

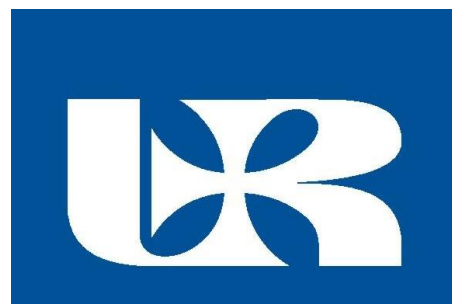
# Podstawy programowania w języku C

```
1  #include <math.h>
2  #include <stdio.h>
3  #include <conio.h>
4
5
6  double X(double t) {
7      return (1.0 + sin(2.0 * M_PI * 10000.0 * t) + cos(2.0 * M_PI * 10000.0 * t));
8  }
9
10 struct complexx {
11     double Re;
12     double Im;
13 };
14
15 struct complexx cmplx(double A, double B) {
16     struct complexx result;
17     result.Re = A;
18     result.Im = B;
19     return (result);
20 }
21
22 struct complexx cMult(struct complexx wA, struct complexx B) {
23
24     struct complexx result;
25
26     result.Re = A.Re * B.Im + A.Im * B.Re;
```

## Operacje na plikach

Do operacji na plikach służą funkcje z biblioteki stdio.h. Aby zacząć operować na pliku należy go otworzyć, a na koniec należy plik zamknąć. Otwierając plik należy określić w jakim celu plik jest otwierany (do czytania, do pisania etc.) oraz czy plik ma być otwarty w trybie binarnym (czyli jako ciąg bajtów) czy tekstowym.

## Lab7



Uniwersytet Rzeszowski  
ur.edu.pl

# Obsługa plików - zapis i odczyt danych

Najprościej mówiąc, plik to pewne dane zapisane na dysku. Każdy plik ma określoną nazwę. W programach do każdego pliku, który chcemy przeczytać lub w którym chcemy zapisać dane tworzymy identyfikator. Dzięki temu kod programu jest czytelniejszy i nie trzeba korzystać ciągle z pełnej nazwy pliku. Aby skojarzyć identyfikator z plikiem korzystamy z funkcji **open** lub **fopen**. Różnica wyjaśniona została poniżej.

## Podstawowa obsługa plików

Istnieją dwie metody obsługi plików: wysokopoziomowa i niskopoziomowa. Nazwy funkcji reprezentujących pierwszą metodę zaczynają się od litery "f" (np. **fopen()**, **fread()**, **fclose()**). Identyfikatorem pliku w tym przypadku jest wskaźnik na strukturę typu FILE. Owa struktura to pewna grupa zmiennych, która przechowuje dane o pliku. Identyfikator pliku stanowi jego "uchwyt". Funkcje niskopoziomowe to: **read()**, **open()**, **write()** i **close()**. Podstawowym identyfikatorem pliku jest tu liczba całkowita, która jednoznacznie identyfikuje dany plik w systemie operacyjnym. Liczba ta w systemach typu UNIX jest nazywana deskryptorem pliku. Należy pamiętać, że nie wolno nam używać funkcji z obu tych grup jednocześnie w stosunku do jednego, otwartego pliku, tzn. nie można najpierw otworzyć pliku za pomocą **fopen()**, a następnie odczytywać danych z tego samego pliku za pomocą **read()**. Czym różnią się oba podejścia do obsługi plików? Otóż metoda wysokopoziomowa ma swój własny bufor, w którym znajdują się dane po odczytaniu z dysku a przed wysłaniem ich do programu użytkownika. W przypadku funkcji niskopoziomowych dane kopiowane są bezpośrednio z pliku do pamięci programu. W praktyce używanie funkcji wysokopoziomowych jest prostsze, a przy czytaniu danych małymi porcjami również często szybsze i właśnie ten model zostanie tutaj zaprezentowany.

