego. Jak i wykorzystania SOLID.

Dodawanie produktu do rachunku:

|  |
| --- |
| void Bill::addProductToBill()  {  system("cls");  Product newRecordOnBill;  std::cout << "enter the product id: ";  int ID;  TakeFromUserData <int> IDs(ID);  ProductListManager productList;  if (productList.findProduct(ID))  {  std::cout << "How much?" << std::endl;  int howMuch;  TakeFromUserData numOfProduct(howMuch);  std::cout << "Add Product to bill? (y/n)" << std::endl;  char yesOrNo;  TakeFromUserData Trash(yesOrNo);  int i;  for (i=0; i < 1; i++)  switch (yesOrNo)  {  case 'y':  {  newRecordOnBill.price = productList.listOfProduct[ID - 1].price;  newRecordOnBill.productID = productList.listOfProduct[ID - 1].productID;  newRecordOnBill.PTU = productList.listOfProduct[ID - 1].PTU;  newRecordOnBill.productName = productList.listOfProduct[ID - 1].productName;  for(int j=0;j<howMuch;j++)  listOfProductBuyByConsumer.push\_back(newRecordOnBill);  break;  }  case 'n':  {  break;  }  default:  std::cout << "only (y/n): " << std::endl;  TakeFromUserData userChoice(yesOrNo);  --i;  break;  }  }  std::cout << "Your bill: " << std::endl;  this->printBill();  } |

Dodawanie produktu do rachunku odbywa się poprzez popranie przez obiekt takeUserData następnie sprawdzenia czy produkt o podanym ID istnieje po czym użytkownik proszony jest o podanie ilości, która tak samo wykorzystany jest obiekt takeUserData

Zliczanie podatku Vat dla rachunku:

|  |
| --- |
| void Bill::taxesCounter()  {  for (auto elements : listOfProductBuyByConsumer)  switch (elements.PTU)  {  case 'A':  {  vat = vat + (elements.price \* 0.23);  break;  }  case 'B':  {  vat += elements.price \* 0.08;  break;  }  case 'C':  {  vat += elements.price \* 0.05;  break;  }  case 'D':  {  vat += 0;  break;  }  case 'E':  {  vat += 0;  break;  }  default:  {  std::cout << "Error! Incorrectly given tax!" << std::endl;  elements.display(1);  break;  }  }  } |

Jako że chciałem uniknąć wprowadzania przez użytkownika dowolnej wartości podatku Vat to postanowiłem wykorzystać ustanowione przez Ministerstwo Finansów oznaczenia na Konkretne progi podatkowe, ale, żeby na „paragonie” ukazała się liczbowa wartość podatku potrzebna jest konwersja. Jednakowo chciałbym podkreślić jaka byłaby łatwość w dodaniu do rachunku możliwości pokazania konkretnych stawek VAT, ponieważ wystarczyłoby zastąpić zmienna VAT zmienna dla przykładu VAT.A, nie zostało do przez ze mnie na tym etapie to zaimplementowane, ponieważ nie było o tym mowy w specyfikacji początkowej.

Template klasy TakeFromUserData

|  |
| --- |
| template <typename T=int>  class TakeFromUserData  {  TakeFromUserData(T &output)  {  for (;;)  {  std::cin >> output;  if (std::cin.fail())  {  std::cin.clear();  std::cout << "Error! incorrect data" << std::endl;  std::cout << "Enter again data" << std::endl;  std::cin.ignore(std::numeric\_limits<T>::max(), '\n');  }  else  break;  }  }  }; |

Klasa TakeFromUserData pozwala na pobranie dowolnej zmiennej od użytkownika i za pomocą cin.fail sprawdzenia poprawności tej zmiennej na uwagę zasługuje wykorzystanie funkcji cin.ignore która ignoruje wszystkie znaki wpisane przez użytkownika do znaku spacji z uwzględniem tego, że różne zmienne mogą mieć rożną maksymalną wartość.

