Politechnika Białostocka
Wydział Elektryczny
Katedra Elektrotechniki Teoretycznej i Metrologii

Instrukcja
do pracowni specjalistycznej z przedmiotu

Informatyka 2

Kod przedmiotu: **EZ1C300 014**

(studia niestacjonarne)

ZAAWANSOWANE OPERACJE WEJŚCIA-WYJŚCIA, PLIKI TEKSTOWE I BINARNE W JĘZYKU C

Numer ćwiczenia

INF22Z

Autor:
dr inż. Jarosław Forenc

Białystok 2016

Spis treści

Materiały dydaktyczne przeznaczone dla studentów Wydziału Elektrycznego PB.

1. Opis stanowiska

1.1. Stosowana aparatura

Podczas zajęć wykorzystywany jest komputer klasy PC z systemem operacyjnym Microsoft Windows (XP/Vista/7).

1.2. Oprogramowanie

Na komputerach zainstalowane jest środowisko programistyczne Microsoft Visual Studio 2008 Standard Edition lub Microsoft Visual Studio 2008 Express Edition zawierające kompilator Microsoft Visual C++ 2008.

2. Wiadomości teoretyczne

2.1. Strumienie

Operacje wejścia-wyjścia nie są elementami języka C. Zostały zrealizowane jako funkcje zewnętrzne, znajdujące się w odpowiednich bibliotekach dostarczanych wraz z kompilatorem. Funkcje te pozwalają na wykonywanie operacji wejścia-wyjścia w różny sposób i na różnym poziomie. Najczęściej do tego celu wykorzystuje się **strumienie**.

Strumień (ang. *stream*) jest pojęciem abstrakcyjnym. Jego nazwa bierze się z analogii między przepływem danych, a np. wody. W strumieniu dane płyną od źródła do odbiorcy. Zadaniem użytkownika jest określenie źródła i odbiorcy, wybranie typu danych oraz sposobu ich przesyłania. Strumień może być skojarzony ze zbiorem danych na dysku (np. plik) lub zbiorem danych pochodzących z urządzenia znakowego (np. klawiatura). Niezależnie od fizycznego medium, z którym strumień jest skojarzony, wszystkie strumienie mają podobne właściwości.

W języku C strumienie reprezentowane są przez zmienne będące wskaźnikami do struktur typu **FILE**. Definicja struktury **FILE** znajduje się w pliku nagłówkowym **stdio.h**:


```
struct _iobuf {
        char *_ptr;
        int   _cnt;
        char *_base;
        int   _flag;
        int   _file;
        int   _charbuf;
        int   _bufsiz;
        char *_tmpfname;
        };
typedef struct _iobuf FILE;
```

Podczas pisania programów nie ma potrzeby bezpośredniego odwoływania się do pól powyższej struktury.

Gdy program w języku C rozpoczyna działanie automatycznie tworzone są i otwierane trzy standardowe strumienie wejścia-wyjścia:

- **stdin** - standardowe wejście, skojarzone z klawiaturą;
- **stdout** - standardowe wyjście, skojarzone z ekranem monitora;
- **stderr** - standardowe wyjście dla komunikatów o błędach, domyślnie skojarzone z ekranem monitora.

Definicje standardowych strumieni umieszczone są w pliku nagłówkowym **stdio.h**:

```
_CRTIMP FILE * __cdecl __iob_func(void);

#define stdin  (&__iob_func()[0])
#define stdout (&__iob_func()[1])
#define stderr (&__iob_func()[2])
```

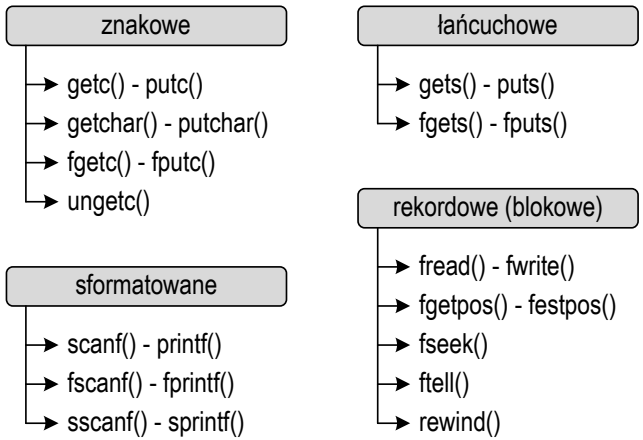
W dotychczas pisanych programach korzystano już z tych strumieni. Podczas każdego wywołania funkcji **printf()** niejawnie wykorzystywany był strumień **stdout**, a podczas wywołania funkcji **scanf()** - strumień **stdin**.

2.2. Typy operacji wejścia-wyjścia

Operacje wejścia-wyjścia można podzielić na cztery typy:

- **znakowe** - przetwarzanie danych odbywa się znak po znaku;
- **łańcuchowe** - przetwarzanie danych odbywa się wierszami;
- **sformatowane** - przy przetwarzaniu danych stosowane są specyfikatory formatu;
- **rekordowe (blokowe)** - dane przetwarzane są całymi blokami (rekordami).

Nazwy wybranych funkcji wykonujących poszczególne typy operacji przedstawiono na Rys. 1. Zastosowanie tych funkcji w programie wymaga dołączenia pliku nagłówkowego **stdio.h**.



Rys. 1. Typy operacji wejścia-wyjścia w języku C

W kolejnych rozdziałach przedstawiono opis wybranych funkcji wykonujących operacje znakowe, łańcuchowe i sformatowane.

