Gliwice, 01.02.2024

Programowanie I

**Projekt zaliczeniowy**

**"*Eksperyment z Win32*"**

**Kamil Bublij gr. 4/7**

1. **Opis projektu.**

Celem projektu było poznanie różnych technik programistycznych języka C++, w szczególności za pośrednictwem funkcjonalności dostępnych w środowisku Win32 i złożenie ich w spójną całość.

1. **Funkcjonalności.**

Zagadnienia poruszane w projekcie:

* elementy programowania obiektowego;
* biblioteka graficzna Direct2D, dokładniej DirectWrite;
* biblioteka graficzna GDI;
* dynamiczne zarządzanie pamięcią;
* kontenery (*map*, *set*, *string*, *vector*);
* wielowątkowość;
* szeroko rozumiane dialogi i kontrolki Win32;
* rozwinięty model rodzic-dziecko, w tym haki i dane dodatkowe okien;
* treści multimedialne (dźwięk, bitmapy);
* tworzenie przejrzystego interfejsu (menu, wstążki, skróty klawiszowe);
* weryfikacja danych wprowadzonych przez użytkownika;
* komendy preprocesora i konsolidatora;
* algorytmika (kodowania Huffmana).

1. **Przebieg realizacji.**

Program jest podzielony na mniejsze części – każda znacząco różniąca się funkcjonalnością część kodu ma osobny plik źródłowy i własny nagłówek.

**Wykaz plików:**

* *main.cpp, main.h* – główne pliki, zawierają podstawowe okno, instrukcje uruchamiania modułów i dialogu *O programie*;
* *Trigonometry.cpp, Trigonometry.h –* moduł rysujący funkcje okresowe;
* *GraphicsDemo.cpp, GraphicsDemo.h* – pobiera informacje   
  o możliwościach graficznych systemu użytkownika, po czym je wyświetla;
* *Tree.cpp, Tree.h* – pobiera informacje o tekście (lub parametrach kodowania), po czym na ich podstawie rysuje drzewo kodowania;
* *DWrite.cpp, DWrite.h –* pliki pomocnicze do obsługi biblioteki Direct2D;
* *Resources.rc, Resources.h –* pliki zasobów. Nie zawierają kodu, a elementy statyczne potrzebne programowi do działania;
* *tada.wav, (pliki osadzone w Resources.rc)* – pliki multimedialne utworzone przeze mnie, użyte za zgodą autora lub będące częścią domeny publicznej.

**Zastosowane algorytmy:**

W opisanych algorytmach będzie pojawiała się struktura zdefiniowana w kodzie jako *LEAF*:

TYPEDEF STRUCT LEAF{

UINT ID = NULL;

std::wstring tSymbol = {};

std::wstring tFPValue = {};

DOUBLE FPValue = 0;

UINT FFValue = 0;

STRUCT LEAF\* LeftChild = nullptr;

STRUCT LEAF\* RightChild = nullptr;

} LEAF, \* LPLEAF;

W dalszej części dokumentacji będę nazywał ją „liściem”.

* **tworzenie optymalnego drzewa kodowania Huffmana z kontenera liści**

Wstępne założenia: algorytm otrzymuje niepusty kontener, który zawiera wyłącznie liście. *Zaleca się*, żeby prawdopodobieństwa były znormalizowane (tutaj: suma *FPValue* po liściach równa 1), ale algorytm tego ani nie sprawdza, ani nie wymaga – inna część programu informuje użytkownika o problemie.

* 1. Jeśli w kontenerze pozostał jeden element, zakończ.
  2. Znajdź i wybierz dwa liście o najmniejszym prawdopodobieństwie.
  3. Utwórz nowy liść, który jako dzieci (tutaj: *LeftChild, RightChild*) będzie miał liście wybrane w kroku 2). Jego prawdopodobieństwo jest równe sumie prawdopodobieństw jego dzieci.
  4. Usuń liście z kroku 2) z listy.
  5. Umieść liść z kroku 3) w kontenerze.
  6. Wróć do 1).

W praktyce program tworzy drugą tablicę, która przechowuje częściowo przetworzone liście, a punkt 2) realizuje za pomocą sortowania tablicy   
i wybrania dwóch pierwszych elementów.

