1. Treść zadania

Napisać szesnastkowy edytor plików. Program powinien wyświetlać szesnastkowe wartości kolejnych bajtów pliku, umozliwiać ich edycje i zapis, a także umożliwiać cofanie i ponawianie zmian.

2. Analiza problemu

Aby przechowywać zawartość pliku w formie liczb szesnastkowych, użyta zostanie struktura HexByte (1) wykorzystywana wcześniej podczas laboratorium. Przechowuje ona wartość bajtu w dwóch formach. Pierwsza to bezpośrednia wartość zero-jedynkowa, czyli zmienna typu *unsigned char* która przedstawia wartość odczytaną bezpośrednio z pliku wejściowego. Aby zrozumieć drugą formę należy zauważyć, że 8 bitów składające się na bajt możemy zapisać jako dwie cyfry szesnastkowe. Każdą z tych cyfr zapisujemy osobno do zmiennej *char* jako znak ASCII.

```
typedef struct _HexByte
 {                            // Bajt informacji zapisany jako:
   char text_form[2];         // w postaci dwóch cyfr ascii
   unsigned char bin_form;    // w postaci binarnej
 } HexByte;
```

1. Struktura HexByte.

Program zostanie podzielony na dwie części, czyli część nagłówkową zawierającą deklaracje funkcji oraz część zawierającą definicje funkcji.

Używanie funkcji przez użytkownika w interfejsie będzie możliwe poprzez wpisanie znaku symbolizującego wybraną opcję i potwierdzenie przyciskiem Enter. Program umożliwia wczytanie pliku o rozszerzeniu *.txt lub *.bin oraz umożliwia zapis w tych rozszerzeniach. Edycja zawartości jest dokonywana na odczytanej zawartości pliku.

Program zawiera krótką instrukcję użytkowania uruchamioną przez wpisanie znaku (wybór opcji) 'h'.

Program posiada zdolność cofania i przywracania zmian dokonywanych na przechowywanej zawartości. Służyć będzie do tego struktura Kolejka, która zawiera w sobie odnośnik do listy zmian oraz ilość przetrzymywanych obecnie zmian. Struktura Zmiana służy do przechowywania informacji o pojedynczej zmianie dokonanej na zawartości programu. Dlatego struktura ta posiada ciąg znaków poddany zmianie, pozycję na której był oraz typ akcji, który mówi nam czy zawartość ta była usuwana czy dodawana zanim trafiła do kolejki.

```
typedef struct _Zmiana
{
    char* ciag;
    int pozycja;
    int typ_akcji; //dod = 1 us = 0
    struct Zmiana* nast;
} Zmiana;

typedef struct _Kolejka
{
    Zmiana* do_kolejki;
    int ilosc_zmian;
} Kolejka;
```

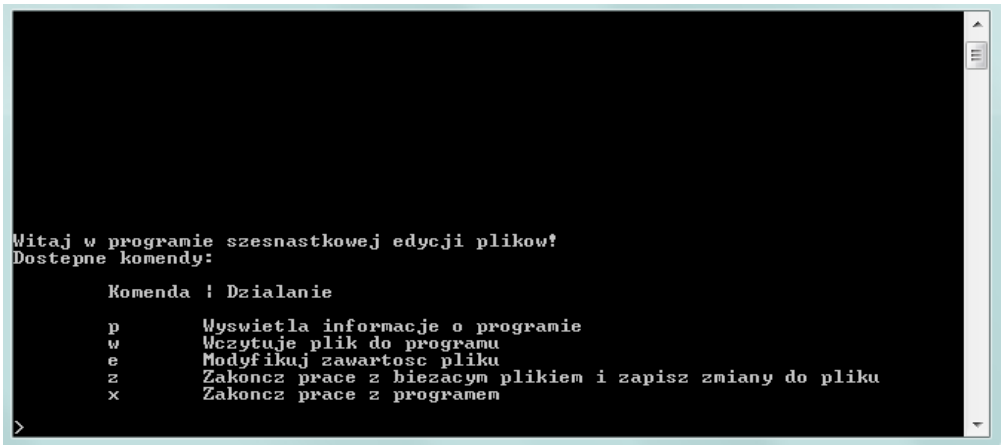
2. Struktury służące do przetrzymywania dokonanych zmian.

Struktury te są jak widać strukturami dynamicznymi. Wybór ten uzasadniony jest faktem elastyczności rozwiązania: Rozwiązanie jest bardzo efektywne ponieważ wykorzystujemy tylko tyle pamięci ile potrzeba w odróżnieniu od tworzenia i realokowania statycznych tablic zmian.

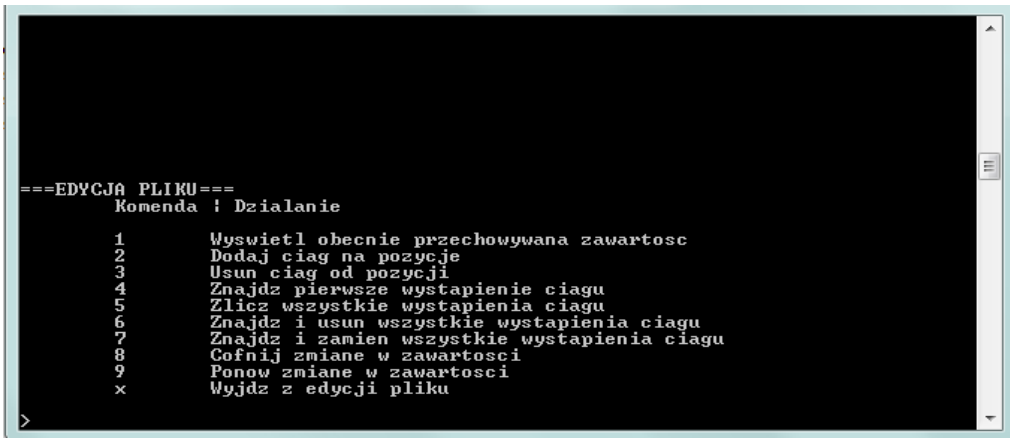
3. Specyfikacja zewnętrzna

1. Interfejs użytkownika

Program komunikuje się z użytkownikiem poprzez konsolę, wyświetlając dostępne opcje i potrzebne informacje. Program posiada dwa menu wyboru opcji, przedstawione poniżej. Użytkownik wybiera daną opcję, przyciskając klawisz powiązany z wybranym opisem funkcji i wciskając enter po wprowadzeniu symbolu.



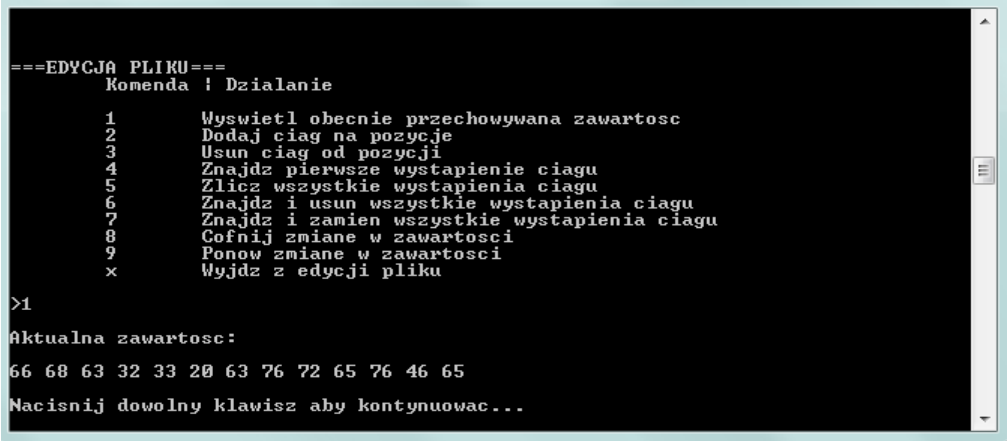
Menu główne programu



Menu edycji wczytanej zawartości

Pierwsze menu wyboru to menu główne programu. Tutaj użytkownik może wczytać zawartość, zapisać ją, lub wyświetlić tekst pomocy (opcja 'h'). Wybierając edycję wczytanej zawartości, program zabiera nas do menu po prawej, w którym wykonujemy wszystkie pożądane akcje modyfikacji zawartości, po czym możemy wrócić do menu głównego programu (symbol 'x') i zpaisać zmiany oraz zamknąć program, lub wczytać następny plik do zawartości programu.

Program pozwala użytkownikowi dodać ciąg symboli, usunąć go, znaleźć wszystkie wystąpienia lub tylko pierwsze wystąpienie ciągu symboli, znaleźć i usunąć ciąg symboli, znaleźć i zamienić ciąg symobli, oraz cofnąć lub ponowić zmianę w zawartości obecnie przetrzymywanej w programie.



Po wybraniu opcji, program wykona funkcje po czym wyświetli ponoiwnie menu wyboru

2. Cykl pracy programu

Program przyjmuje dane wejściowe we wczytywanych plikach. Aby wczytać plik do zawartości programu i aby móc na niej operować, wybierz opcję 'w' w menu głównym. Jeśli program nie wczytał poprzednio zawartości z pliku, lub zawartość została zapisana, program nie będzie pozwalał na edycję zawartości. Aby móc na czymś operować, trzeba przecież to – coś – mieć. Po wczytaniu zawartości, możemy wybrać opcję 'e' i oddać się właściwej pracy – czyli zmianie zawartości. Program wyświetla wartości jako wartości szesnastkowe, ale by uprościć użytkownikowi zadanie, wszelkie ciągi należy podawać jako zwykły ciąg symobli (czyli *„af5$”* zamiast *„45 39 F3 AB”*). Program zapyta nas o pozycję i ciąg jaki chcemy dodać usunąć lub znaleźć. Po zakończeniu edycji zawartości, wychodzimy z menu edycji opcją 'x' i zapisujemy zmiany do pliku.

Jeśli użytkownik wprowadzi ciąg znaków zamiast pojedynczego znaku w menu wyboru, pod uwagę zostanie wzięty tylko pierwszy symbol z ciągu i to on zadecyduje którą opcję należy uruchomić.

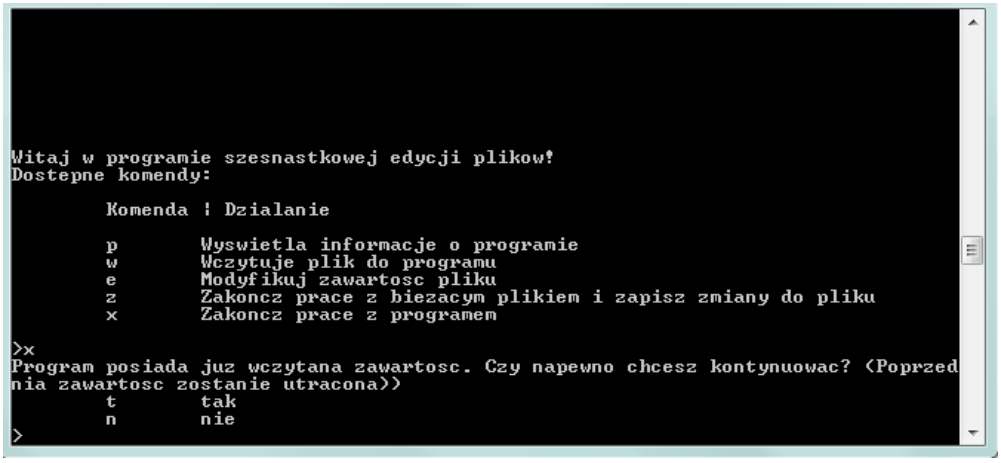
3. Komunikacja z użytkownikiem

Program napotykając wykroczenie poza cykl pracy (przedstawionym w poprzednim punkcie) poinformuje nas o niemożności wykonania takiego działania po czym zabierze nas z powrotem do ostatnio otwartego menu wyboru.

Możliwe błędy to:

- próba edycji zawartości przed wczytaniem zawartości do pliku
- próba wyłączenia programu z wczytaną i niezapisaną zawartością
- wprowadzenie błędnych symboli w menu wyboru
- próba ponowienia lub cofnięcia zmiany bez jej wcześniejszej egzystencji
- próba odczytu pliku który jest błędny

Program wyświetli krótki tekst pomocy w menu głównym po wyborze opcji 'h'.



Sygnalizacja wykrycia błędu. W tym przypadku użytkownik nie zapisał zawartości przetrzymywanej w programie, ale polecił zakończyć pracę programu. Wiąże się to z utratą zawartości, co wykrył program.

4. Specyfikacja wewnętrzna

Program jest podzielony na pliki. Początek każdego pliku z wyjątkiem *main.c* jest opatrzony dyrektywą *#pragma once*.

W pliku *main.c* zakodowana jest alokacja i dealokacja zmiennych, oraz pierwsza, główna pętla wyboru programu i druga, pętla edycyjna zawartości. Tutaj również następuje ustawienie wszystkich wymaganych pól zmiennych. W obu pętlach użyta została instrukcja *switch* do wywołania odpowiedniej funkcji w zależności od wyboru.

Ważniejsze zmienne

Typ zmiennej	Nazwa zmiennej	Przeznaczenie
HexByte**	Tab	Tablica dynamiczna zawartości. Do niej wczytywana jest treść pliku wejściowego i na niej dokonywane są operacje edycji zawartości. Każdy element HexByte odpowiada jednemu symbolowi z ciągu zawartości.
Kolejka*	kolejka_cofania	Kolejki służące do przechowywania zmian w zawartości.
Kolejka*	kolejka_ponawiania	
Int*	ilosc_znakow	Obecna ilość znaków w tablicy tab.

Zmienna *tab* jest podawana poprzez referencję, aby móc realokować tablicę *tab* podczas edycji zawartości.

Wywoływane funkcje i ich rola

```
int wczytaj_plik(HexByte***);
```

Odczytuje z konsoli nazwe pliku, sprawdza czy można go otworzyć, po czym ładuje bajt-po-bajcie plik wejsciowy uzywajac bufora typu unsigned char. Zwraca 0 gdy wczytywanie nie udało się.

```
int edytuj_zawartosc(HexByte***, int*, char*, Kolejka*, Kolejka*);
```

W zależności od symbolu przekazywanego w zmiennej char* uruchamia odpowiednią funkcję

edycji zawartości. Zwraca 0 gdy użytkownik zażąda przerwania edycji zawartości.

```
HexByte* znajdz_wystapienie(char*, HexByte**, int*, int, int);
```

Znajduje pojedyncze wystąpienie ciągu symobli w zawartości tablicy tab. Zwraca NULL gdy funkcja nie jest w stanie znaleźć następnego wystąpienia ciągu. Gdy znajdzie, zwraca początek znalezionego ciągu.

```
void usun_elementy_z_pozycji(HexByte***, int*, int, int, Kolejka*);
```

Usuwa ciąg o podanej długości z podanej pozycji. Pozostałą zawartość przesuwa aż luka po usuwanym ciągu nie zniknie. Dodaje usunięty ciąg do kolejki cofania.

```
void usun(HexByte***, char*, int*, Kolejka*);
```

Służy do wielokrotnego usuwania tego samego ciągu, szukając jego wystąpień oraz usuwając je.

```
void wstaw_ciag_na_pozycje(HexByte***, int*, int, char*, int);
```

Prosta funkcja wstawiania podanego ciągu na podaną pozycję, przesuwa obecną zawartość tworząc miejsce dla ciągu wstawianego.

```
void zamien_ciagi(char*, int, HexByte***, int, int, int*);
```

Zamienia podany ciąg z podanej pozycji na inny ciąg. Jeśli trzeba, rozszerzy lub usunie odpowiednio elementy tablicy tab.

```
int zamien(HexByte***, char*, char*, int*, Kolejka*);
```

Szuka i zamienia każde wystąpienie ciągu na drugi ciąg. Zwraca zawsze 0.

```
void dodaj_do_kolejki_cofania(Kolejka*, char*, int, int);
```

Dodaje do kolejki cofania Zmianę i wypełnia jej pola : zmieniony ciąg, pozycję oraz czy ciąg był usuwany czy wstawiany.

```
Zmiana* pobierz_z_kolejki(HexByte***, int*, Kolejka*, int);
```

Pobiera zmianę z kolejki. Ostatni parametr określa z której kolejki pobieramy zmianę. Jeśli pobieramy Zmianę z kolejki cofania, to chcemy odwrócić typ akcji zapisany w zmianie. (ponieważ wcześniej dodaliśmy ciąg, ale się rozmyśliliśmy – dlatego cofamy zmianę (czyli usuwamy ciąg). Teraz jednak chcemy by wróciła, więc chcemy ją ponownie dodać). Funkcja zabiera zmianę z kolejki i zwraca ją jako rezultat. Jeśli kolejka jest pusta, zwraca NULL.
Uwaga: Gdy usuwamy pamięć alokowaną dynamicznie, funkcja jako ostatni parametr otrzymuje wartość '2' co oznacza, że ma nie wprowadzać zmian do zawartości. Spowodowane to jest tym, że nikogo ta zawartość już nie interesuje, została ona zapisana i nie trzeba jej zmieniać.

```
void zapisywanie_posprzataj(HexByte***, int*, Kolejka*, Kolejka*);
```

Usuwa wszelkie zmiany z kolejek i usuwa zawartość.

```
int zapisz_plik(HexByte**, int*);
```

Sprawdza czy możemy zapisać do podanego pliku i zapisuje przechowywaną zawartość. Zwraca 1

jeśli zapisywanie się powiodło.

Funkcje komunikacji z użytkownikiem służą do komunikacji z użytkownikiem i najczęściej wypisują tylko na ekran komunikat. Dlatego nie zostały tu umieszczone.

Program deklaruje trzy nowe typy strukturalne: HexByte, Kolejka i Zmiana. Ich definicje zostały umieszczone w punkcie 2.

5. Testowanie

Wszystkie funkcje edycji zawartości zostały przetestowane i działają poprawnie. Program poprawnie prowadzi swój cykl pracy (zgłasza użytkownikowi błąd przy jego złamaniu).
Repozytorium zawiera przykładowe dane testowe.
Plik: dane.txt

Zawartość: fhc23 cvrevFe

Dodano na pozycji 5 ciąg „abab”
Usunięto z pozycji 4 jeden element
Znajdz i usun kazde wystapienie ciagu „e”
Usunięto z pozycji 3 5 elementów
Cofnięto zmianę.
Zapisano zawartość do pliku wyniki.txt

Zawartość po przetworzeniu: fhc2abab cvrvF

6. Wnioski

Zgodnie z przypuszczeniami, najwięcej czasu zabrało poprawianie błędów logicznych programu, gdzie jakaś zmienna powinna mieć inną wartość niż ma. Praca nad tym projektem poszerzyła znacznie moją wiedzę o programowanie w języku C i o jego najczęstszym zastosowaniu, czyli wykorzystaniu struktur połączonych wskaźnikami. Struktury cofania i ponawiania zmian okazały się prostsze w budowie i wykorzystaniu w stosunku do tego co wydawało się na początku.