Fundusze Europejskie Wiedza Edukacja Rozwój

Rzeczpospolita Polska

Unia Europejska Europejski Fundusz Społeczny

Politechnika Białostocka

Wydział Elektryczny

Katedra Elektrotechniki Teoretycznej i Metrologii

Instrukcja do pracowni specjalistycznej

Temat ćwiczenia:

JĘZYK C - OPERACJE WEJŚCIA-WYJŚCIA, PLIKI TEKSTOWE

Ćwiczenie nr INF_D12

Pracownia specjalistyczna z przedmiotu:

Informatyka

Kod: **EDS1B 1007**

Opracował:

dr inż. Jarosław Forenc

Białystok 2018

Materiały zostały opracowane w ramach projektu „PB2020 – Zintegrowany Program Rozwoju Politechniki Białostockiej” realizowanego w ramach Działania 3.5 Programu Operacyjnego Wiedza, Edukacja, Rozwój 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Społecznego.

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 10.

1.2. Oprogramowanie

Na komputerach zainstalowane jest środowisko programistyczne Microsoft Visual Studio 2008 Standard Edition lub nowsze.

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 struktury **FILE**.

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])
```

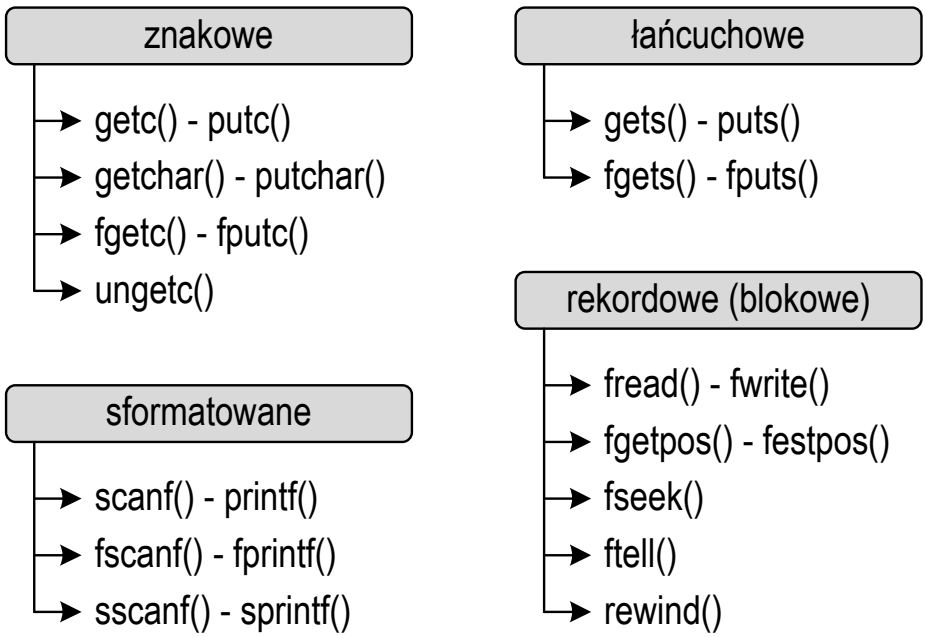
W dotychczas pisanych programach korzystano już z tych strumieni. Podczas każdego wywołania funkcji **printf()** niejawnie wykorzystywany jest strumień **stdout**, a podczas wywołania funkcji **scanf()** - strumień **stdin**.

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):

```
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(znak)** jest równoważna funkcji **putc(znak,stdout)**;
- przykład wyświetlenia kolejnych liter alfabetu na standardowym wyjściu (ekranie monitora):

```
int znak;
for (znak='a'; znak<='z'; znak++)
    putchar(znak);
```

Wynik wykonania kodu programu:

```
abcdefghijklmnopqrstuvwxyz
```

`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 ***str);**

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

- jeśli wykonanie zakończyło się poprawnie, to zwraca wskazanie do łańcucha **str**; 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 ***str);**

- funkcja **puts()** wpisuje łańcuch **str** do strumienia **stdout** (standardowo ekran), zastępując znak **'\0'** znakiem **'\n'** (co oznacza automatyczne przejście do nowego wiersza po wyświetleniu zawartości łańcucha **str**);
- 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 odczytaniu 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()**, … .
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 *fp1, *fp2, *fp3;
fp1 = fopen("dane.txt","r");
fp2 = fopen("c:\\baza\\data.bin","wb");
fp3 = 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 tym trybie.

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 *fp;

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

    /* przetwarzanie pliku */

    fclose(fp);

    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 (**fp == 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, łańcuchowe i sformatowane, które zostały szczegółowo opisane w Rozdziałach 2.3, 2.4 i 2.5.

W poniższym przykładzie pokazany jest zapis danych do pliku tekstowego **wynik.txt** przy wykorzystaniu funkcji znakowych, łańcuchowych i sformatowanych.

Zapis danych do pliku tekstowego przy wykorzystaniu różnych typów funkcji.

```
#include <stdio.h>

int main(void)
{
    FILE *fp;
    char txt[16] = "Tekst w tablicy", zn = 't';
    int x = 123;
    float y = 98.76543;

    fp = fopen("wynik.txt","w");

    // funkcje znakowe: putc(), fputc()
    putc('S',fp);   putc(zn,fp);   putc(97,fp);
    fputc('r',fp);  fputc(zn,fp);  fputc(10,fp);

    // funkcje łańcuchowe: fputs()
    fputs("Witaj swiecie!\n",fp);
    fputs(txt,fp);
    fputs("\n",fp);

    // funkcje sformatowane: fprintf()
    fprintf(fp,"txt = [%s]\n",txt);
    fprintf(fp,"x = [%d], y = [%.2f]\n",x,y);
    fprintf(fp,"Koniec\n");

    fclose(fp);

    return 0;
}
```

Zawartość pliku **wynik.txt** po wykonaniu programu:

```
Start
Witaj swiecie!
Tekst w tablicy
txt = [Tekst w tablicy]
x = [123], y = [98.77]
Koniec
```

Plik **wynik.txt** otwierany jest w trybie tekstowym do zapisu (**w**). W celu skrócenia kodu programu pominięto w nim sprawdzenie poprawności otwarcia pliku. Funkcje **putc()** i **fputc()** pozwalają zapisać do pliku tylko jeden znak, będący pierwszym ich argumentem. Znak może być podany w postaci stałej znakowej (**'S'**), zmiennej przechowującej znak (**zn**) lub liczby będącej kodem ASCII tego znaku (**97** - litera **'a'**). Funkcja **fputs()** umożliwia zapisanie do pliku całego łańcucha znaków. Łańcuch ten może być umieszczony bezpośrednio w wywołaniu funkcji (**"Witaj swiecie!\n"**) lub znajdować się w tablicy znaków (**txt**). Należy zwrócić uwagę na to, że funkcja **fputs()** nie dodaje na końcu znaku przejścia do nowego wiersza. Funkcja **fprintf()** działa analogicznie jak funkcja **printf()**. Dzięki specyfikatorom formatu (**%s**, **%d**, **%.2f**) można w dowolny sposób formatować dane zapisywane do pliku tekstowego.

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 ma 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 powyższych 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 do otwartego pliku. Pozostałe argumenty mają identyczną postać w obu funkcjach.

Do odczytania danych z pliku tekstowego także można wykorzystać funkcje znakowe, łańcuchowe i sformatowane. Załóżmy, że mamy plik tekstowy **liczby.txt** zawierający 5 liczb:

```
12.5  -3.33
4     6.25
-10.5
```

Z punktu widzenia języka C w pliku zapisane są bajty będące kodami ASCII poszczególnych znaków (Rys. 4).

```
00000000: 31 32 2E 35 20 20 2D 33|2E 33 33 0D 0A 34 20 20  |12.5  -3.33..4
00000010: 20 20 20 36 2E 32 35 0D|0A 2D 31 30 2E 35 0D 0A  |   6.25..-10.5..
```

Rys. 4. Zawartość kolejnych bajtów pliku **liczby.txt**

Wybór typu funkcji (znakowa, łańcuchowa, sformatowana) stosowanej do odczytania zawartości pliku zależny jest od operacji, którą chcemy wykonać na odczytanych danych. Jeśli zostaną użyte funkcje znakowe (**fgetc()**, **getc()**) to będzie możliwy dostęp tylko do pojedynczych znaków. Dzięki zastosowaniu funkcji łańcuchowych (**fgets()**) będziemy mieli dostęp do całych, kolejnych, wierszy pliku. Funkcje sformatowane (**fscanf()**) umożliwią m.in. odczytanie poszczególnych liczb i wykonanie na nich operacji arytmetycznych.

W większości przypadków nie wiemy ile dokładnie danych zapisanych jest w pliku. W związku z tym przy odczytywaniu jego zawartości stosowana jest pętla **while** oraz konieczne jest wykrycie końca pliku. Funkcje **fgetc()** i **getc()** po przeczytaniu znacznika końca pliku zwracają wartość **EOF**. Funkcja **fgets()** po osiągnięciu końca pliku zwraca wartość **NULL**. W przypadku odczytywania danych funkcją **fscanf()** należy zastosować dodatkową funkcję **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).

