

Wykład 8: Obsługa pliku.

dr inż. Andrzej Stafiniak

Wrocław 2023



HR EXCELLENCE IN RESEARCH



Politechnika Wroclawska

Urządzenia wejścia – wyjścia w języku C

Powtórka We-Wy

- Deklaracje większości funkcje obsługi **urządzeń wejścia/wyjścia** umieszczone są w pliku nagłówkowym **biblioteki standardowej** języka C - **stdio.h** (ang. *standard input/output*). Część funkcji (np. `getch()`, `putch()`) zdeklarowane są pliku **conio.h**, który nie jest częścią biblioteki standardowej.
- Jednym z dostępnych najczęściej stosowanych procedur obsługi operacji wejścia-wyjścia są **procedury strumieniowe** (czyli wysokiego poziomu, istnieją również procedury niskiego poziomu oraz obsługi portów we/wy).
- W **procedurach strumieniowych** pliki traktowane są jako ciąg, strumień bajtów, a obsługa operacji wejścia-wyjścia wymaga stosowania bufora. Buforowanie danych przy operacjach odczytu i zapisu przyspiesza działanie ponieważ zmniejsza ilość operacji na dysku.

Urządzenia wejścia – wyjścia w języku C

Powtórka We-Wy

- Podczas uruchomienia programu otwieraną są predefiniowane strumienie/kanały (pliki) między innymi:
 - **stdin** - standardowy strumień wejścia, domyślnie standardowe urządzenie wejściowe wczytuje znaki z klawiatury, ale strumień **stdin** może ulec przekierowaniu np. na plik czyli umożliwia zacytanie z pliku lub innego urządzenia,
 - **stdout** - standardowy strumień wyjścia, domyślnie standardowe urządzenie wyjściowe wyświetla dane na ekran, ale strumień **stdout** może ulec przekierowaniu np. na plik czyli umożliwia zapisanie do pliku,
 - **stderr** - standardowy strumień wyjściowy do obsługi błędów, domyślnie wysyła informacje na ekran, możliwe przekierowanie na plik.

Urządzenia wejścia – wyjścia w języku C

Powtórka We-Wy

- Podczas uruchomienia programu otwieraną są predefiniowane strumienie/kanały (pliki) między innymi:
 - **stdin** - standardowy strumień wejścia, domyślnie standardowe urządzenie wejściowe wczytuje znaki z klawiatury, ale strumień **stdin** może ulec przekierowaniu np. na plik czyli umożliwia zacytanie z pliku lub innego urządzenia
 - **stdout** - standardowy strumień wyjścia, domyślnie standardowe urządzenie wyjściowe wyświetla dane na ekran, ale strumień **stdout** może ulec przekierowaniu np. na plik czyli umożliwia zapisanie do pliku
 - **stderr** - standardowy strumień wyjściowy do obsługi błędów, domyślnie wysyła informacje na ekran, możliwe przekierowanie na plik.

wskazniki plikowe

Wykorzystując przekierowanie można wskazać inne pliki jako **pliki/strumienie standardowego wejścia/wyjścia**.

Podstawowe informacje o plikach

Plik (ang. file) jest to:

- pewien ciąg/sekwencja bajtów,
- umieszczonych na dowolnym nośniku pamięci,
- posiadających odrębną nazwę,
- które mogą być oddzielnie odczytywane lub modyfikowane.

Pliki mogą być otwarte z dwoma trybami:

- trybie **binarnym** – każdy bajt jest dosłownie rozumiany przez program.
- trybie **tekstowym** – zawartość pliku może się różnić od informacji przetwarzanych przez program, który obsługuje plik. Wynika to z faktu, że w standardzie ASCII, wyróżniamy **znaki drukowalne** (tekst) oraz **znaki sterujące** (obsługujące tekst).

Pliki **binarne** dedykowane są dla danych numerycznych. Informacje zapisane w trybie **binarnym**, nie da się wyświetlić w zrozumiały sposób prostym edytorem tekstu.

Na ilu bajtach liczba 22 222 może być zapisana w pamięci komputera w trybie **binarnym**, a jak w trybie **tekstowym** ?

Urządzenia wejścia – wyjścia w języku C

Wracając do procedur obsługi plików:

- w przypadku metod strumieniowych (wysokiego poziomu) funkcje obsługujące zaczynają się najczęściej od literki 'f' jak np.: `fopen()`, `fread()`, `fclose()` - identyfikatorem pliku jest wskaźnik na **FILE**.
- w przypadku metod niskiego poziomu mamy funkcję: `open()`, `read()`, `close()` - identyfikatorem pliku jest liczba całkowita.

Jak zostało już wspomniane metody strumieniowych wykorzystują bufor, natomiast metody niskopoziomowe wczytują/zapisują dane bezpośrednio z/do pamięci komputera.

Funkcje obsługi plików - **fopen()**

Nagłówek funkcji:

```
FILE * fopen(const char * filename, const char * mode)
```



Funkcja zwraca wskaźnik na utworzony lub otwarty pakiet danych czyli plik, inaczej **wskaźnik plikowy** (*file pointer*) – wskaźnik na typ pochodny **FILE**. W przypadku niepowodzenia utworzenia/otwarcia pliku zwraca wartość **NULL** (makro **NULL** o wartości 0).

FILE (struct **FILE**) *file handle*, pochodny typ danych, zadeklarowana w pliku nagłówkowym **stdio.h**. Zawiera informacje o pliku niezbędne do wykonania na nim operacji wejścia lub wyjścia, takie jak: deskryptor pliku (identyfikator), informacje o buforze, wskaźnik stanu buforowania strumienia, bieżąca pozycja strumienia, wskaźnik końca pliku i błędu.

Funkcje obsługi plików - **fopen()**

Nagłówek funkcji:

```
FILE * fopen(const char * filename, const char * mode)
```



Nazwa tworzonego/otwieranego pliku, może być podana jako stała znakowa zapisana między znakami "" np.: "fileName.txt" lub jako wskaźnik na tablicę przechowującą łańcuch znakowy.

Funkcje obsługi plików - **fopen()**

Nagłówek funkcji:

```
FILE * fopen(const char * filename, const char * mode)
```



Tryb **tekstowy/binarny** otwarcia/utworzenia pliku przekazany jako łańcuch znaków.

Tryb txt	Tryb binarny	Objaśnienie	Plik istnieje	Plik nie istnieje
"r"	"rb"	Odczyt	Odczyt od początku pliku	Błąd otwarcia
"w"	"wb"	Zapis	Usuwa zawartość pliku	Tworzy nowy plik
"a"	"ab"	Zapis	Dopisuje dane do końca pliku	Tworzy nowy plik
"r+"	"rb+"	Zapis/odczyt– uaktualnienie	Odczyt od początku pliku	Błąd otwarcia
"w+"	"wb+"	Zapis/odczyt – uaktualnienie	Usuwa zawartość pliku	Tworzy nowy plik
"a+"	"ab+"	Zapis/odczyt– uaktualnienie	Dopisuje dane do końca pliku	Tworzy nowy plik

Funkcje obsługi plików - **fclose()**

Nagłówek funkcji:

```
int fclose(FILE * fPtr)
```



Argumentem funkcji jest
wskaźnik plikowy.

Funkcja **fclose()** zamyka otwarty w programie plik oraz zwalnia załakowaną, przez funkcję `fopen()`, pamięć na potrzeby realizacji bufora. Funkcja zwraca wartość 0, jeśli została wykonana poprawnie oraz wartość ujemną EOF (end-of-file), w przeciwnej sytuacji.

Funkcje obsługi plików – **fopen()** , **fclose()**

Zabezpieczenie przed nieprawidłowym otwarciem lub zamknięciem pliku:

```
#include <stdio.h>
#include <stdlib.h>
#define NAME "c:\\MyProgsC\\fName.txt"

int main(){
    FILE *fPtr;

    fPtr = fopen("fName.txt", "w");
    if(fPtr == NULL){
        printf("Problem z otwarciem pliku \n");
        exit(EXIT_FAILURE);
    }
    ...
    ...

    if(fclose(fPtr) != 0){
        printf("Problem z zamknięciem pliku \n");
        exit(EXIT_FAILURE);
    }
    return EXIT_SUCCESS
}
```

```
// dla funkcji exit()

// fopen(NAME, "w");
// rownowazne if(!fPtr)

// rownowazne exit(-1)
```

Funkcja `exit()` zamyka program wykonując standardowe sprzątanie aktywnych zasobów (np. gwarantuje zamknięcie otwartych plików)

Funkcje obsługi plików

Funkcje obsługi plików standardowego wejścia-wyjścia w języku C możemy podzielić na dwie grupy:

➤ domyślne realizujące obsługę strumieni **stdin** (klawiatura) oraz **stdout** (ekran):

- we/wy znakowe (**getchar()** , **putchar()**)
- we/wy łańcuchowe (~~**gets()**~~ , **puts()**)
- we/wy sformatowane (**printf()** , **scanf()**)

➤ realizujące obsługę plików (dyskowe wejście-wyjście):

- we/wy łańcuchowe (**fgets()** , **fputs()**)
- we/wy sformatowane (**fprintf()** , **fscanf()**)
- we/wy rekordowe (**fread()** , **fwrite()**)
- we/wy znakowe (**getc()** , **putc()**)

Funkcje obsługi plików

Funkcje obsługi plików standardowego wejścia-wyjścia w języku C możemy podzielić na dwie grupy:

➤ domyślne realizujące obsługę strumieni `stdin` (klawiatura) oraz `stdout` (ekran):

- we/wy znakowe (`getchar()`, `putchar()`)
- we/wy łańcuchowe (~~`gets()`~~, `puts()`)
- we/wy sformatowane (`printf()`, `scanf()`)

Usunięte od C11

Ze względu że funkcja przyjmuje tylko jeden argument, adres bufora, istnieje możliwość przepełnienia bufora, lepiej skorzystać z funkcji `fgets()`.

➤ realizujące obsługę plików (dyskowe wejście-wyjście):

- we/wy łańcuchowe (`fgets()`, `fputs()`)
- we/wy sformatowane (`fprintf()`, `fscanf()`)
- we/wy rekordowe (`fread()`, `fwrite()`)
- we/wy znakowe (`getc()`, `putc()`)

np. `sizeof` bufor

```
char * fgets ( char * bufor, int size, FILE * stream );
```

np. `stdin`

Tablica char'ów realizująca bufor

Funkcje obsługi plików – `fprintf()` , `fscanf()`

Nagłówki funkcji:

```
int fprintf(FILE * stream, const char * format, ...)
```

```
int fscanf(FILE * stream, const char * format, ...)
```

Funkcje jako pierwszy argument przyjmują wskaźnik na plik, z/do którego zostanie zaczytana/zapisana sformatowana wartość.

Jaki będzie rezultat, gdy podstawimy wskaźniki `stdin` lub `stdout`?

Tekst sterujący ze specyfikatorami formatu, analogicznie jak w funkcjach `printf()`, `scanf()`:

%Specyfikator typu :

%c pojedynczy znak

%s łańcuch znaków

%d liczba dziesiętna ze znakiem

%f liczba zmiennoprzecinkowa (notacja dziesiętna)

%e liczba zmiennoprzecinkowa (notacja wykładnicza)

%u liczba dziesiętna bez znaku

%x liczba w kodzie szesnastkowym (bez znaku)

%o liczba w kodzie ósemkowym (bez znaku)

Kolejne argumenty

Funkcje obsługi plików – `fprintf()` , `fscanf()`

Nagłówki funkcji:

```
int fprintf(FILE * stream, const char * format, ...)
```

```
int fscanf(FILE * stream, const char * format, ...)
```



Funkcje jako pierwszy argument przyjmują wskaźnik na plik, z/do którego zostanie zaczytana/zapisana sformatowana wartość.

Co te funkcje zwracają?

Jaki będzie rezultat, gdy podstawimy wskaźniki `stdin` lub `stdout`?

```
printf("Yo, to działa tak");
```

```
fprintf(stdout, "Yo, to działa tak samo");
```

Zmodyfikowany przykład z instrukcji, zapis oraz odczyt do/z pliku tekstowego

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// #define NAME "c:\\MyProgsC\\fName.txt"

int main(){
    FILE *fPtr;
    fPtr = fopen("fName.txt", "w+");

    if(fPtr == NULL){
        printf("Problem z otwarciem pliku \n");
        exit(EXIT_FAILURE);
    }

    const char * text = "Ala ma kota";
    const unsigned int textLength = strlen(text);

    if(fprintf (fPtr , text) != textLength)
        printf ("Failed to write to the file \n");

    rewind(fPtr);
    char buffer[textLength + 1];

    if(fscanf (fPtr , "%s", buffer) != 1)
        printf("Failed to read from the file \n");
    else
        printf ("Data read from file : %s\n", buffer );

    if(fclose(fPtr)!=0){
        printf("Problem z zamknięciem pliku \n");
        exit(EXIT_FAILURE);
    }
    return EXIT_SUCCESS
}
```

```
// dla funkcji exit()
// dla funkcji strlen()

// fopen(NAME, "w+");
// rownowazne if(!fPtr)
// rownowazne exit(-1)

// stala lancuchowa
// dlugosc lancucha

// zapisanie tekstu do pliku

// wroc do poczatku pliku
// bufor (+1 na '\0')
```

scanf() oraz fscanf() domyślnie kończą pobieranie danych w momencie napotkania znaku białego (spacja, tabulator, itp.), więc efektem wywołania tego programu będzie wyświetlenie "Ala".

Funkcje obsługi plików – **fgets()** , **fputs()**

Nagłówki funkcji:

```
int fputs(const char * str , FILE * stream)
```

Funkcja **fputs()** zapisuje łańcuch znakowy (do napotkania `\0`) , zawarty w tablicy wskazywanej przez wskaźnik **str**, do pliku wskazywanego przez wskaźnik plikowy **stream** (**ale!** bez umieszczania znaku nowej linii tak jak to robi **puts()**).

Przy przypadku sukcesu zwracana jest 0, a w przeciwnym razie wartość `EOF`.

```
char * fgets(char * str, int num, FILE * stream)
```

Funkcja **fgets()** pobiera trzy argumenty: wskaźnik na tablice **str** do której mają zostać zapisane dane, maksymalna ilość znaków **num** do zaczytania oraz wskaźnik plikowy **stream**, z którego funkcja pobiera łańcuch znaków do napotkania `\n` (razem z `\n`) lub `EOF` (**Zaczytuje tylko linię!**). Na końcu zaczytanego łańcucha wstawia znak `\0`. W przypadku sukcesu funkcja zwraca wskaźnik na tablice czarów **str** , w przeciwnym wypadku zwracana jest wartość **NULL**.

Zmodyfikowany przykład z instrukcji, zapis oraz odczyt do/z pliku tekstowego

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// #define NAME "c:\\MyProgsC\\fName.txt"

int main(){
    FILE *fPtr;
    fPtr = fopen("fName.txt", "w+");

    if(fPtr == NULL){
        printf("Problem z otwarciem pliku \n");
        exit(EXIT_FAILURE);
    }

    const char * text = "Ala ma kota";
    const unsigned int textLength = strlen(text);

    if(fputs(text, fPtr) == EOF)
        printf ("Failed to write to the file \n");

    rewind(fPtr);
    char buffer[textLength + 1];

    if(fgets (buffer, textLength + 1, fPtr) == NULL)
        printf("Failed to read from the file \n");
    else
        printf ("Data read from file : %s\n", buffer );

    if(fclose(fPtr)!=0){
        printf("Problem z zamknięciem pliku \n");
        exit(EXIT_FAILURE);
    }
    return EXIT_SUCCESS;
}
```

// dla funkcji exit()
// dla funkcji strlen()

// fopen(NAME, "w+");
// rownowazne if(!fPtr)
// rownowazne exit(-1)

// stala lancuchowa
// dlugosc lancucha

// zapisanie tekstu do pliku

// wroc do poczatku pliku
// bufor (+1 na '\0')

Efektem zadziałania funkcji `fputs()` oraz `fgets()` jest zapisanie łańcucha znaków do pliku i odczytanie pełnego łańcucha "Ala ma kota" z pliku .

Zmodyfikowany przykład z instrukcji, zapis oraz odczyt do/z pliku tekstowego

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// #define NAME "c:\\MyProgsC\\fName.txt"

int main(){
    FILE *fPtr;
    fPtr = fopen("fName.txt", "w+");

    if(fPtr == NULL){
        printf("Problem z otwarciem pliku \n");
        exit(EXIT_FAILURE);
    }

    const char * text = "Ala ma \nkota";
    const unsigned int textLength = strlen(text);

    if(fputs(text, fPtr) == EOF)
        printf ("Failed to write to the file \n");

    rewind(fPtr);
    char buffer[textLength + 1];

    if(fgets (buffer, textLength + 1, fPtr) == NULL)
        printf("Failed to read from the file \n");
    else
        printf ("Data read from file : %s\n", buffer );

    if(fclose(fPtr)!=0){
        printf("Problem z zamknięciem pliku \n");
        exit(EXIT_FAILURE);
    }
    return EXIT_SUCCESS;
}
```

```
// dla funkcji exit()
// dla funkcji strlen()

// fopen(NAME, "w");
// rownowazne if(!fPtr)
// rownowazne exit(-1)

// stala lancuchowa
// dlugosc lancucha

// zapisanie tekstu do pliku

// wroc do poczatku pliku
// bufor (+1 na '\0')
```

W tym przypadku funkcja `fgets()` odczyta łańcuch razem ze znakiem `'\n'` i do tablicy `buffer[]` wstawi łańcuch `"Ala ma \n"` razem ze znakiem `\0` na końcu.

Zmodyfikowany przykład z instrukcji, zapis oraz odczyt do/z pliku tekstowego

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// #define NAME "c:\\MyProgsC\\fName.txt"

int main(){
    FILE *fPtr;
    fPtr = fopen("fName.txt", "w+");

    if(fPtr == NULL){
        printf("Problem z otwarciem pliku \n");
        exit(EXIT_FAILURE);
    }

    const char * text = "Ala ma \nkota";
    const unsigned int textLength = strlen(text);

    if(fputs(text, fPtr) == EOF)
        printf ("Failed to write to the file \n");

    rewind(fPtr);
    char buffer[textLength + 1];

    while(!feof(fPtr)){
        if(fgets (buffer, textLength + 1, fPtr) == NULL)
            printf("Failed to read from the file \n");
        else
            printf ("Data read from file : %s\n", buffer );
    }

    if(fclose(fPtr) != 0){
        printf("Problem z zamknięciem pliku \n");
        exit(EXIT_FAILURE);
    }

    return EXIT_SUCCESS;
}
```

Sprawdza czy
osiągnięto EOF

W tym przypadku program będzie
czytał liniami. Tablica buffer za każdym
razem będzie nadpisywana.

```
Data read from file : Ala ma
Data read from file : kota!
```

Zmodyfikowany przykład z instrukcji, zapis oraz odczyt do/z pliku tekstowego

```
#include <stdio.h>
#include <stdlib.h>           // dla funkcji exit()
#include <string.h>           // dla funkcji strlen()
#define NAME "c:\\MyProgsC\\fName.txt"

int main(){
    FILE *fPtr;
    fPtr = fopen("fName.txt", "w");           // fopen(NAME, "w");

    if(fPtr == NULL){                       // rownowazne if(!fPtr)
        printf("problem z zamknieciem pliku\n");
        exit(EXIT_FAILURE);                 // rownowazne exit(-1)
    }

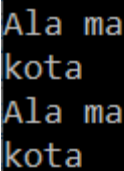
    const char * text = "Ala ma \nkota";      // stala lancuchowa
    const unsigned int textLength = strlen(text); // dlugosc lancucha

    if(fputs(text, fPtr ) == EOF)              // zapisanie tekstu do pliku
        printf ("Failed to write to the file \n");

    rewind(fPtr);                             // wroc do poczatku pliku
    char buffer[textLength + 1];               // bufor (+1 na '\0')

    int i=0;
    char c;
    while ( (c = fgetc(fPtr)) != EOF){
        buffer[i++]=c;
        fputc (c, stdout);
        //fputc (c, fPtr2);
    }
    buffer[i]='\0';
    printf("\n%s", buffer);

    if(fclose(fPtr) !=0){
        printf("Problem z zamknieciem pliku \n");
        exit(EXIT_FAILURE);
    }
    return EXIT_SUCCESS;
}
```



Ala ma
kota
Ala ma
kota

Po wskazaniu **stdin** będzie czytał z klawiatury do wciśnięcia *Ctrl Z* lub *Ctrl D*.

W tym przypadku program czyta znak po znaku z pliku wskazanego przez `fPtr` i zapisuje do tablicy oraz wysyła na standardowe wyjście **stdout**.

Może również zapisywać do innego wskazanego pliku.

Funkcje obsługi plików

```
int fsetpos( FILE * stream, fpos_t * pos )
```

```
int fgetpos( FILE * stream, fpos_t * pos )
```

```
int fseek( FILE * stream, long int offset, int whence )
```

```
long ftell( FILE * stream )
```

Razem mogą służyć do określenia np. rozmiaru pliku: `fseek()` służy do ustawiania wskaźnika na odpowiedniej pozycji w pliku np. na końcu, `fgetpos()`, `ftell()` służą do odczytywania, na którym miejscu znajduje się kursor, wskaźnik w pliku.

```

#include <stdio.h> // dla funkcji exit()

int main(void){
    FILE *fPtr;
    fpos_t pos1; //Typ danych -alias, używany do określania położenia danych w strumieniu
    fpos_t pos2 = 20LL; //typedef long long fpos_t;

    fPtr = fopen("fName.txt", "w+");
    fgetpos(fPtr, &pos1); // zapamiętanie pozycji kursora
    printf("Aktualna pozycja w pliku: %ld\n", pos1); // wyświetlenie pozycji kursora

    fprintf(fPtr, "To jest przykładowy tekst zapisany do pliku");

    fgetpos(fPtr, &pos1); // zapamiętanie aktualnej pozycji kursora
    printf("Aktualna pozycja w pliku: %ld\n", pos1);
    //
    if (fsetpos(fPtr, &pos2) == 0){ // ustawienie pozycji kursora
        fgetpos(fPtr, &pos1);
        printf("Aktualna pozycja w pliku: %ld\n", pos1);
    }
    else{
        printf("Błąd ustawiania kursora.\n");
        exit(EXIT_FAILURE);
    }
    fclose(fPtr);

    long int pozycja;
    fPtr = fopen("fName.txt", "r+");

    pozycja = ftell(fPtr); // zwróć aktualną pozycję kursora
    printf("Aktualna pozycja w pliku: %ld\n", pozycja);

    fseek(fPtr, -4L, SEEK_END); // ustaw kursor na pozycji koniec pliku - 4
    printf("Aktualna pozycja w pliku: %ld\n", ftell(fPtr));

    fseek(fPtr, 8L, SEEK_SET); // ustaw kursor na pozycji początek pliku + 8
    printf("Aktualna pozycja w pliku: %ld\n", ftell(fPtr));

    fseek(fPtr, 12L, SEEK_CUR); // ustaw kursor na pozycji obecnej + 8
    printf("Aktualna pozycja w pliku: %ld\n", ftell(fPtr));

    fclose(fPtr);
    return 0;
}

```

```

Aktualna pozycja w pliku: 0
Aktualna pozycja w pliku: 44
Aktualna pozycja w pliku: 20

```

```

Aktualna pozycja w pliku: 0
Aktualna pozycja w pliku: 40
Aktualna pozycja w pliku: 8
Aktualna pozycja w pliku: 20

```

```

#include <stdio.h> // dla funkcji exit()

int main(void){
    FILE *fPtr;
    fpos_t pos1; //Typ danych -alias, używany do określania położenia danych w strumieniu
    fpos_t pos2 = 20LL; //typedef long long fpos_t;

    fPtr = fopen("fName.txt", "w+");
    fgetpos(fPtr, &pos1); // zapamiętanie pozycji kursora
    printf("Aktualna pozycja w pliku: %ld\n", pos1); // wyświetlenie pozycji kursora

    fprintf(fPtr, "To jest przykładowy tekst zapisany do pliku");

    fgetpos(fPtr, &pos1); // zapamiętanie aktualnej pozycji kursora
    printf("Aktualna pozycja w pliku: %ld\n", pos1);
    //
    if (fsetpos(fPtr, &pos2) == 0){ // ustawienie pozycji kursora
        fgetpos(fPtr, &pos1);
        printf("Aktualna pozycja w pliku: %ld\n", pos1);
    }
    else{
        printf("Błąd ustawiania kursora.\n");
        exit(EXIT_FAILURE);
    }
    fclose(fPtr);

    long int pozycja;
    fPtr = fopen("fName.txt", "r+");

    pozycja = ftell(fPtr); // zwróć aktualną pozycję kursora
    printf("Aktualna pozycja w pliku: %ld\n", pozycja);

    if (fseek(fPtr, -4L, SEEK_END) == 0) //ustaw kursor na pozycji koniec pliku - 4
        printf("Aktualna pozycja w pliku: %ld\n", ftell(fPtr));
    else{
        printf("Błąd ustawiania kursora.\n");
        exit(EXIT_FAILURE);
    }

    fseek(fPtr, 12L, SEEK_CUR); // ustaw kursor na pozycji obecnej + 8
    printf("Aktualna pozycja w pliku: %ld\n", ftell(fPtr));

    fclose(fPtr);
    return 0;
}

```

```

Aktualna pozycja w pliku: 0
Aktualna pozycja w pliku: 44
Aktualna pozycja w pliku: 20

```

```

Aktualna pozycja w pliku: 0
Aktualna pozycja w pliku: 40
Aktualna pozycja w pliku: 8
Aktualna pozycja w pliku: 20

```


Odczyt i zapis do pliku w języku C++

W języku C++, w pliku nagłówkowym `<fstream>` zdefiniowane są obiektowe mechanizmy obsługi plików, podobnie jak w pliku `<iostream>` dla mechanizmy obiektowego wejścia-wyjścia `std::cin`, `std::cout`.

W języku C++ do obsługi plików stosuje się **klasy**:

- **`std::ofstream`** - obsługa wyjściowego strumienia plikowego)
- **`std::ifstream`** - obsługa wejściowego strumienia plikowego).

Utworzenie lub otwarcie pliku oraz jego zamknięcie można wykonać za pomocą metod (czyli funkcji danej klasy) **`open()`** oraz **`close()`**.

Odczyt i zapis do pliku w języku C++

Nagłówki metod:

```
//klasa std::ofstream
void open(const char * filename,
          std::ios_base::openmode mode = std::ios_base::out)

//klasa std::ifstream
void open(const char * filename,
          std::ios_base::openmode mode = std::ios_base::in)
```

Metoda **open()** przyjmuje dwa argumenty: łańcuch znaków z nazwą pliku (ścieżka do pliku) oraz tryb otwarcia pliku.

Tryb otwarcia pliku domyślnie ustawiony jest dla metody z klasy `std::ofstream` na plik do zapisu, a dla metody z klasy `std::ifstream` na plik do odczytu.

Odczyt i zapis do pliku w języku C++

Tryby otwarcia pliku:

Tryb	Objaśnienie
<code>std::ios_base::app</code>	dopisywanie do pliku
<code>std::ios_base::binary</code>	otwarcie pliku w trybie binarnym
<code>std::ios_base::in</code>	otwarcie pliku do odczytu
<code>std::ios_base::out</code>	otwarcie pliku do zapisu
<code>std::ios_base::trunc</code>	nadpisywanie pliku
<code>std::ios_base::ate</code>	przesunięcie wskaźnika położenia w strumieniu na koniec pliku

Tryby otwarcia pliku można łączyć wykorzystując operator **|**

Tryb w języku C++	Tryb w języku C
<code>std::ios_base::in</code>	<code>"r"</code>
<code>std::ios_base::out</code> oraz <code>std::ios_base::out std::ios_base::trunc</code>	<code>"w"</code>
<code>std::ios_base::out std::ios_base::app</code>	<code>"a"</code>
<code>std::ios_base::in std::ios_base::out</code>	<code>"r+"</code>
<code>std::ios_base::in std::ios_base::out std::ios_base::trunc</code>	<code>"w+"</code>
<code>std::ios_base::binary</code>	<code>"b"</code>

Odczyt i zapis do pliku w języku C++

Metoda `close()` jest bezargumentowa. Wywołać ją można na obiekcie klasy `ofstream` czy `ifstream` za pomocą operatora dostępu do składowych `–'`.

```
void close(void);  
obiekt.close();
```

Przesyłanie danych do wyjściowego strumienia plikowego może być realizowany podobnie jak w przypadku obiektu `std::cout`, za pomocą przeciążonego operatora przesunięcia bitowego `<<`.

```
obiekt << "Tekst do zapisania w pliku";
```

Istnieje ponadto metoda `is_open()` lub `good()`, która sprawdza czy plik został poprawnie otwarty, i w takim przypadku zwraca wartość `true`.

```
bool good(), bool is_open()  
obiekt.is_open();
```

Odczyt i zapis do pliku w języku C++

Przykład zapisu do pliku z instrukcji.

```
#include <iostream>
#include <fstream>
#include <cstdlib>

int main() {
    std::ofstream file;
    file.open("data.txt"); // Otwarcie pliku

    if(!file.is_open()) {
        std::cout<<"Failed to open the file"<<std::endl;
        exit(EXIT_FAILURE);
    }

    file << " Grades :" <<std::endl // Wpisanie danych do pliku
    <<"Jan Kowalski | " <<2.0<<std::endl
    <<"Adam Nowak | " <<4.5<<std::endl
    <<"Robert Malinowski | " << 5.5<<std::endl;

    file.close(); // Zamknięcie pliku
    return EXIT_SUCCESS ;
}
```

Odczyt i zapis do pliku w języku C++

Przesyłanie danych z pliku do wejściowego strumienia plikowego (obsługującego obiekty klasy `std::ifstream`) może być realizowany podobnie jak w przypadku obiektu `std::cin`, za pomocą przeciążonego operatora przesunięcia bitowego `>>`.

```
obiekt >> zmiennaJakiegoSTypu;
```

Do dyspozycji są jeszcze dwie przydatne metody do sprawdzania stanu odczytu: `eof()` sprawdza czy napotkano znak końca pliku (EOF) oraz `fail()`, zwraca wartość `true`, jeżeli odczyt pliku został przerwany w wyniku błędu.

```
bool eof(), bool fail()
```

```
obiekt.eof(), obiekt.fail()
```

Odczyt i zapis do pliku w języku C++

```
#include <iostream>
#include <fstream>
#include <cstdlib>
```

Przykład zapisu do pliku z instrukcji.

```
int main(){
    std::ifstream file;
    file.open("oceny.txt"); // Otwarcie pliku

    if(!file.is_open()){
        std::cout<<"Failed to open the file"<<std::endl;
        exit(EXIT_FAILURE);}

    float ocena, suma;
    unsigned int count = 0;

    while(file >> ocena){ // Wykonuj dopóki odczyt się powiodł i nie napotkano EOF
        ++count;          // Czyta do napotkania białego znaku, inna możliwość:
        suma += ocena;    file.getline(buffer , size)

    if (!file.eof() && file.fail()) // Sprawdzenie błędu odczytu
        std::cout << "Error occurred during reading the file" << std::endl;

    if(count!=0) //Obliczenie sredniej ocen
        std::cout << "The group average is: " << suma/count<< std::endl ;
    else
        std::cout << "No data read "<< std::endl ;

    file.close(); // Zamknięcie pliku
    return EXIT_SUCCESS ;
}
```

Odczyt i zapis do pliku w języku C++

Funkcja `getline()`, zaszyta wewnątrz klasy `fstream`.

```
istream & getline(char * buffer, streamsize n);
```

```
std::fstream fPtr("fName.txt", std::ios::in); //jeśli plik istnieje
char buffer[255];
fPtr.getline(buffer, 255); //wczytanie jednego wiersza danych (lub
//części wiersza jeśli się nie zmieści)
```

```
std::cout << "Podaj dane do wprowadzenia: ";
char buffer[255];
std::cin.getline(buffer, 255);
```