**4.Opis Najważniejszych klas i metod**

Bill:

Jest to klasa odpowiedzialna za rachunek w jej ciele znajdują się metody:

void addProductToBill();

void printBill();

void displayBillMenu();

void billMenuChooser();

void taxesCounter();

void printBillForCustomer();

Metoda addProduct przestawiona w poprzednim podpunkcie odpowiada za dodawanie do rachunku produktów.

Metoda printBill odpowiada za wirtualne „wydrukowanie” rachunku i zapisanie go do vectora rachunków będącego częścią klasy DailyRaport. Jest wywoływana wraz z metodą printBillForCustomer

Metoda printBillForCustomer tworzy plik, który możemy nazwać „wirtualnym paragonem” ponieważ jest to plik którego nazwa wygląda następująco godzinaUtworzeniaRachunku.txt w pliku zawarta jest lista produktów zawarta w rachunku, suma do zapłaty wraz z wyszczególnionym rachunkiem

Metoda taxesCounter zlicza podatek VAT za produkty zawarte w rachunku.

DailyRaport:

Jest to klasa odpowiedzialna za obsługę tworzenia Raportu dziennego w jej ciele znajdują się metody:

DailyRaport();

~DailyRaport();

void addData(Bill bill);

void removeData();

bool findBill(int);

W konstruktorze tworzony jest plik o nazwie „aktualnyDzień.txt”.

W metodzie addData dodawany jest obiekt bill do vectora dziennego raportu a dodatkowo jest dodawana suma Podatku Vat do łącznej sumy podatków.

W metodzie removeData jest usuwany rachunek odbywa się to przez wyszukanie rachunku o podanym przez użytkownika ID sprawdzeniu czy taki Rachunek istnieje i wyzerowanie go wraz z odjęciem jego Sumy Vat od łącznej sumy podatku. Poniżej znajduje się fragment kodu odpowiedzialny za tą metodę

|  |
| --- |
| system("cls");  int numberOfTheBill;  std::cout << "enter the bill id you want to delete: ";  TakeFromUserData <int> IDs(numberOfTheBill);  if (findBill(numberOfTheBill))  {  std::cout << "Remove ?" << std::endl;  bills[numberOfTheBill - 1].printBill();  std::cout << "(y/n) : ";  char yesOrNo;  TakeFromUserData <char> userChoice(yesOrNo);  for (int i = 0; i < 1; i++)  {  switch (yesOrNo)  {  case 'y':  {  bills[numberOfTheBill - 1].listOfProductBuyByConsumer.clear();  bills[numberOfTheBill - 1].dateOfBill.clear();  bills[numberOfTheBill - 1].numberOfBill = NULL;  Tax -= bills[numberOfTheBill - 1].vat;  bills[numberOfTheBill - 1].vat = NULL;  break;  }  case 'n':  break;  default:  std::cout << "only (t/n)" << std::endl;  TakeFromUserData <char> userChoice(yesOrNo);  --i;  break;  }  }  }  else  {  std::cout << "Error!! The bill does not exist"<<std::endl;  std::cout << "press any Key to countinue" << std::endl;  char trash;  TakeFromUserData Trash(trash);  } |

ProductListMenager:

Jest to klasa odpowiedzialna za obsługę listy produktów i pliku ListOfProduct.txt

W jej ciele znajdują się metody:

ProductListManager();

~ProductListManager();

void addProductToList();

void removeProductFromList();

void findProduct();

bool findProduct(int);

void displayAllProductsFromList();

bool corectPTU(char);

W konstruktorze Pobierane są dane z pliku ListOfProduct i przekazywane do vectora produktów.