W poniższym programie zawartość pliku **liczby.txt** odczytywana jest trzykrotnie. Za pierwszym razem odczyt odbywa się znak po znaku przy zastosowaniu funkcji **fgetc()**. Odczytane znaki wyświetlane są na ekranie w dodatkowych nawiasach kwadratowych. W drugim przypadku zastosowana została funkcja **fgets()**. Odczytywane wiersze zapamiętywane są w tablicy **line**, a następnie wyświetlane na ekranie (także w dodatkowych nawiasach kwadratowych). W trzecim przypadku zastosowana została funkcja **fscanf()**. Dzięki jej użyciu odczytane dane są automatycznie zamieniane na liczby typu **float** (specyfikator formatu **%f**) i zapamiętywane w zmiennej **x**.

Odczytanie zawartości pliku tekstowego przy wykorzystaniu różnych typów funkcji.

```c
#include <stdio.h>

int main(void)
{
    FILE *fp;

    fp = fopen("liczby.txt","r");

    // funkcje znakowe
    int zn;

    zn = fgetc(fp);
    while(zn!=EOF)
    {
        printf("[%c]",zn);
        zn = fgetc(fp);
    }
    printf("\n--------------------------------\n");
    rewind(fp);

     // funkcje łańcuchowe
    char line[20];

    while(fgets(line,20,fp)!=NULL)
        printf("[%s]",line);
    printf("\n--------------------------------\n");
    rewind(fp);

    // funkcje sformatowane
    float x, suma = 0;

    fscanf(fp,"%f",&x);
    while(!feof(fp))
    {
        printf("%f\n",x);
        suma = suma + x;
        fscanf(fp,"%f",&x);
    }
    printf("--------------------------------\n");
    printf("Suma = %f\n",suma);

    fclose(fp);

    return 0;
}
```

Wynik uruchomienia programu:

```
[1][2][.][5][ ][ ][−][3][.][3][3][
][4][ ][ ][ ][ ][ ][6][.][2][5][
][−][1][0][.][5][
]
--------------------------------
[12.5  −3.33
][4     6.25
][−10.5
]
--------------------------------
12.500000
−3.330000
4.000000
6.250000
−10.500000
--------------------------------
Suma = 8.920000
```

Odczytane liczby rzeczywiste wyświetlane są na ekranie. Dodatkowo obliczana jest i wyświetlana ich suma. 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. Ostatnia 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.

Za każdym razem po dojściu do końca pliku należy wrócić na jego początek. Służy do tego funkcja **rewind()**.

rewind()	Nagłówek: void **rewind(FILE *stream);**

- ustawia wskaźnik pozycji w pliku wskazywanym przez **stream** na początek pliku.

3. Przebieg ćwiczenia

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

1. Wczytaj z klawiatury datę w formacie **dd-mm-rrrr**. Zadeklaruj trzy zmienne (**d**, **m**, **r**) i przypisz im dzień, miesiąc i rok zawarte w dacie. Wyświetl otrzymane wartości na ekranie.

   Przykładowe wywołanie programu:

   ```
   Podaj date: 21-09-2021
   ----------------------
   Dzien:    21
   Miesiac:  9
   Rok:      2021
   ```

2. Napisz program wyświetlający na ekranie wizytówkę o poniższej postaci.

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

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

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 zawierające: wartość chwilową napięcia, wartość chwilową prądu, wartość chwilową mocy. Plik **pomiar.txt** wskaże prowadzący zajęcia.

4. 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
```

5. 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).

6. W pliku **pesel.txt** zapisane są numery **PESEL** (każdy numer w oddzielnym wierszu). Napisz program, który odczyta zawartość tego pliku i sprawdzi:

   a) czy dany numer **PESEL** jest prawidłowy?

   b) czy dany numer **PESEL** należy do mężczyzny czy kobiety?

   Plik **pesel.txt** wskaże prowadzący zajęcia. Przykładowe wykonanie programu:

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

7. W pliku tekstowym zapisane są współczynniki wielomianu (w kolejności od stojącego przy najwyższej potędze). Napisz program, który obliczy wartość wielomianu dla argumentu wczytanego z klawiatury. Plik tekstowy wskaże prowadzący zajęcia.

8. Napisz program, który dla każdego wiersza pliku tekstowego obliczy i wyświetli liczbę wszystkich znaków, liczbę liter i liczbę cyfr. Nazwę pliku wczytaj z klawiatury.

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] Deitel P.J., Deitel H.: Język C. Solidna wiedza w praktyce. Wydanie VIII. Helion, Gliwice, 2020.

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

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

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

[7] https://cpp0x.pl/dokumentacja/standard-C/1 - Standard C

5. Pytania kontrolne

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.

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.