* **rysowanie drzewa binarnego**

Założenia: algorytm otrzymuje niepuste, **regularne** (każdy liść ma albo dwoje dzieci, albo żadnego) drzewo binarne.

1. Utwórz pusty stos przeznaczony na liście.
2. Ustaw się na najstarszym liściu (*korzeniu*).
3. Jeśli istnieje, narysuj lewe dziecko. Liść, na którym jesteś, wrzuć na stos z kroku 1) i ustaw się na jego lewym dziecku. Powtarzaj ten krok.
4. Dopóki jesteś prawym dzieckiem swojego rodzica, cofnij się do swojego rodzica (pierwszy element stosu z kroku 1)), po czym go usuń. Jeśli stos jest pusty, zakończ. Powtarzaj ten krok.
5. Trafiłeś do swojego rodzica z lewej strony, więc narysuj prawe dziecko i wrzuć je na stos z kroku 1). Wróć do kroku 3).

**Ważniejsze użyte funkcje bibliotek zewnętrznych:**

1. GDI:

**Pojęcia**: *kontekst urządzenia* – struktura przyznawana przez system w celu utworzenia jakiejś grafiki. Urządzeniem nie musi być monitor. Strukturę „obsadza się” obiektem (to **nie** obiekt jak w programowaniu obiektowym);   
*pędzel* – obiekt, którego biblioteka używa do rysowania grafiki (zawiera kolory, desenie itp.). Pędzel zna swoje współrzędne.

Jeśli nie podano inaczej, funkcje zwracają kod informujący o powodzeniu.

* + *Ellipse(), Rectangle()* – funkcje rysujące podstawowe kształty geometryczne. Przyjmują kontekst urządzenia i swoje wymiary.
  + *PolylineTo()* – funkcja rysująca ciągłą linię łamaną na podstawie podanych współrzędnych. Przyjmuje kontekst urządzenia, listę punktów i jej długość. Zmienia koordynaty pędzla.
  + *MoveToEx()* – zmienia współrzędne pędzla, niczego nie rysuje. Przyjmuje kontekst urządzenia, nowe współrzędne i opcjonalnie wskaźnik, gdzie funkcja zapisze, skąd się przesuwa.
  + *DrawTextEx()* – rysuje tekst. Przyjmuje kontekst urządzenia, wskaźnik do tekstu, jego długość, prostokąt, w którym będzie tekst i dwa argumenty ze szczegółami formatowania.

1. DirectWrite

**Pojęcia**: *fabryka* – obiekt generujący inne obiekty.

Jeśli nie podano inaczej, funkcje zwracają kod informujący o powodzeniu.

* *CreateTextFormat()* – tworzy parametry formatowania tekstu, które będą potem stosowane. Przyjmuje nazwę czcionki, opcjonalnie dodatkową nazwę kroju, informacje o: grubości, stylu (kursywa itp.), rozciągnięciu, rozmiar, język i wskaźnik do struktury, w której to zapisze.
* *DrawText()* – rysuje tekst. Przyjmuje wskaźnik do tekstu, długość do napisania, parametry formatowania, prostokąt, w którym ma być narysowany i pędzel.

Poza tym były użyte wyłącznie biblioteki standardowe i systemowe:

* *<array>, <cmath>, <string>, <map>, <array>, <vector>, <set>, <algorithm>*
* *<Windows.h>, <CommCtrl.h>, <strsafe.h>, <windowsx.h>, <winerror.h>, <DWrite.h>, <D2D1Helper.h>*

Mimo załadowania bibliotek, kilka plików trzeba było dołączyć ręcznie (z użyciem *#pragma*):

* *winmm.lib* – biblioteka do dźwięku
* *comctl32.lib* – bardziej zaawansowane kontrolki standardowe
* *dwrite.lib* – DirectWrite – dziecko D2D służące do wyświetlania czcionek
* *d2d1.lib* – Direct2D, służący do grafiki i nie tylko

1. **Instrukcja użytkownika.**

Program powinien być opatrzony ikoną, która, naturalnie, przedstawia kubek herbaty.

Użytkownik jest witany ekranem powitalnym, który mówi mu, co ma dalej zrobić.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

Ku górze ekranu znajduje się menu, z którego można wybrać kilka pozycji:

*Notatka*: Zgodnie z konwencją, wszystkie pozycje menu zakończone wielokropkiem (…) powodują otwarcie nowego okna.

* Plik:
  + Zakończ (*Alt + F4*) – kończy program.
* Opcje:
  + Tryb szybki (*F4*) – na żądanie włącza/wyłącza użycie biblioteki Direct2D.
  + **<tylko skrót klawiszowy>** (*Shift + F4*) – niszczy klasę Direct2D, zmniejszając zużycie pamięci około 10-krotnie.
* „Fajne rzeczy”:
  + Właściwości grafiki… - otwiera opisany wcześniej moduł *GraphicsDemo*.
  + Pokaz trygonometrii… (*Ctrl + T*) – otwiera moduł *Trigonometry*.
  + Drzewko… (*Ctrl + H*) – otwiera moduł *Tree*.
* Pomoc:
  + O programie… (*F1*) – wyświetla informacje o projekcie.

**Moduł *GraphicsDemo***

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Moduł nie zawiera menu. Okno reaguje na przewijanie paskiem i kółkiem myszy, przy zmianie rozmiaru paski dostosowują się odpowiednio. Mały moduł; nie zawiera funkcji innych niż standardowe *WndProc*.

**Moduł *Trigonometry***

Obraz zawierający tekst, linia, Wykres, diagram

Opis wygenerowany automatycznie

Moduł zawiera klasyczne menu oraz *wstążkę*, która zawiera suwaki, służące do zmiany odpowiednio prędkości i przyspieszenia przesuwania się funkcji.   
Co ciekawe, funkcje odpowiednio skalują się wraz ze zmianą rozmiaru ekranu.

Wzór funkcji *Pokazowa 1*:

Wzór funkcji *Pokazowa 2*:

Struktura menu:

* Funkcje:

(wszystkie opcje odpowiadają za przełączanie wyświetlania funkcji)

* + - Sinus
    - Cosinus -
    - ~~Cocosinus~~ - :)
    - Tangens
    - Cotangens
    - Pokazowa 1
    - Pokazowa 2

*Dialog „Zmień zakres…”*

Obraz zawierający tekst, zrzut ekranu, Małe i średnie samochody, Kotowate

Opis wygenerowany automatycznie

Dialog składa się z kilku podokien. Podokno *Wybór funkcji* pozwala wybrać funkcję spośród tych, które są narysowane (żeby zmienić parametry funkcji, należy ją włączyć). Podokno *Właściwości* pozwala na dostosowanie precyzji rysowania, koloru oraz grubości kreski funkcji. Żeby zmienić kolor, należy użyć przycisku *Zmień kolor*…, który otwiera dialog:

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, Wielobarwność

Opis wygenerowany automatycznie

Ważniejsze funkcje

* *DrawTrig()* – dynamicznie tworzy funkcje
* *AnimateTrig()* – odpowiada za animacje robiąc zrzuty ekranu (!)

Znane problemy

* Funkcja *AnimateTrig()* ma problemy z przewijaniem, kiedy część okna znajduje się poza ekranem. *Możliwa przyczyna*: część okna, która nie jest w danej chwili na ekranie, nie jest odświeżana, a nadal są z niej pobierane dane.
* Rozmiar wektora przechowującego współrzędne punktów na wykres jest zależny od rozmiaru okna i aktualizuje się z niewielkim, ale istniejącym opóźnieniem. Zbyt szybkie zmniejszenie rozmiaru okna powoduje zakończenie programu (operacje odczyt/zapis poza zakresem wektora). Choć naprawa błędu jest prosta, uważam nowy, ergonomiczny sposób na zamknięcie okna za bardziej przydatny.

**Moduł *Tree***

Nie ma tu menu, jest za to ramka z oknem ustawień:

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznie

W ramce można zmienić tryb pracy programu na „ręczny” (tabelka wartości) lub „automatyczny” (użytkownik podaje tekst, program robi resztę)

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznie

Niezależnie od wybranego trybu pracy, po wprowadzeniu wartości i wciśnięciu przycisku *OK* program wygeneruje optymalne drzewo kodowania Huffmana.

Edycja tabelki w trybie „ręcznym” – podwójne kliknięcie w dowolnym miejscu tabeli otworzy pole tekstowe, w którym można wpisać wymarzoną wartość.   
Po wciśnięciu Enter lub otwarciu innego pola tekstowego wartość zostanie zweryfikowana i, jeżeli uznana za poprawną, wprowadzona do tabelki.

Konflikty – gdyby tabelka była źle znormalizowana, występowało w niej niepoprawne prawdopodobieństwo (większe niż 1 bądź ujemne), pojawi się ikonka ostrzegawcza. Obraz zawierający tekst, diagram, zrzut ekranu, linia

Opis wygenerowany automatycznie

**Uwaga:** Program informuje o niepoprawnych parametrach drzewa, ale i tak będzie próbował wykonać polecenie w miarę możliwości. Odpowiedzialność za stan drzewa spoczywa na użytkowniku, program wydaje wszystkie potrzebne ostrzeżenia.

Ważniejsze funkcje

* *CreateTreeFromList()* – przerabia okno z listą na (niezoptymalizowane) drzewo
* *PrepareTree() –* zoptymalizuj istniejące drzewo
* *DrawLeaf() –* narysuj liść na podstawie współrzędnych
* *DrawTree() –* utwórz współrzędne na podstawie optymalnego drzewa
* *AuxiliaryEditProc()* – obsługuje pola tekstowe, które pojawiają się dynamicznie, a rodzic nie może przejąć komunikatu
* *VerifyAndProcessInput()* – przetwarza dane wprowadzone przez użytkownika i, jeśli uzna je za poprawne, aktualizuje listę
* *HandleConflicts()* – zajmuje się konfliktami w tabeli
* *ProcessAutoText()* – przetwórz pole tekstowe na drzewo

Znane problemy

* Przy bardzo dużych drzewach może zabraknąć ekranu i/lub część pól może być przycięta. Rozwiązaniem jest utworzenie przewijalnego podokna, jest to jeden z kierunków rozwoju projektu.
* Z powodów optymalizacji część drzewa, która zniknie z ekranu, już na niego nie wróci. Należy wówczas ponownie wygenerować drzewo, żadne inne dane nie uległy zmianie.

**Dialog *O programie***

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, Czcionka

Opis wygenerowany automatycznie

Przy uruchomieniu powinna odegrać się dołączona z programem ścieżka dźwiękowa. Jeżeli jej nie ma, uruchomi się dźwięk jak przy systemowym komunikacie *Informacja*. Nie zawiera dodatkowych funkcji.

**DWrite.cpp**

Ta część programu nie ma swojego okna, jej zadaniem jest wczytanie i przygotowanie do użycia biblioteki Direct2D. Tutaj jest użyte programowanie obiektowe.

Ważniejsze funkcje:

* *CreateIndependentResources()* – tworzy zasoby niezależne od sprzętu;
* *CreateDependentResources()* – utwórz zasoby pod konkretne urządzenie;
* *DiscardResources()* – zwalnia pamięć

**5. Podsumowanie i wnioski.**

Programowanie z użyciem WinAPI jest uciążliwe, jednak programy wykonane w ten sposób działają bardzo sprawnie i zużywają mniej pamięci. Samo środowisko nie ułatwia sprawy – jego część liczy sobie już dobre 40 lat; da się odczuć przeskok technologiczny w trakcie korzystania z co starszych funkcji tego samego *<Windows.h>*. Niestety, te przeskoki da się również odczuć w kodzie, ilość rzutowań niezbędna do poprawnego działania programu jest zdecydowanie wyższa niż w przypadku korzystania z bibliotek standardowych.

Mimo tego, zamierzam rozwijać projekt, zaczynając od następujących spraw:

* Naprawienie *znanych problemów*;
* Modernizacja użytych bibliotek graficznych – część używanej przeze mnie biblioteki GDI naprawdę liczy sobie 40 lat[[1]](#footnote-1);
* Zmiana mechanizmu rysowania funkcji – ten aż się prosi o problemy   
  z rozwojem;
* Dodanie wsparcia dla funkcji nie-okresowych;
* Intensywniejsze korzystanie z tabel tekstowych – ułatwi to ewentualny przyszły proces tłumaczenia programu na inny język;
* (daleka przyszłość) Zmiana platformy z Win32 na UWP;

1. [Byte Magazine Volume 08 Number 12 - Easy Software : Free Download, Borrow, and Streaming : Internet Archive](https://archive.org/details/byte-magazine-1983-12/page/n49/mode/2up?view=theater) [↑](#footnote-ref-1)