Destruktor zapisuje dane z vectora produktów do pliku ListOfProdukt i zamyka plik ListOfProduct

Metoda addProductToList została już przestawiona w podpunkcie trzecim więc Pan pozwoli, że nie będę się powtarzać

Metoda removeProductFromList usuwa produkt z Listy Produktów przez wyzerowanie jego danych który następnie nie będzie zapisany do listy produktów przez kontrole w destruktorze tej klasy. Poniżej umieszczam fragment kodu odpowiedzialny za tą metodę:

|  |
| --- |
| system("cls");  int ID;  std::cout << "enter the product id you want to delete: ";  TakeFromUserData<int> IDs(ID);  std::cout << "Remove ?" << std::endl;  listOfProduct[ID - 1].display(true);  std::cout << "(y/n) : ";  char yesOrNo;  TakeFromUserData userChoice(yesOrNo);  for (int i = 0; i < 1; i++)  {  switch (yesOrNo)  {  case 'y':  listOfProduct[ID - 1].productID = NULL;  listOfProduct[ID - 1].price = NULL;  listOfProduct[ID - 1].PTU = NULL;  listOfProduct[ID - 1].productName = " ";  break;  case 'n':  break;  default:  std::cout << "only (t/n)" << std::endl;  TakeFromUserData userChoice(yesOrNo);  --i;  break;  }  } |

Metoda findProdukt jest przeciążona, ponieważ w przypadku, kiedy użytkownik chce tylko sprawdzić jaki Produkt znajduje się pod danym Produktem nie potrzebne jest, aby funkcja zwracała informacje w postaci zmiennej typu bool o istnieniu tego produktu. Poniżej fragment kodu:

|  |
| --- |
| system("cls");  std::cout << "enter the product id: ";  int ID;  TakeFromUserData IDs(ID);  if (ID > listOfProduct.size()|| ID - 1 < 0)  {  std::cout << "product with given id does not exist" << std::endl;  }  else  {  if(listOfProduct[ID-1].productID==0)  std::cout << "product with given id does not exist" << std::endl;  listOfProduct[ID-1].display(true);  }  std::cout << "press any Key to countinue" << std::endl;  char trash;  TakeFromUserData Trash(trash); |

Ale mimo wszystko ta metoda używana jest także do sprawdzenia przed dodaniem do rachunku czy taki produkt istnieje wtedy potrzebna jest metoda, która nie prosi użytkownika o podanie id a dodatkowo zwraca informacje czy taki produkt został odnaleziony. Poniżej fragment kodu:

|  |
| --- |
| system("cls");  bool productExist;  if (ID > listOfProduct.size() || ID - 1 < 0)  {  std::cout << "product with given id does not exist" << std::endl;  productExist = false;  }  else  {  if (listOfProduct[ID - 1].productID == 0)  std::cout << "product with given id does not exist" << std::endl;  listOfProduct[ID - 1].display(true);  productExist = true;  }  return productExist; |

Metoda displayAllProducts wyświetla wszystkie produkty które znajdują się w vetorze produktów. Nie zamieszczam kodu, ponieważ jest to bardzo prosta pętla for.

Metoda CorectPTU również została już przestawiona w podpunkcie 3.

**5.Instrukcja posługiwania się programem**

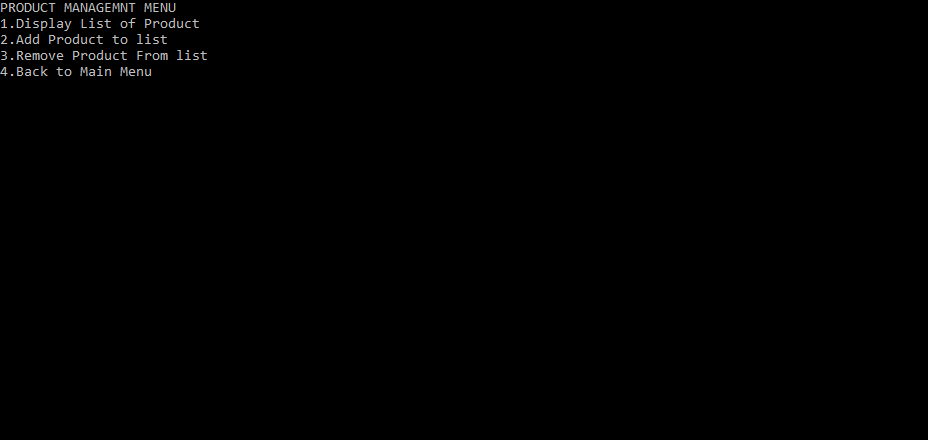
Po uruchomieniu Programu wyświetli się Główne Menu, aby wybrać jedną z opcji należy podać numer opcji a następnie zatwierdzić ją klikając enter. Wpisanie czegoś innego niż opcji zakończy się komunikatem o niepoprawnych danych i prośbą o ponowną próbę wyboru. Uwaga w programie progi podatkowe określone są literami A, B, C, D, E określające po kolei stawki (23% 8% 5% 0% jak i produkt zwolniony z podatku VAT). Próba wpisania liczbowej wartości przy dodawaniu nowego Produktu zakończy się błędem i komunikatem o ponowne dodanie danych wraz z prostą podpowiedzią.