2.3. Operacje znakowe

`getc()`	Nagłówek: `int getc (FILE *stream);`

- funkcja **getc()** pobiera jeden znak z aktualnej pozycji otwartego strumienia **stream** i uaktualnia pozycję;
- zmienna **stream** powinna wskazywać strukturę **FILE** reprezentującą strumień skojarzony z otwartym plikiem lub jeden ze standardowo otwartych strumieni (np. **stdin**);
- funkcja zwraca wartość całkowitą kodu wczytanego znaku lub wartość **EOF**, jeśli wystąpił błąd lub przeczytany został znacznik końca pliku;
- przykład odczytania jednego znaku ze standardowego wejścia (klawiatury) i jego wyświetlenia na ekranie:

```
int znak;
znak = getc(stdin);
printf("%c",znak);
```

`putc()`	Nagłówek: `int putc(int znak, FILE *stream);`

- funkcja **putc()** wpisuje znak do otwartego strumienia reprezentowanego przez argument **stream**;
- zmienna **stream** powinna wskazywać strukturę **FILE** reprezentującą strumień skojarzony z otwartym plikiem lub jeden ze standardowo otwartych strumieni (np. **stdout**);
- jeśli wykonanie zakończyło się poprawnie, to zwraca wypisany **znak**; jeśli wystąpił błąd, to zwraca wartość **EOF**;
- przykład wyświetlenia jednego znaku na standardowym wyjściu (ekranie monitora):

```
int znak = 'x';
putc(znak,stdout);
```

`getchar()`	Nagłówek: int **getchar**(void);

- funkcja **getchar()** pobiera znak ze strumienia **stdin** (klawiatura);
- jeśli wykonanie zakończyło się poprawnie, to zwraca przeczytany znak;
- jeśli wystąpił błąd albo został przeczytany znacznik końca pliku, to zwraca wartość **EOF**;
- funkcja **getchar()** jest równoważna wywołaniu funkcji **getc(stdin)**;
- przykład odczytania jednego znaku ze standardowego wejścia (klawiatury) i jego wyświetlenia na ekranie:

```
int znak;
znak = getchar();
printf("%c",znak);
```

`putchar()`	Nagłówek: int **putchar**(int **znak**);

- funkcja **putchar()** wpisuje **znak** do strumienia **stdout** (standardowo ekran);
- jeśli wykonanie zakończyło się poprawnie, to zwraca wypisany **znak**;
- jeśli wystąpił błąd, to zwraca wartość **EOF**;
- funkcja **putchar()** jest równoważna funkcji **putc()** wywołanej z drugim argumentem równym **stdout**;
- przykład wyświetlenia jednego znaku na standardowym wyjściu (ekranie monitora):

```
int znak = 'x';
putchar(znak);
```


`fgetc()`	Nagłówek: int **fgetc**(FILE *stream);

- funkcja **fgetc()** pobiera jeden znak ze strumienia wskazywanego przez **stream**;
- jeśli wykonanie zakończyło się poprawnie, to zwraca przeczytany znak po przekształceniu go na typ **int**;
- jeśli wystąpił błąd lub został przeczytany znacznik końca pliku, to zwraca wartość **EOF**.

`fputc()`	Nagłówek: int **fputc**(int **znak**, FILE *stream);

- funkcja **fputc()** wpisuje **znak** do otwartego strumienia reprezentowanego przez argument **stream**;
- jeśli wykonanie zakończyło się poprawnie, to zwraca wypisany **znak**;
- jeśli wystąpił błąd to zwraca wartość **EOF**.

`ungetc()`	Nagłówek: int **ungetc**(int **znak**, FILE *stream);

- funkcja **ungetc()** umieszcza **znak** z powrotem w strumieniu wejściowym.

2.4. Operacje łańcuchowe

`gets()`	Nagłówek: char* **gets**(char ***s**);

- funkcja **gets()** pobiera do bufora pamięci wskazywanego przez argument **s** linię znaków ze strumienia **stdin** (standardowo klawiatura);
- wczytywanie jest kończone po napotkaniu znacznika nowej linii **'\n'**, który w buforze pamięci **s** zastępowany jest znakiem końca łańcucha **'\0'**;
- **gets()** umożliwia wczytanie łańcucha zawierającego spacje i tabulatory;

- jeśli wykonanie zakończyło się poprawnie, to zwraca wskazanie do łańcucha **s**; jeśli napotka znacznik końca pliku lub gdy wystąpił błąd, to zwraca **EOF**;
- przykład odczytania jednego wiersza tekstu z klawiatury:

```
char tablica[80];
gets(tablica);
```

puts()	Nagłówek: int **puts**(const char ***s**);

- funkcja **puts()** wpisuje łańcuch **s** do strumienia **stdout** (standardowo ekran), zastępując znak **'\0'** znakiem **'\n'** (co oznacza automatyczne przejście do nowego wiersza po wyświetleniu zawartości łańcucha **s**);
- jeśli wykonanie zakończyło się poprawnie, to funkcja **puts()** zwraca ostatni wypisany znak; jeśli wystąpił błąd, to zwraca wartość **EOF**;
- przykład wyświetlenia jednego wiersza tekstu:

```
char tablica[30] = "Programowanie nie jest trudne";
puts(tablica);
```

fgets()	Nagłówek: char* **fgets**(char ***buf**, int **max**, FILE ***stream**);

- funkcja **fgets()** pobiera znaki z otwartego strumienia reprezentowanego przez **stream** i zapisuje je do bufora pamięci wskazanego przez **buf**;
- pobieranie znaków jest przerywane po napotkaniu znacznika końca linii **'\n'** lub odczytaniu **max-1** znaków;
- po ostatnim przeczytanym znaku wstawia do bufora **buf** znak **'\0'**;
- jeśli wykonanie zakończyło się poprawnie, to zwraca wskazanie do łańcucha **buf**; jeśli napotka znacznik końca pliku albo gdy wystąpił błąd, to zwraca wartość **NULL**.

fputs()	Nagłówek: int **fputs**(const char ***buf**, FILE ***stream**);

- funkcja **fputs()** wpisuje łańcuch **buf** do strumienia **stream**, nie dołącza znaku końca wiersza '**\n**';
- jeśli wykonanie zakończyło się poprawnie, to zwraca ostatni wypisany znak; jeśli wystąpił błąd, to zwraca wartość **EOF**.

2.5. Operacje sformatowane

scanf()	Nagłówek: int **scanf**(const char ***format, ...**);

- funkcja **scanf()** wprowadza dane ze strumienia **stdin** zgodnie z podanymi specyfikatorami formatu.

fscanf()	Nagłówek: int **fscanf**(FILE ***stream**, const char ***format, ...**);

- funkcja **fscanf()** działa podobnie jak **scanf()**, ale czyta dane z otwartego strumienia **stream** do listy argumentów.

sscanf()	Nagłówek: int **sscanf**(const char ***buf**, const char ***format, ...**);

- funkcja **sscanf()** działa podobnie jak **scanf()**, ale czyta dane z bufora pamięci **buf** do listy argumentów.

printf()	Nagłówek: int **printf**(const char ***format, ...**);

- funkcja **printf()** wyprowadza dane do strumienia **stdout** (standardowo ekran monitora) zgodnie z podanymi specyfikatorami formatu.

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

- funkcja **fprintf()** działa podobnie jak **printf()**, ale wyprowadza dane do otwartego strumienia **stream**.

`sprintf()`	Nagłówek: int **sprintf(**char ***buf,** const char ***format, ...);**

- funkcja **sprintf()** działa podobnie jak **printf()**, ale wyprowadza dane do bufora pamięci wskazywanego przez **buf**.

2.6. Schemat przetwarzania pliku

Plik jest wydzielonym fragmentem pamięci posiadającym określoną nazwę. Najczęściej jest to pamięć dyskowa. Z punktu widzenia języka C plik jest ciągiem bajtów, z których każdy może zostać oddzielnie odczytany. Do obsługi plików w języku C stosowane są **strumienie**. Strumień wiąże się z plikiem za pomocą **otwarcia**, zaś połączenie to jest przerywane przez **zamknięcie** strumienia. Często zamiast mówić o otwarciu lub zamknięciu strumienia, mówi się po prostu o otwarciu i zamknięciu pliku.

Zazwyczaj wszystkie operacje związane z przetwarzaniem pliku składają się z trzech etapów:

1. Otwarcie pliku (strumienia) - funkcja **fopen()**.
2. Operacje na pliku (strumieniu), np. czytanie, pisanie - funkcje:
   - dla plików tekstowych: **fprintf()**, **fscanf()**, **getc()**, **putc()**, **fgetc()**, **fputc()**, **fgets()**, **fputs()**, … ;
   - dla plików binarnych: **fread()**, **fwrite()**, **fseek()**, **ftell()**, **rewind()**, … .
3. Zamknięcie pliku (strumienia) - funkcja **fclose()**.

`fopen()`	Nagłówek: **FILE* fopen(**const char ***fname,** const char ***mode);**

- funkcja **fopen()** otwiera plik o nazwie **fname**, nazwa może zawierać całą ścieżkę dostępu do pliku;
- **mode** określa tryb otwarcia pliku (rodzaj dostępu do pliku):
  - "**r**" - plik tekstowy do odczytu;
  - "**w**" - plik tekstowy do zapisu; jeśli pliku nie ma to zostanie utworzony, jeśli jest, to jego poprzednia zawartość zostanie usunięta;
  - "**a**" - plik tekstowy do zapisu (dopisywania); dopisuje dane na końcu istniejącego pliku, jeśli pliku nie ma to zostanie utworzony;
  - "**r+**" - plik tekstowy do uaktualnienia, czyli zarówno do odczytywania jak i zapisywania;
  - "**w+**" - plik tekstowy do uaktualnienia (odczytu i zapisu); jeśli pliku nie ma to zostanie utworzony, jeśli jest, to jego poprzednia zawartość zostanie usunięta;
  - "**a+**" - plik tekstowy do uaktualnienia (odczytu i zapisu); dopisuje dane na końcu istniejącego pliku, jeśli pliku nie ma to zostanie utworzony; odczyt może dotyczyć całego pliku, zaś zapis może polegać tylko na dodawaniu nowego tekstu;
- funkcja **fopen()** zwraca wskaźnik na strukturę **FILE** skojarzoną z otwartym plikiem;
- w przypadku, gdy otwarcie pliku nie powiodło się, funkcja zwraca wartość **NULL**;
- po otwarciu pliku odwołania do niego następują przez wskaźnik pliku;
- domyślnie plik otwierany jest w trybie tekstowym, dodanie litery "**b**" w trybie oznacza otwarcie pliku w trybie binarnym, np.

```
FILE *stream1, *stream2, *stream3;
stream1 = fopen("dane.txt","r");
stream2 = fopen("c:\\baza\\data.bin","wb");
stream3 = fopen("wynik.txt","wt");
```

Plik **dane.txt** otwierany jest w trybie tekstowym tylko do odczytu. Plik **data.bin**, znajdujący się na dysku **C** w folderze **baza**, otwierany jest w trybie binarnym tylko do zapisu. Przy podawaniu ścieżki dostępu do tego pliku, zamiast jednego znaku \ należy podać dwa znaki - \\. Plik **wynik.txt** otwierany jest w trybie tekstowym tylko do zapisu. Litera "**t**" w trybie otwarcia jawnie wskazuje na otwarcie w trybie tekstowym.

`fclose()`	Nagłówek: int **fclose(FILE *stream);**

- funkcja **fclose()** zamyka plik wskazywany przez **stream** zwracając zero jeśli zamknięcie pliku było pomyślne lub **EOF** w przypadku wystąpienia błędu;
- wszystkie bufory związane ze strumieniem są opróżniane;
- po zamknięciu pliku wskaźnik **stream** może być wykorzystany do otwarcia innego pliku;
- w programie może być otwartych jednocześnie wiele plików.

Program przedstawiający schemat przetwarzania pliku.

```
#include <stdio.h>

int main(void)
{
    FILE *stream;

    stream = fopen("dane.txt","w");
    if (stream == NULL)
    {
        printf("Blad otwarcia pliku dane.txt!\n");
        return -1;
    }

    /* przetwarzanie pliku */

    fclose(stream);

    return 0;
}
```


W powyższym programie przedstawiono praktyczne zastosowanie funkcji otwierającej i zamykającej plik. Plik **dane.txt** jest otwierany funkcją **fopen()** tylko do zapisu. Po otwarciu pliku sprawdzana jest poprawność otwarcia (**stream == NULL**). Jeśli wystąpił błąd, to na ekranie zostanie wyświetlony odpowiedni komunikat i działanie programu zakończy się. Przetwarzanie pliku oznacza wykonanie na nim operacji typu czytanie i pisanie. Na koniec plik jest zamykany funkcją **fclose()**.

2.7. Pliki tekstowe

Elementami pliku tekstowego są **wiersze**, które mogą mieć różną długość. W systemach DOS i Microsoft Windows każdy wiersz pliku tekstowego zakończony jest parą znaków:

- **CR**, ang. *carriage return* - powrót karetki, kod ASCII - $13_{(10)} = 0D_{(16)}$;
- **LF**, ang. *line feed* - przesunięcie o wiersz, kod ASCII - $10_{(10)} = 0A_{(16)}$.

Załóżmy, że plik tekstowy ma postać przedstawioną na Rys. 2.

```
Pierwszy wiersz pliku
Drugi wiersz pliku
Trzeci wiersz pliku
```

Rys. 2. Przykładowa zawartość pliku tekstowego

Rzeczywista zawartość pliku jest następująca:

```
00000000: 50 69 65 72 77 73 7A 79|20 77 69 65 72 73 7A 20 | Pierwszy wiersz
00000010: 70 6C 69 6B 75 0D 0A 44|72 75 67 69 20 77 69 65 | pliku■■Drugi wie
00000020: 72 73 7A 20 70 6C 69 6B|75 0D 0A 54 72 7A 65 63 | rsz pliku■■Trzec
00000030: 69 20 77 69 65 72 73 7A|20 70 6C 69 6B 75 0D 0A | i wiersz pliku■■
```

Rys. 3. Bajty znajdujące się w pliku tekstowym

Na wydruku znajdują się (od lewej): przesunięcie od początku pliku (szesnastkowo), wartości poszczególnych bajtów pliku (szesnastkowo), znaki odpowiadające bajtom pliku (traktując bajty jako kody ASCII). Na Rys. 3 oznaczono znaki **CR** (**0D**) i **LF** (**0A**) znajdujące się na końcu każdego wiersza. W czasie

wczytywania pliku tekstowego do pamięci komputera znaki te zastępowane są jednym znakiem - **LF**, który w języku C reprezentowany jest przez **'\n'**. Przy zapisywaniu łańcucha znaków do pliku tekstowego sytuacja jest odwrotna - znak **LF** zastępowany jest parą **CR** i **LF**. W systemach Unix i Linux znakiem końca wiersza jest tylko **LF**.

2.8. Operacje na plikach tekstowych

Podczas przetwarzania plików tekstowych można stosować funkcje znakowe (**getc()**, **putc()**, **fgetc()**, **fputc()**), łańcuchowe (**fgets()**, **fputs()**) i sformatowane (**fscanf()**, **fprintf()**), które zostały szczegółowo opisane we wcześniejszej części tej instrukcji.

Załóżmy, że mamy następujące deklaracje zmiennych:

```
char  imie[10] = "Jan";
char  nazw[10] = "Kowalski";
int   wiek = 21;
float wzrost = 1.78f;
```

Wyświetlenie wartości powyższych zmiennych na ekranie może mieć następującą postać:

```
printf("Imie:     %s\n",imie);
printf("Nazwisko: %s\n",nazw);
printf("Wiek:     %d [lat]\n",wiek);
printf("Wzrost:   %.2f [m]\n",wzrost);
```

Wydruk będzie wyglądał w następujący sposób:

```
Imie:     Jan
Nazwisko: Kowalski
Wiek:     21 [lat]
Wzrost:   1.78 [m]
```

Zapisanie wartości tych samych zmiennych do pliku tekstowego **dane.txt** w takiej samej postaci wygląda następująco:


```
FILE *stream;
stream = fopen("dane.txt","w");
fprintf(stream,"Imie:     %s\n",imie);
fprintf(stream,"Nazwisko: %s\n",nazw);
fprintf(stream,"Wiek:     %d [lat]\n",wiek);
fprintf(stream,"Wzrost:   %.2f [m]\n",wzrost);
fclose(stream);
```

Zmiana funkcji z **printf()** na **fprintf()** wymaga dodania tylko jednego argumentu - wskaźnika **stream** do otwartego pliku. Pozostałe argumenty mają identyczną postać w obu funkcjach. W powyższym przykładzie, w celu skrócenia zapisu, pominięto sprawdzenie poprawności otwarcia pliku.

W kolejnym przykładzie do pliku tekstowego **liczby.txt** zapisywanych jest 6 wygenerowanych pseudolosowo liczb całkowitych z zakresu ⟨**0, 99**⟩. Każda liczba zapisywana jest w oddzielnym wierszu pliku.

Zapisanie do pliku 6 wygenerowanych pseudolosowo liczb całkowitych.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    FILE *stream;

    stream = fopen("liczby.txt","w");
    if (stream == NULL)
    {
        printf("Blad otwarcia pliku liczby.txt!\n");
        return -1;
    }

    srand((unsigned int)time(NULL));

    for (int i=0; i<6; i++)
        fprintf(stream,"%d\n",rand()%100);
```

```
    fclose(stream);

    return 0;
}
```

Przykładowa zawartość pliku **liczby.txt**:

```
25
89
6
31
75
29
```

Odczytanie pliku i wyświetlenie zapisanych w nim liczb na ekranie może mieć poniższą postać.

Wyświetlenie liczb całkowitych znajdujących się w pliku tekstowym.

```c
#include <stdio.h>

int main(void)
{
    FILE *stream;
    int x;

    stream = fopen("liczby.txt","r");
    if (stream == NULL)
    {
        printf("Blad otwarcia pliku liczby.txt!\n");
        return -1;
    }

    for (int i=0; i<6; i++)
    {
        fscanf(stream,"%d",&x);
        printf("%d\n",x);
    }
    fclose(stream);

    return 0;
}
```


W powyższy sposób można odczytać liczby tylko wtedy, gdy znamy ich ilość. W przypadku nieznanej ilości liczb zapisanych w pliku, podstawowym problemem jest wykrycie jego końca. Do wykrycia końca pliku stosowana jest funkcja **feof()**.

feof()	Nagłówek: int **feof(FILE *stream);**

- funkcja **feof()** sprawdza, czy podczas ostatniej operacji wejścia dotyczącej strumienia **stream** został osiągnięty koniec pliku;
- **feof()** zwraca wartość różną od zera, jeśli podczas ostatniej operacji wejścia został wykryty koniec pliku, w przeciwnym razie zwraca wartość **0** (zero).

Poniżej zamieszczono kod programu, w którym zastosowano funkcję **feof()**.

Wyświetlenie liczb całkowitych znajdujących się w pliku tekstowym.

```c
#include <stdio.h>

int main(void)
{
    FILE *stream;
    int x;

    stream = fopen("liczby.txt","r");
    if (stream == NULL)
    {
        printf("Blad otwarcia pliku liczby.txt!\n");
        return -1;
    }

    fscanf(stream,"%d",&x);
    while(!feof(stream))
    {
        printf("%d\n",x);
        fscanf(stream,"%d",&x);
    }

    fclose(stream);

    return 0;
}
```

W powyższym programie odczytanie pierwszej liczby (wywołanie funkcji **fscanf()**) następuje przed pętlą **while**. Jeśli podczas tej operacji nie osiągnięto końca pliku, to funkcja **feof()** zwróci wartość równą zero i nastąpi wejście do pętli. Pierwsza instrukcja w pętli wyświetla liczbę na ekranie. Druga instrukcja odczytuje kolejną liczbę z pliku. Sposób zapisu liczb w pliku wejściowym nie ma znaczenia dla prawidłowości ich odczytu. Liczby powinny być oddzielone od siebie znakami spacji, tabulacji lub znakiem nowego wiersza.

Niektóre funkcje czytające pliki same rozpoznają jego koniec. Zwracają wtedy wartość **EOF**. Sytuacja taka występuje w poniższym programie.

Wyświetlenie zawartości pliku tekstowego znak po znaku.

```
#include <stdio.h>

int main(void)
{
    FILE  *stream;
    int   znak;

    stream = fopen("prog.cpp","r");
    if (stream == NULL)
    {
        printf("Blad otwarcia pliku prog.cpp!\n");
        return -1;
    }

    while ((znak=fgetc(stream))!=EOF)
        putchar(znak);

    fclose(stream);

    return 0;
}
```

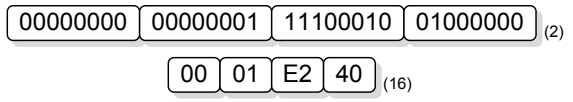
Program pobiera kolejne znaki z pliku **prog.cpp** wywołując funkcję **fgetc()**. Wartość zwracana przez funkcję jest podstawiana pod zmienną **znak**, która następnie jest porównywana ze stałą **EOF**. Jeśli **znak** nie jest równy **EOF**, to **putchar()** wyświetla go na ekranie. W przeciwnym przypadku pętla kończy się.

2.9. Pliki binarne

Plik binarny, w przeciwieństwie do pliku tekstowego, nie ma ściśle określonej struktury. Jeśli dane w pliku są przechowywane w sposób zgodny z ich reprezentacją w pamięci komputera, to mówimy, że są one zapisane w postaci binarnej. Zakładamy, że w programie została zadeklarowana i zainicjalizowana zmienna **x** typu **int**:

```
int x = 123456;
```

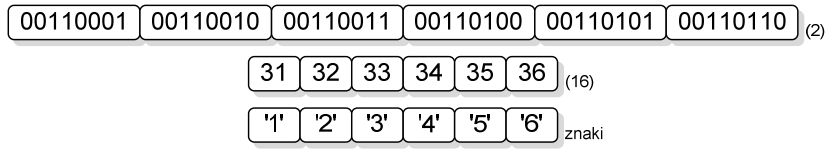
Liczba ta zajmuje w pamięci komputera 4 bajty (Rys. 4).



Rys. 4. Sposób przechowywania liczby całkowitej 123456 (typ int) w pamięci komputera (zapis w systemie dwójkowym i szesnastkowym)

Sposób przechowywania liczby **x** w pliku binarnym jest taki sam, jak w pamięci komputera (Rys. 4).

W przypadku zapisania tej samej liczby do pliku tekstowego, znajdzie się w nim 6 bajtów zawierających kody ASCII kolejnych cyfr liczby: **'1'**, **'2'**, **'3'**, **'4'**, **'5'**, **'6'** (Rys. 5).



Rys. 5. Sposób przechowywania liczby całkowitej 123456 (typ int) w pliku tekstowym (zapis w systemie dwójkowym i szesnastkowym)

2.10. Operacje na plikach binarnych

Na plikach binarnych wykonywane są dwie podstawowe operacje:

- zapis do pliku - funkcja **fwrite()**;
- odczyt z pliku - funkcja **fread()**.

`fwrite()`	Nagłówek: **size_t fwrite(const void *p, size_t s, size_t n, FILE *stream);**

- funkcja **fwrite()** zapisuje **n** elementów o rozmiarze **s** bajtów każdy, do pliku wskazywanego przez **stream**, biorąc dane z obszaru pamięci wskazywanego przez **p**;
- funkcja zwraca liczbę zapisanych elementów - jeśli jest ona różna od **n**, to wystąpił błąd zapisu (brak miejsca na dysku lub dysk zabezpieczony przed zapisem).

Poniższy przykład zawiera deklaracje i inicjalizacje zmiennych różnych typów oraz sposób ich zapisu do pliku binarnego, wskazywanego przez zmienną o nazwie **stream**.

```
int   x = 10;
int   tab[5] = {1,2,3,4,5};
float y = 1.2345;

fwrite(&x,sizeof(int),1,stream);
fwrite(tab,sizeof(int),5,stream);
fwrite(tab,sizeof(tab),1,stream);
fwrite(&y,sizeof(float),1,stream);
```

Pierwszym argumentem funkcji **fwrite()** jest adres w pamięci komputera, spod którego czytane są dane do zapisania w pliku. Z tego względu przed zmiennymi **x** i **y** pojawia się znak **&**. Nazwa tablicy **tab** jest adresem pierwszego jej elementu więc znak **&** nie jest potrzebny. Drugi argument funkcji **fwrite()** określa rozmiar jednego elementu zapisywanego do pliku. Do określenia rozmiaru we wszystkich przypadkach zastosowano operator **sizeof**. Trzecim argumentem jest liczba zapisywanych elementów. Tablica **tab** zapisywana jest dwukrotnie. Za pierwszym

razem zapisywanych jest 5 elementów tablicy, każdy o rozmiarze 4 bajtów. Za drugim razem zapisywana jest od razu cała tablica (drugi argument funkcji **fwrite()** jest równy rozmiarowi całej tablicy, zaś trzeci jest równy **1**). W obu przypadkach całkowita liczba zapisywanych bajtów jest taka sama (**20**). Ostatni argument funkcji **fwrite()** wskazuje plik, do którego mają być zapisane dane.

W kolejnym programie do pliku binarnego **liczby.dat** zapisywanych jest 10 wygenerowanych pseudolosowo liczb całkowitych (typ **int**). Dla uproszczenia zapisu pominięto sprawdzenie poprawności otwarcia pliku.

Zapisanie 10 liczb całkowitych do pliku binarnego.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    FILE  *stream;
    int   x;

    srand((unsigned int)time(NULL));

    stream = fopen("liczby.dat","wb");

    for (int i=0; i<10; i++)
    {
        x = rand() % 100;
        fwrite(&x,sizeof(int),1,stream);
    }

    fclose(stream);

    return 0;
}
```

`fread()`	Nagłówek: **size_t fread(void *p, size_t s, size_t n, FILE *stream);**

- funkcja **fread()** pobiera **n** elementów o rozmiarze **s** bajtów każdy, z pliku wskazywanego przez **stream** i umieszcza odczytane dane w obszarze pamięci wskazywanym przez **p**;
- funkcja zwraca liczbę odczytanych elementów - w przypadku gdy liczba ta jest różna od **n**, to wystąpił błąd końca strumienia (w pliku było mniej elementów niż podana wartość argumentu **n**).

Przy założeniu, że nie jest znana ilość liczb w pliku **liczby.dat**, odczytanie pliku i wyświetlenie liczb na ekranie będzie wyglądało następująco.

Odczytanie liczb całkowitych z pliku binarnego.

```
#include <stdio.h>

int main(void)
{
    FILE *stream;
    int x, ile = 0;

    stream = fopen("liczby.dat","rb");

    fread(&x,sizeof(int),1,stream);
    while (feof(stream) == 0)
    {
        printf("%d\n",x);
        ile++;
        fread(&x,sizeof(int),1,stream);
    }
    fclose(stream);

    printf("Odczytano: %d liczb\n",ile);

    return 0;
}
```


Odczytanie pierwszej liczby (wywołanie funkcji **fread()**) następuje przed pętlą **while**. Jeśli podczas tej operacji nie osiągnięto końca pliku, to funkcja **feof()** zwraca wartość równą zero i następuje wejście do pętli. W pętli wyświetlana jest na ekranie odczytana liczba **x** i zwiększana wartości zmiennej **ile** o jeden (**ile++**). Na koniec pętli odczytywana jest kolejna liczba z pliku. Warunek w pętli **while** można zapisać także w uproszczonej postaci:

```
while (!feof(stream))
```

W kolejnym przykładzie do pliku binarnego **dane.dat** zapisywana jest jedna struktura przechowująca dane osobowe oraz tablica przechowująca 5 liczb typu **float**.

Zapisanie struktury i tablicy do pliku binarnego.

```
#include <stdio.h>

struct osoba
{
    char imie[15];
    char nazw[20];
    int wiek;
};

int main(void)
{
    FILE *stream;
    struct osoba os = {"Jan","Nowak",23};
    float tab[5] = {3.5f, 2.1f, -3.7f, 0.0f, 8.2f};

    stream = fopen("dane.dat","wb");

    fwrite(&os,sizeof(struct osoba),1,stream);
    fwrite(tab,sizeof(float),5,stream);

    fclose(stream);

    return 0;
}
```

W przypadku sprawdzania rozmiaru struktury należy po operatorze **sizeof** zawsze podawać nazwę typu strukturalnego (**struct osoba**) lub nazwę zmiennej strukturalnej (**os**). Nie można natomiast określać rozmiaru struktury na podstawie sumy rozmiarów jej pól. Jest to spowodowane tym, że kompilatory mogą pomiędzy polami struktury lub na jej końcu wstawiać dodatkowe bajty (tzw. wypełniacze). Dzięki temu pola struktury będą rozpoczynały się od adresów podzielnych przez 4. W powyższym przykładzie suma rozmiarów pól struktury wynosi **39** (15 × 1 bajt + 20 × 1 bajt + 1 × 4 bajty = 39 bajtów), zaś operator **sizeof** zwraca dla całej struktury wartość **40**.

Oprócz wymienionych funkcji **fwrite()** i **fread()**, do operacji na plikach binarnych stosowane są również funkcje:

- void **rewind(FILE *stream);** - ustawia wskaźnik pozycji w pliku wskazywanym przez **stream** na początek pliku;
- int **fseek(FILE *stream,** long int **offset,** int **mode);** - pozwala przejść bezpośrednio do dowolnego bajtu w pliku wskazywanym przez **stream**; **offset** określa wielkość przejścia w bajtach, zaś **mode** - punkt początkowy, względem którego określane jest przejście (**SEEK_SET** - początek pliku, **SEEK_CUR** - bieżąca pozycja, **SEEK_END** - koniec pliku); gdy wywołanie jest poprawne, to funkcja zwraca wartość **0**; gdy wystąpił błąd (np. próba przekroczenia granic pliku), to funkcja zwraca wartość **-1**;
- long int **ftell(FILE *stream);** - zwraca bieżące położeniu w pliku wskazywanym przez **stream** (liczbę bajtów od początku pliku);
- int **fgetpos(FILE *stream, fpos_t *pos);** - zapamiętuję pod zminną **pos** bieżące położenie w pliku wskazywanym przez **stream**; zwraca **0**, gdy wywołania jest poprawne i wartość niezerową, gdy wystąpił błąd;
- int **fsetpos(FILE *stream,** const **fpos_t *pos);** - przechodzi do położenia **pos** w pliku wskazywanym przez **stream**; zwraca **0**, gdy wywołania jest poprawne i wartość niezerową, gdy wystąpił błąd.

3. Przebieg ćwiczenia

Na pracowni specjalistycznej należy wykonać zadania wskazane przez prowadzącego zajęcia. W różnych grupach mogą być wykonywane inne zadania.

1. Napisz program wyświetlający na ekranie wizytówkę o poniższej postaci. Wpisz w wizytówce swoje dane.

```
*******************************
*         Jan Kowalski        *
* e-mail: j.kowalski@gmail.com *
*       tel. 123-456-789      *
*******************************
```

   Zapisz wizytówkę w takiej samej postaci do pliku tekstowego **vcard.txt**.

2. Napisz program, który do pliku tekstowego **jeden.txt** zapisze macierz jednostkową o rozmiarze wprowadzonym z klawiatury.

   Przykładowe wywołanie programu:

```
Podaj rozmiar macierzy: 5
```

   Otrzymana zawartość pliku **jeden.txt**:

```
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

3. W pliku tekstowym **pomiar.txt** znajdują się dwie kolumny liczb rzeczywistych zawierające wyniki pomiarów wartości chwilowych napięcia i prądu. Napisz program, który odczyta zawartość pliku **pomiar.txt** i na jego podstawie utworzy plik **moc.txt** zawierający trzy kolumny liczb oznaczające: wartość chwilową napięcia, wartość chwilową prądu, wartość chwilową mocy. Plik **pomiar.txt** wskaże prowadzący zajęcia.

4. Napisz program, w którym linie tekstu wpisywane przez użytkownika z klawiatury są zapisywane do pliku **tekst.txt**. Zapisywanie kończymy, gdy użytkownik wprowadzi pusty łańcuch znaków (naciśnie klawisz ENTER).

5. W pliku **pesel.txt** zapisane są numery **PESEL** (każdy numer w oddzielnym wierszu). Napisz program, który odczyta zawartość tego pliku i sprawdzi:
   - czy dany numer **PESEL** jest prawidłowy?
   - czy dany numer **PESEL** należy do mężczyzny czy kobiety?

   Plik **pesel.txt** wskaże prowadzący zajęcia.

   Przykładowe wywołanie programu:

```
92040251610 – OK - M
92040251611 – BLAD
92040264401 – OK – K
```

6. Plik binarny **dane.dat** zawiera liczby całkowite typu **int**. Napisz program, który wyświetli na ekranie liczby odczytane z tego pliku oraz poda ich ilość, sumę i średnią arytmetyczną. Plik **dane.dat** wskaże prowadzący zajęcia.

   Przykładowe wywołanie programu:

```
10
15
30
----------------
Ilosc liczb: 3
Suma:        55
Srednia:     18.33333
```

7. W pliku binarnym **pomiar.dat** zapisane są naprzemiennie wyniki pomiaru wartości chwilowych napięcia **u(t)** i prądu **i(t)** (liczby zmiennoprzecinkowe pojedynczej precyzji). Napisz program, który odczyta zawartość pliku **pomiar.dat** i na jego podstawie utworzy plik tekstowy **moc.txt** zawierający trzy kolumny liczb oznaczające: wartość chwilową napięcia, wartość chwilową prądu, wartość chwilową mocy. Plik **pomiar.dat** wskaże prowadzący zajęcia.

u(t)	i(t)	u(t)	i(t)	u(t)	i(t)	…

8. Plik binarny **hdd.dat** zawiera struktury **dysk** opisujące dysk twardy. Kolejne pola struktury opisują: **producenta**, **model**, **pojemność** (w GB), **prędkość obrotową** (w obr/min), **cenę** (w PLN). Plik **hdd.dat** wskaże prowadzący zajęcia.


```
struct dysk
{
    char producent[20];
    char model[20];
    int pojemnosc;
    int predkosc;
    int cena;
};
```

Napisz program, który:

- odczyta zawartość pliku i wyświetli dane o dyskach na ekranie;
- zapisze odczytane dane do pliku tekstowego **hdd.txt** (jeden wiersz pliku powinien zawierać dane jednego dysku);
- obliczy i wyświetli średnią cenę dysków o pojemności większej lub równej 1000 GB i średnią cenę dysków o pojemności mniejszej od 1000 GB.

4. Literatura

[1] Prata S.: Język C. Szkoła programowania. Wydanie VI. Helion, Gliwice, 2016.

[2] Kernighan B.W., Ritchie D.M.: Język ANSI C. Programowanie. Wydanie II. Helion, Gliwice, 2010.

[3] King K.N.: Język C. Nowoczesne programowanie. Wydanie II. Helion, Gliwice, 2011.

[4] Kochan S.G.: Język C. Kompendium wiedzy. Wydanie IV. Helion, Gliwice, 2015.

[5] Wileczek R.: Microsoft Visual C++ 2008. Tworzenie aplikacji dla Windows. Helion, Gliwice, 2009.

[6] http://www.cplusplus.com/reference/clibrary - C library - C++ Reference

5. Zagadnienia na zaliczenie

1. Wyjaśnij pojęcie strumienia.
2. Omów standardowe strumienie wejścia-wyjścia.

3. Scharakteryzuj typy operacji wejścia-wyjścia w języku C.
4. Opisz schemat przetwarzania pliku w języku C.
5. Scharakteryzuj tryby otwarcia pliku stosowane w funkcji **fopen()**.
6. Wyjaśnij budowę pliku tekstowego.
7. Opisz sposoby wykrywania końca pliku tekstowego i binarnego.
8. Podaj różnice pomiędzy plikami tekstowymi i binarnymi.
9. Scharakteryzuj argumenty funkcji **fwrite()** i **fread()**.

6. Wymagania BHP

Warunkiem przystąpienia do praktycznej realizacji ćwiczenia jest zapoznanie się z instrukcją BHP i instrukcją przeciw pożarową oraz przestrzeganie zasad w nich zawartych.

W trakcie zajęć laboratoryjnych należy przestrzegać następujących zasad.

- Sprawdzić, czy urządzenia dostępne na stanowisku laboratoryjnym są w stanie kompletnym, nie wskazującym na fizyczne uszkodzenie.
- Jeżeli istnieje taka możliwość, należy dostosować warunki stanowiska do własnych potrzeb, ze względu na ergonomię. Monitor komputera ustawić w sposób zapewniający stałą i wygodną obserwację dla wszystkich członków zespołu.
- Sprawdzić prawidłowość połączeń urządzeń.
- Załączenie komputera może nastąpić po wyrażeniu zgody przez prowadzącego.
- W trakcie pracy z komputerem zabronione jest spożywanie posiłków i picie napojów.
- W przypadku zakończenia pracy należy zakończyć sesję przez wydanie polecenia wylogowania. Zamknięcie systemu operacyjnego może się odbywać tylko na wyraźne polecenie prowadzącego.

- Zabronione jest dokonywanie jakichkolwiek przełączeń oraz wymiana elementów składowych stanowiska.
- Zabroniona jest zmiana konfiguracji komputera, w tym systemu operacyjnego i programów użytkowych, która nie wynika z programu zajęć i nie jest wykonywana w porozumieniu z prowadzącym zajęcia.
- W przypadku zaniku napięcia zasilającego należy niezwłocznie wyłączyć wszystkie urządzenia.
- Stwierdzone wszelkie braki w wyposażeniu stanowiska oraz nieprawidłowości w funkcjonowaniu sprzętu należy przekazywać prowadzącemu zajęcia.
- Zabrania się samodzielnego włączania, manipulowania i korzystania z urządzeń nie należących do danego ćwiczenia.
- W przypadku wystąpienia porażenia prądem elektrycznym należy niezwłocznie wyłączyć zasilanie stanowiska. Przed odłączeniem napięcia nie dotykać porażonego.