## Dane znakowe

Skupimy się teraz na najprostszym z możliwych zagadnień - zapisie i odczycie znaków oraz napisów (łańcuchów znaków). Napiszmy program, który stworzy plik "test.txt" i umieści w nim napis "Hello world":

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

int main() {
    FILE *fp; /* używamy metody wysokopoziomowej - musimy mieć zatem identyfikator pliku, uwaga na gwiazdkę! */
    char tekst[] = "Hello world";
    if ((fp = fopen("test.txt", "w")) == NULL) {
        printf("Nie mogę otworzyć pliku test.txt do zapisu!\n");
        exit(1);
    }
    fprintf(fp, "%s", tekst); /* zapisz nasz napis w pliku */
    fclose(fp); /* zamknij plik */
    return 0;
}
```

Jak już było wspomniane wyżej, do identyfikacji pliku używa się wskaźnika na strukturę **FILE** (czyli **FILE \***). Funkcja **fopen()** zwraca ów wskaźnik w przypadku poprawnego otwarcia pliku, bądź też **NULL**, gdy plik nie może zostać otwarty. Pierwszy argument funkcji to nazwa pliku, natomiast drugi to tryb dostępu, "w" oznacza "write" (zapis). Zwrócony "uchwyt" do pliku (**\*fp**) będzie mógł być wykorzystany jedynie w funkcjach zapisujących dane. Gdy otworzymy plik podając tryb "r" ("read", czytanie), będzie można z niego jedynie czytać dane. Funkcja **fopen()** została dokładniej opisana na stronie <http://pl.wikibooks.org/wiki/C/fopen>. Po zakończeniu korzystania z pliku należy plik zamknąć. Robi się to za pomocą funkcji **fclose()**.

---

*Jeśli zapomnimy o zamknięciu pliku, wszystkie dokonane w nim zmiany  
zostaną utracone!*

---

## Pliki a strumienie

Można zauważyć, że do zapisu do pliku używamy funkcji **fprintf()**, która wygląda bardzo podobnie do **printf()** (obie funkcje tak naprawdę robią tak samo). Jedyną różnicą jest to, że w **fprintf()** musimy jako pierwszy argument podać identyfikator pliku. Używana do wczytywania danych z klawiatury funkcja **scanf()** też ma swój odpowiednik wśród funkcji operujących na plikach - funkcję **fscanf()**.

W rzeczywistości język C traktuje tak samo klawiaturę i plik - są to źródła danych, podobnie jak ekran i plik, do których można dane kierować. Między klawiaturą i plikiem na dysku są podstawowe różnice i dostęp do nich odbywa się inaczej, jednak funkcje języka C pozwalają nam o tym zapomnieć i same zajmują się szczegółami technicznymi. Z punktu widzenia programisty, urządzenia te sprowadzają się do nadanego im w programie identyfikatora. Uogólnione pliki nazywa się w C strumieniami.

Każdy program w momencie uruchomienia "otrzymuje" od razu trzy otwarte strumienie:

- **stdin** (wejście)
- **stdout** (wyjście)
- **stderr** (wyjście błędów)

Aby z nich korzystać należy dołączyć plik nagłówkowy **stdio.h**.

Pierwszy z tych strumieni umożliwia odczytywanie danych wpisywanych przez użytkownika, natomiast pozostałe dwa służą do wyprowadzania informacji oraz powiadamiania o błędach.

Warto tutaj zauważyć, że konstrukcja:

```
fprintf (stdout, "Hej, ja działałem!");
```

jest równoważna konstrukcji:

```
printf ("Hej, ja działałem!");
```

Podobnie jest z funkcją `scanf()`.

```
fscanf (stdin, "%d", &zmienna);
```

działa tak samo jak:

```
scanf ("%d", &zmienna);
```

Program wykonujący operację na plikach powinien zachować schemat działania zapewniający poprawną pracę:

1. Otwarcie pliku w odpowiednim trybie (do zapisu, odczytu, zapisu i odczytu) - Operację otwarcia pliku wykonujemy za pomocą funkcji ***fopen***. Funkcja ta posiada dwa parametry: pierwszy - nazwę pliku (właściwie: ścieżkę do pliku w postaci napisu np. "plik.txt", "/etc/fstab", "C: \\CONFIG.SYS"), drugi - tryb otwarcia pliku (również napis). Jeżeli operacja otwarcia powiodła się funkcja zwraca uchwyt do pliku wykorzystywany później w operacjach zapisu, odczytu i zamknięcia pliku. Jeżeli wystąpił błąd otwarcia pliku funkcja zwraca wartość ***NULL***. Tryb otwarcia pliku podajemy w postaci łańcucha tekstowego składającego się z odpowiednich składowych. Łańcuchy tekstowe identyfikujące tryb otwarcia przedstawione zostały poniżej:

<b>r</b>	otwarcie pliku w trybie tekstowym do odczytu
<b>rb</b>	otwarcie pliku w trybie binarnym do odczytu
<b>w</b>	otwarcie pliku w trybie tekstowym do zapisu
<b>wb</b>	otwarcie pliku w trybie binarnym do zapisu
<b>a</b>	otwarcie pliku w trybie tekstowym, w celu dopisania na koniec pliku
<b>ab</b>	otwarcie pliku w trybie binarnym, w celu dopisania na koniec pliku
<b>r+</b>	otwarcie pliku w trybie tekstowym, zarówno do pisania i czytania
<b>rb+</b>	otwarcie pliku w trybie binarnym, zarówno do pisania i czytania
<b>w+</b>	otwarcie pliku w trybie tekstowym, zarówno do pisania i czytania
<b>wb+</b>	otwarcie pliku w trybie binarnym, zarówno do pisania i czytania
<b>a+</b>	otwarcie pliku w trybie tekstowym, zarówno do czytania i dopisywania na koniec pliku
<b>ab+</b>	otwarcie pliku w trybie binarnym, zarówno do czytania i dopisywania na koniec pliku

Poniższy fragment kodu otworzy w bieżącym katalogu plik tekstowy o nazwie raport.txt w trybie do zapisu:

```
FILE *out;  
out = fopen("raport.txt", "w");
```

Poniższy fragment kodu otworzy plik binarny (ciąg bajtów) o nazwie plik.bin w miejscu /home/student/Desktop/plik.bin do zapisu i odczytu istniejącego pliku - jeżeli plik nie istnieje funkcja zwróci ***NULL***:

```
FILE *out;  
out = fopen("/home/student/Desktop/plik.bin", "rb+");
```

Tryb binarny otwarcia pliku (do zapisu i/lub odczytu) oznacza, że po otworzeniu takiego pliku w edytorze tekstu możemy zobaczyć 'krzaki'. Ich zawartość to po prostu 'odbitka' surowych danych zapisanych w pamięci programu, który je utworzył, bez jakiegokolwiek przetwarzania na formę odczytywalną przez człowieka. W pliku binarnym poszczególne bajty mają dowolne wartości, niekoniecznie są interpretowalne jako znaki alfanumeryczne. Taki plik jest zazwyczaj nieczytelny dla człowieka po otwarciu w podstawowym edytorze tekstu. Struktura informacji w plikach binarnych jest ściśle określona przez oprogramowanie, które zapisuje tego typu pliki (np. określoną strukturę ma plik MS Word, MS Excel itp.). Programista chcący odczytać i właściwie zinterpretować w swoim programie dane z pliku binarnego, powinien znać jego strukturę. Tryb tekstowy otwarcia pliku (do zapisu i/lub odczytu) poszczególne bajty w pliku można zinterpretować jako dane alfanumeryczne (znaki), zapisane przy pomocy określonego kodowania (np. ASCII). Jak pamiętamy, funkcja ***fopen*** zwraca uchwyt do otwartego pliku, jeśli czynność się powiodła. W przeciwnym razie zwracana jest wartość ***NULL***. W celu zabezpieczenia programu przed niewłaściwym działaniem (odwołaniem do zerowego uchwytu) należy sprawdzić poprawność otwarcia pliku, np.:

```
FILE *f;                                     FILE *f;  
f = fopen("program.c", "r");                 f = fopen("program.c", "r");  
if (!f) {                                     if (f == NULL) {  
    perror("fopen");                          fprintf(stderr, "Niemoznaotworzycpliku.\n");  
    exit(1);                                  exit(1);  
}                                              }
```

2. Wykonanie operacji zapisu lub odczytu - W zależności od rodzaju danych (tekstowe, binarne) używamy różnych funkcji:

**Odczyt formatowany (plik tekstowy):**

```
fscanf(in, "%s", bufor);
```

gdzie:

- ***in*** - uchwyt do pliku otwartego do odczytu (plik, z którego chcemy wczytać dane)
- ***%s*** - typ danych, jakie chcemy wczytać z pliku - tu napis (specyfikatory typów są tu identyczne jak w funkcji ***scanf***)
- ***bufor*** - bufor (tablica) do której chcemy wpisać dane wczytane z pliku
- funkcja zwraca liczbę znaków, którą udało jej się przeczytać z pliku

Używając funkcji ***fscanf*** możemy wczytywać dane do tablicy, liczby całkowitej, zmiennoprzecinkowej, czy znaku.

#### Przykład:

Założmy, że nasz plik wygląda jak poniżej:

```
We are in 2022
```

Wczytajmy za pomocą funkcji **fgetc** 6 znaków z pliku do tablicy (zamiast tabicy, możemy np. użyć 6 zmiennych):

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

int main() {
    char bufor[6];
    FILE *fp;
    int i;

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

    bufor[0] = fgetc(fp);
    bufor[1] = fgetc(fp);
    bufor[2] = fgetc(fp);
    bufor[3] = fgetc(fp);
    bufor[4] = fgetc(fp);
    bufor[5] = fgetc(fp);

    printf("Odczytane znaki:|%s|\n", bufor);
    printf("Odczytane znaki:\n\n");

    for (i = 0; i < 6; i++)
        printf("%c\n", bufor[i]);

    fclose(fp);

    return 0;
}
```

Zapis formatowany danych do pliku (plik tekstowy):

```
fprintf(out, "%d", n);
fprintf(out, "%d", 1024);
```

gdzie:

- **out** - uchwyt do pliku otwartego do zapisu (plik, do którego chcemy wpisać dane)
- **%d** - typ danych, jakie chcemy zapisać do pliku - tu liczba całkowita (specyfikatory typów są tu identyczne jak w funkcji **printf**)
- **n** - liczba, którą chcemy wpisać do pliku

#### Przykład:

Założmy, że do pliku chcemy wpisać liczbę całkowitą, zmiennoprzecinkową i tekst (napis)

```
2022
3.140000
Programowanie w języku C.
```

Zapiszemy te dane do pliku za pomocą funkcji **fprintf**:

```
#include<stdio.h>

int main() {
    FILE *fp;

    int n = 2022;
    char text[] = "Programowanie w języku C.\n";
    double pi = 3.14;

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

    fprintf(fp, "%d\n", n);
    fprintf(fp, "%lf\n", pi);
    fprintf(fp, "%s\n", text);

    fclose(fp);

    return 0;
}
```

Zapis całej linii/łańcucha znakowego tekstu do pliku:

```
fputs("This is c programming.", out);
fputs(bufor, out);
```

gdzie:

- **out** - uchwyt do pliku otwartego do zapisu (plik, do którego chcemy wpisać dane)
- **bufor** - tablica znaków (napis), którą chcemy wpisać do pliku

### Przykład:

Założmy, że do pliku chcemy wpisać tekst (napis):

```
This is C programming.  
This is a system programming language.  
We are programming in C in 2022 year.  
Today is 20.12.2022
```

W celu zapisania całej linii do pliku, użyjemy funkcji **fputs**:

```
#include<stdio.h>  
  
int main() {  
  
    FILE *fp;  
    fp = fopen("file.txt", "w+");  
  
    fputs("This is C programming.\n", fp);  
    fputs("This is a system programming language.\n", fp);  
    fputs("We are programming in C in 2022 year.\n", fp);  
    fputs("Today is 20.12.2022", fp);  
    fclose(fp);  
  
    return 0;  
}
```

Zapis pojedynczego znaku tekstu do pliku (zapis pliku znakami):

```
fputc("ch", out);  
fputc("\n", out);  
fputc("a", out);
```

gdzie:

- **out** - uchwyt do pliku otwartego do zapisu (plik, do którego chcemy wpisać dane)
- **ch** - znak, który chcemy wpisać do pliku

### Przykład:

Założmy, że chcemy wpisać do pliku cały alfabet (wielkie i małe litery), wpisując je do pliku po jednym znaku (użyjemy funkcji **fputc**):

<pre>#include&lt;stdio.h&gt;  int main() {     FILE *fp;     int ch;      fp = fopen("alfabet.txt", "w+");      for (ch = 65; ch &lt;= 90; ch++)         fputc(ch, fp);      fputc('\n', fp);      for (ch = 97; ch &lt;= 122; ch++)         fputc(ch, fp);      fclose(fp);      return (0); }</pre>	<pre>#include&lt;stdio.h&gt;  int main() {     FILE *fp;     int ch;      fp = fopen("alfabet.txt", "w+");      for (ch = 'a'; ch &lt;= 'z'; ch++)         fputc(ch, fp);      fputc('\n', fp);      for (ch = 'A'; ch &lt;= 'Z'; ch++)         fputc(ch, fp);      fclose(fp);      return (0); }</pre>
---	--

### Odczyt binarny:

```
size_t fread(void*p, size_ts, size_tn, FILE*stream);  
  
fread(&d2, sizeof(float), 1, plik);  
fread(tab, sizeof(tab2), 1, plik);
```

gdzie:

- **&d1** - wskaźnik do zmiennej, którą chcemy odczytać z pliku
- **sizeof(float)** - rozmiar elementu do odczytu
- **1** - ilość elementów do odczytu
- **plik** - uchwyt do pliku otwartego do zapisu (plik, z którego chcemy odczytać dane binarne)

Poniższy program odczytuje w trybie binarnym z pliku pojedynczą zmienną typu **float** oraz całą tablicę liczb zmiennoprzecinkowych:

```
#include<stdio.h>  
  
int main() {
```

```

float d2, tab2[5];
int i;

FILE *plik2 = fopen("data.bin", "rb");

fread(&d2, sizeof(float), 1, plik2);
fread(tab2, sizeof(tab2), 1, plik2);
fclose(plik2);

printf("d2=%1.2f\n", d2);

for (i = 0; i < 5; i++)
    printf("%1.2f", tab2[i]);

return 0;
}

```

#### Zapis binarny :

```

size_t fwrite(constvoid*p, size_ts, size_tn, FILE*stream);

fwrite(&d1, sizeof(float), 1, plik);
fwrite(tab, sizeof(float), 5, plik)

```

gdzie:

- **&d1** - wskaźnik do zmiennej, którą chcemy zapisać do pliku
- **sizeof(float)** - rozmiar elementu do zapisu
- **1** - ilość elementów do zapisu
- **plik** - uchwyt do pliku otwartego do zapisu (plik, do którego chcemy wpisać dane binarne)

Poniższy program zapisuje w trybie binarnym do pliku pojedynczą zmienną typu **float** oraz całą tablicę liczb zmiennoprzecinkowych:

```

#include<stdio.h>

int main() {
    float d1 = 1.5;
    float tab1[5] = {1.0, 2.0, 3.0, 4.0, 5.0};

    FILE *plik1 = fopen("data.bin", "wb");

    fwrite(&d1, sizeof(float), 1, plik1);
    fwrite(tab1, sizeof(float), 5, plik1);

    fclose(plik1);

    return 0;
}

```

Zamknięcie otwartego pliku - Plik zamykany jest funkcją **fclose**. Funkcja **fclose** zwraca zero jeśli zamknięcie pliku było pomyślne lub **EOF** w przypadku wystąpienia błędu:

```
fclose(plik);
```

gdzie:

- **plik** - uchwyt do otwartego pliku, który chcemy zamknąć

```

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

int main() {
    FILE *plik;
    plik = fopen("plik.txt", "a+");

    if (plik == NULL) {
        printf("Bładotwarciapliku.\n");
        exit(-1);
    }

    /*przetwarzaniepliku*/

    fclose(plik);

    return 0;
}

```

### Sprawdzenie, czy plik się skończył

Funkcja **feof** testuje strumień ma ustawiony znacznik oznaczający koniec pliku, a nie czy nastąpił sam koniec pliku. Oznacza to że taki identyfikator jest ustawiany przez inną funkcję - funkcję która odczytuje dane. Można przyjąć, że ta funkcja czyta wszystkie dane, ale w momencie napotkania końca pliku ustawia znacznik **EOF** na strumieniu. Funkcja zwraca wartość niezerową jeżeli wcześniej napotkano koniec pliku tekstowego (tzn. jeżeli poprzedzająca operacja przeczytała znacznik końca pliku).

## Czytanie pliku liniami

```
#include<stdio.h>

int main() {
    FILE *fp;
    char buffer[255];

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

    while (fgets(buffer, 255, fp) != NULL) {
        printf("%s\n", buffer);
    }

    fclose(fp);

    return (0);
}
```

## Czytanie pliku znak po znaku

```
#include<stdio.h>

int main() {
    FILE *fp;
    char c;

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

    while ((c = fgetc(fp)) != EOF) {
        printf("%c", c);
    }

    fclose(fp);

    return (0);
}
```

## Czytanie pliku za pomocą fscanf

```
#include<stdio.h>

int main() {
    FILE *fp;
    char bufor[1024];
    fp = fopen("Untitled3.c", "r");

    while (fscanf(fp, "%s", bufor)) {
        printf("%s\n", bufor);
        if (feof(fp))
            break;
    }

    fclose(fp);
    return 0;
}
```

```
#include<stdio.h>

int main() {
    FILE *fp;
    char bufor[1024];
    fp = fopen("Untitled3.c", "r");

    while (!feof(fp)) {
        if (fscanf(fp, "%s", bufor) != 1)
            break;
        printf("%s\n", bufor);
    }

    fclose(fp);
    return 0;
}
```

## Parametry/argumenty uruchomienia programu(\*)

Do programu można przekazać dane nie tylko wczytując je z klawiatury, ale również za pomocą argumentów wywołania, podawanych podczas uruchomienia programu. Aby korzystanie z argumentów wywołania było możliwe, funkcja **main** programu powinna być zdefiniowana z nagłówkiem:

```
int main(int argc, char **argv) {
    /* */
    return 0;
}
```

```
int main(int argc, char *argv[]) {
    /* */
    return 0;
}
```

Pierwszy parametr, o zwyczajowej nazwie **argc**, jest liczbą argumentów podanych podczas wywołania programu. Pierwszym z tych argumentów, o numerze 0, jest zawsze nazwa wywoływanego programu. Tak więc **argc** jest zawsze co najmniej jeden. Zmienna **argv** wygląda trochę tajemniczo, ale tak naprawdę jest tablicą napisów zawierających kolejne argumenty wywołania: **argv[0]**, **argv[1]**, ..., **argv[argc - 1]**, indeksowanie rozpoczyna się od zera, dlatego ostatnią wartością indeksu jest **argc - 1** a nie **argc**.

Skompiluj poniższy program (**gcc plik.c -o plik**), i wywołaj go z różnymi argumentami (lub bez):

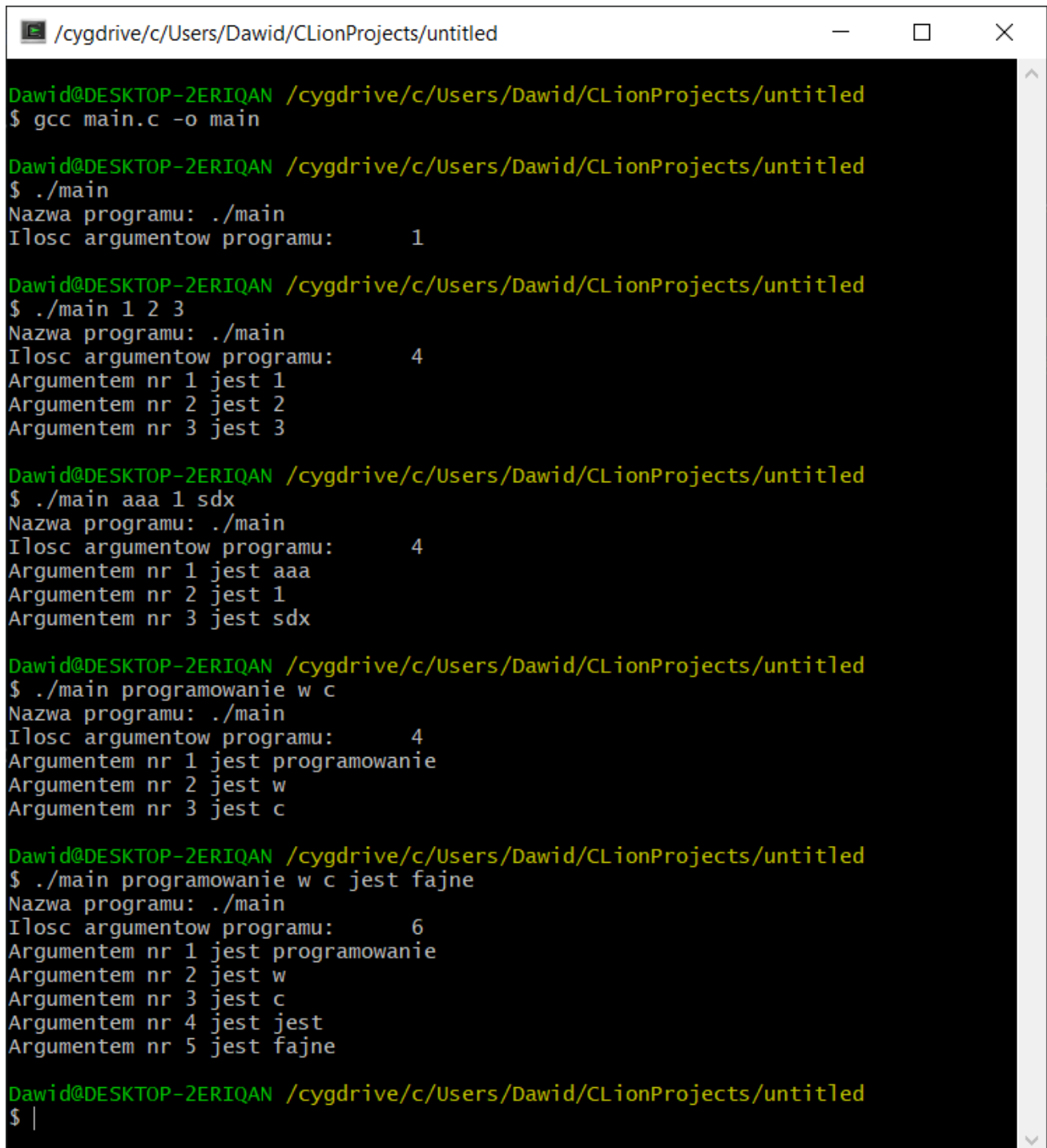
- ./plik
- ./plik 1 2 3
- ./plik aaa 1 sdx
- ./plik programowanie w c
- ./plik programowanie w c jest fajne

```
#include<stdio.h>

int main(int argc, char **argv) {
    int i;

    printf("Nazwa programu:\t%s\n", argv[0]);
    printf("Ilosc argumentow programu:\t%d\n", argc);

    for (i = 1; i < argc; i++)
        printf("Argumentem nr %d jest%s\n", i, argv[i]);
}
```



```
/cygdrive/c/Users/Dawid/CLionProjects/untitled

Dawid@DESKTOP-2ERIQAN /cygdrive/c/Users/Dawid/CLionProjects/untitled
$ gcc main.c -o main

Dawid@DESKTOP-2ERIQAN /cygdrive/c/Users/Dawid/CLionProjects/untitled
$ ./main
Nazwa programu: ./main
Ilosc argumentow programu:      1

Dawid@DESKTOP-2ERIQAN /cygdrive/c/Users/Dawid/CLionProjects/untitled
$ ./main 1 2 3
Nazwa programu: ./main
Ilosc argumentow programu:      4
Argumentem nr 1 jest 1
Argumentem nr 2 jest 2
Argumentem nr 3 jest 3

Dawid@DESKTOP-2ERIQAN /cygdrive/c/Users/Dawid/CLionProjects/untitled
$ ./main aaa 1 sdx
Nazwa programu: ./main
Ilosc argumentow programu:      4
Argumentem nr 1 jest aaa
Argumentem nr 2 jest 1
Argumentem nr 3 jest sdx

Dawid@DESKTOP-2ERIQAN /cygdrive/c/Users/Dawid/CLionProjects/untitled
$ ./main programowanie w c
Nazwa programu: ./main
Ilosc argumentow programu:      4
Argumentem nr 1 jest programowanie
Argumentem nr 2 jest w
Argumentem nr 3 jest c

Dawid@DESKTOP-2ERIQAN /cygdrive/c/Users/Dawid/CLionProjects/untitled
$ ./main programowanie w c jest fajne
Nazwa programu: ./main
Ilosc argumentow programu:      6
Argumentem nr 1 jest programowanie
Argumentem nr 2 jest w
Argumentem nr 3 jest c
Argumentem nr 4 jest jest
Argumentem nr 5 jest fajne

Dawid@DESKTOP-2ERIQAN /cygdrive/c/Users/Dawid/CLionProjects/untitled
$ |
```



Załóżmy, że chcemy do programu przekazać nazwę pliku (za pomocą argumentów wywołania programu). Następnie mamy plik o takiej nazwie otworzyć, przeczytać, i wypisać jego zawartość na ekranie:

```
#include<stdio.h>

int main(int argc, char **argv) {
    int i;

    printf("Nazwa programu:\t%s\n", argv[0]);
    printf("Ilość argumentów programu:\t%d\n", argc);

    for (i = 1; i < argc; i++)
        printf("Argumentem nr %d jest%s\n", i, argv[i]);
}
```

## Zadania do wykonania

1. Przetestuj zamieszczone wyżej fragmenty kodu i sprawdź ich działanie.
2. Napisz program, który zapyta użytkownika o nazwę pliku, a następnie otworzy ten plik do odczytu i wyświetli całą jego zawartość na ekranie.
3. Napisz program, który policzy znaki w pliku (parametr programu).
4. Napisz program, który policzy linie w pliku (parametr programu).
5. Napisz program, który policzy słowa w pliku (parametr programu).
6. (\*)Napisz program, który wpisuje do pliku (parametr programu) 40 znaków w wierszach 5-cio znakowych.
7. Napisz program, który wyświetla na ekranie liczbę wystąpień cyfr, małych i wielkich liter w podanym pliku (parametr programu).
8. (\*)Napisz program, który zlicza zapisane linijki w istniejącym pliku (parametr programu) i zapisuje ich liczbę do innego pliku (drugi parametr programu) (\*).
9. Napisz program, który kopiuje plik tekstowy (pierwszy parametr programu) do drugiego pliku (drugi parametr programu).
10. (\*)Napisz program, który kopiuje plik tekstowy (pierwszy parametr programu) do drugiego pliku (drugi parametr programu) w ten sposób, że każdy ciąg spacji redukuje do jednej i na końcu pliku podaje liczbę usuniętych spacji.
11. Napisz program, który wczytuje liczby z pliku i oblicza ich sumę oraz średnią.
12. (\*)Napisz program, który kopiując podany plik (parametr programu) do innego pliku (drugi parametr programu), zamienia małą literę na wielką, a wielką na małą (inne znaki tj. cyfry, kropki itp. kopiowane są bez zmian).
13. (\*)Napisz program, który sprawdzi i wypisze stosowną informację, czy podany plik (parametr programu) jest taki sam, jak inny plik (drugi parametr programu).
14. Napisz funkcję, która dostaje jako argument ścieżkę dostępu do pliku tekstowego i wypisuje na standardowym wyjściu zawartość pliku z pominięciem białych znaków.
15. Napisz funkcję, która dostaje jako argumenty ścieżkę dostępu do pliku tekstowego oraz znak c i zwraca jako wartość liczbę wystąpień znaku c w podanym w argumencie pliku.
16. Napisz funkcję, która dostaje jako argumenty ścieżki dostępu do dwóch plików i dopisuje zawartość pierwszego pliku na koniec drugiego pliku.
17. Napisz program, który zapisze do pliku (parametr programu) *n* linii wczytanych od użytkownika.
18. (\*)Napisz program, który kopiując podany plik (parametr programu) do innego pliku (drugi parametr programu), "łamie" wiersze, które mają więcej niż 50 znaków (łącznie ze spacjami). Znaki powyżej 50-tego przenoszone są do nowej linii (dodatkowy wiersz). Wiersze krótsze kopiowane są bez zmian.
19. (\*)Napisz program, który otwiera dwa pliki o nazwach podanych w wierszu poleceń. Jeśli argumentów nie podano, wówczas nazwy plików mają być pobrane od użytkownika. Program powinien wyświetlać wiersze z obu plików naprzemiennie, to znaczy: 1-szą linię z pierwszego pliku, 1-szą linię z drugiego pliku, 2-gą linię z pierwszego pliku, 2-gą linię z drugiego pliku, itd., aż do momentu, wyświetlenia ostatniego wiersza pliku zawierającego większą liczbę wierszy.
20. (\*)Napisz program, który przyjmuje 2 argumenty wiersza poleceń. Pierwszy z argumentów jest znakiem, drugi nazwą pliku. Program powinien wyświetlić na ekranie tylko te wiersze pliku wejściowego, które zawierają dany znak. Zakładamy, że każdy wiersz w pliku kończy się znakiem przejścia do nowej linii. Przyjmujemy, że żaden wiersz nie przekracza długości 256 znaków.
21. (\*)W pliku dane.txt znajdują się w kolejnych wierszach losowe liczby.
  - Do pliku *a.txt* wpisz ilość liczb parzystych znajdujących się w pliku *dane.txt* w następującej postaci: "Liczba parzystych jest [ilość liczb]".
  - Do pliku *b.txt* skopiuj wszystkie liczby z pliku *dane.txt*, w których cyfra dziesiątek jest równa 7 lub 0.
  - Do pliku *c.txt* skopiuj wszystkie liczby, które są kwadratami liczb całkowitych, np. taką liczbą jest liczba 225, ponieważ  $225 = 15^2$ .
22. Napisz funkcję, która jako parametry pobiera nazwę pliku do odczytu, nazwę pliku do zapisu oraz 2 napisy, *n1* oraz *n2* (tablice znaków). Zadaniem funkcji jest przepisanie pliku wejściowego do wyjściowego w taki sposób, że każde wystąpienie napisu *n1* w pliku wejściowym ma zostać zamienione na napis *n2* w pliku wyjściowym.

Przykład:

Wywołanie funkcji: `zadanie("we.txt", "wy.txt", "placki", "programowanie");` - chcemy zamienić każde wystąpienie słowa placki słowem programowanie;

Plik we.txt: lubie placki

Plik wy.txt: lubie programowanie

Zagadnienia i zadania oznaczone (\*) są dla chętnych.