**6.Przykład działania programu**

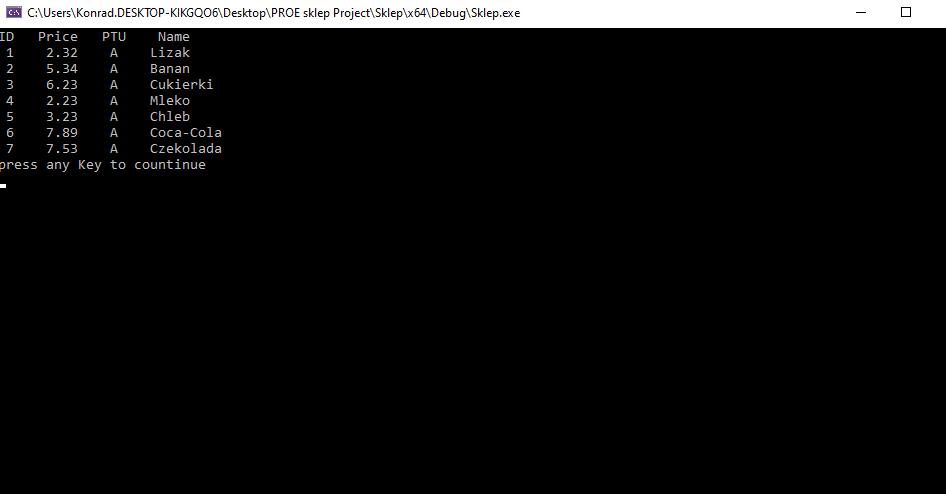
Menu Główne:



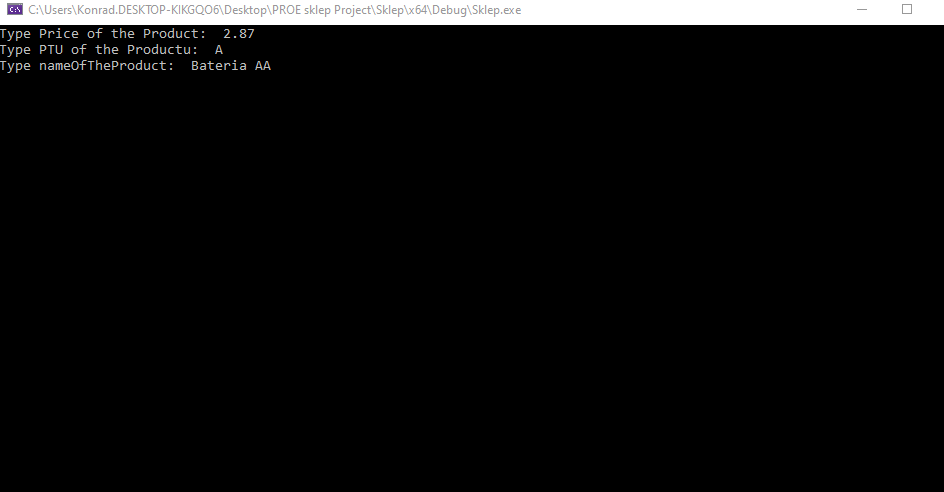
Po wejściu w Product Management:

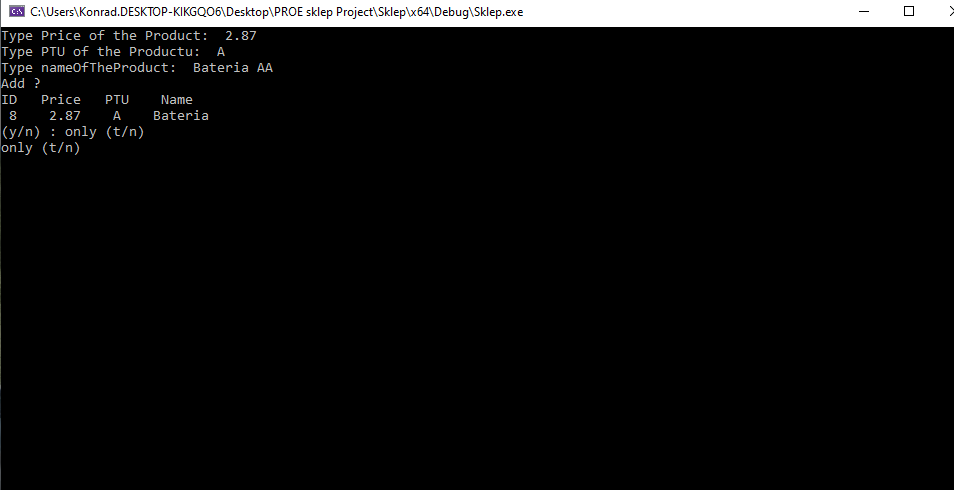


Wyświetlenie listy Produktów:



Dodawanie nowego Produktu do listy produktów:



Usuwanie Produktu z listy Produktów:



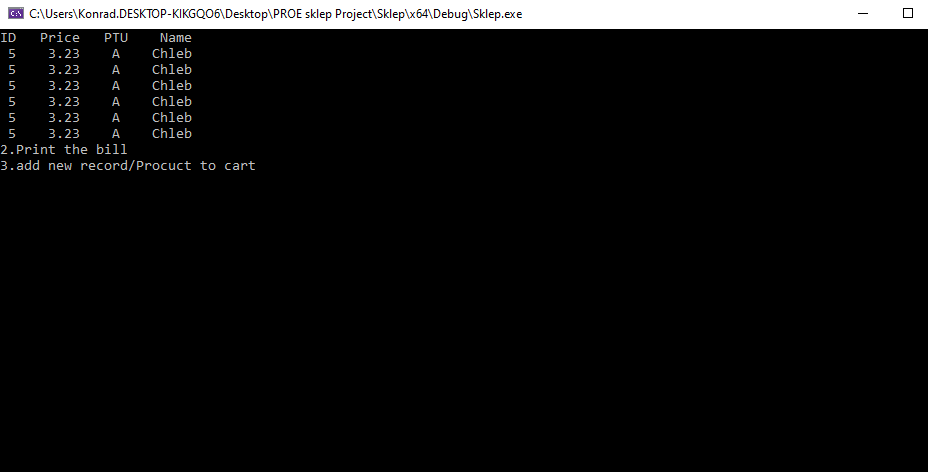
Wejście do menu Sprzedaż:



Utworzenie nowego Rachunku:

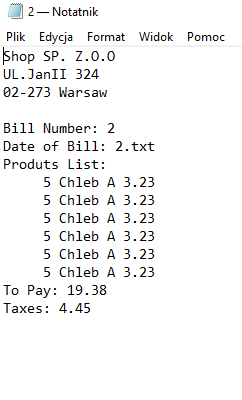


Dodanie produktu do rachunku:



Pliki powstałe w wyniku działania Programu:

Rachunek klienta kupującego na poprzednim screenshot-cie:



Raport dzienny:

