Header stdio.h

Header	Opis zawartych funkcji	Podstawowe funkcje
stdio.h	Operacje wejścia i wyjścia	Podstawowe operacje na plikach: fopen(), fclose(), fread(), fwrite() Bezpośredni odczyt i zapis pojedynczego znaku i łańcucha znakowego: getc(), ~~getch(), getche(),~~ putc(), getchar(), putchar(), gets(), puts(), fgetc(), fputc(), fgets(), fputs(), ungetc() Sformatowane wejście/wyjście: scanf(), fscanf(), sscanf(), printf(), fprintf(), sprintf(), snprintf() Pozycjonowanie kursora odczytu z pliku i wpisu do pliku: ftell(), fseek() razem z: SEEK_SET, SEEK_CUR i SEEK_END, rewind(), fgetpos(), fsetpos() Obsługa błędów: clearerr(), feof(), ferror(), perror() Dalsze operacje na plikach: remove(), rename(), tmpfile()

#include<stdio.h>

Podstawowe operacje na plikach

- **fopen(**"ścieżka dostępu do pliku", "tryb"**)** - otwiera plik
- **fclose(**pointer do pliku**)** - zamyka plik
- **fread(**bufor, rozmiar bloku, liczba ilości 'rozmiar bloku', pointer do pliku**)** - odczytuje z pliku
- **fwrite(**tekst, rozmiar bloku tekstu, liczba ilości 'rozmiar bloku tekstu', pointer do pliku**)** - zapisuje do pliku

Tryby otwierania plików

tryb	Cel	Ginie stary tekst	Tworzy nieistniejący plik
r	Odczyt	Nie	Nie
w	Wpisywanie	Tak	Tak
a	Dopisywanie	Nie	Tak
r+	Odczyt i wpisywanie	Nie	Nie, ale próbuje go czytać
w+	Odczyt i wpisywanie	Tak	Tak
a+	Odczyt i dopisywanie	Nie	Tak

Utwórz na panelu (desktop) plik notatnika (nazwa **test**, rozszerzenie **.txt**) i wpisz tam poniższy tekst zapisując go z opcją **Kodowanie**: ANSI.

```
Litwo! Ojczyzno moja! ty jesteś jak zdrowie:
Ile cię trzeba cenić, ten tylko się dowie,
Kto cię stracił. Dziś piękność twą w całej ozdobie
Widzę i opisuję, bo tęsknię po tobie.
```

```
// Podstawowe operacje na plikach
```

```c
#include<stdio.h>                  // ten temat
#include<locale.h>                 // dla 'setlocale()'

int main(void)
{  setlocale(LC_CTYPE, "Polish");  // polskie znaki
   FILE *fp;                       // fp sugeruje nazwę 'file pointer'
   char bufor[200];
   char tekst[] = "\n          Adam Mickiewicz - Pan Tadeusz";

   fp = fopen("C:/Users/Artur/Desktop/test.txt", "r+"); // zmień identyfikator na swój

   fread(bufor, 103, 1, fp);       // czyta maksymalnie 103 znaki
   printf("fread() test : \n%s\n", bufor );
   printf(tekst);                  // pokaż 'tekst' na ekranie monitora

   fseek(fp, 0, SEEK_END);         // ustawia kursor czytania/zapisu na koniec pliku
   fwrite(tekst, sizeof(tekst), 1, fp );

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

Wyniki:

W pliku test.txt

```
Litwo! Ojczyzno moja! ty jesteś jak zdrowie:
Ile cię trzeba cenić, ten tylko się dowie,
Kto cię stracił. Dziś piękność twą w całej ozdobie
Widzę i opisuję, bo tęsknię po tobie.
           Adam Mickiewicz - Pan Tadeusz
```

Na ekranie:

```
fread() test :
Litwo! Ojczyzno moja! ty jesteś jak zdrowie:
Ile cię trzeba cenić, ten tylko się dowie,
Kto cię stracił
         Adam Mickiewicz - Pan Tadeusz
--------------------------------
Process exited after 10.06 seconds with return value 0
Press any key to continue . . . _
```

Bezpośredni odczyt i zapis pojedynczego znaku i łańcucha znakowego

Gdy wymagany jest odczyt jakiegoś znaku lub łańcucha znakowego z klawiatury, program wstrzymuje swoje działanie czekając na reakcję użytkownika.

Gdy wymagany jest odczyt jakiegoś znaku lub łańcucha znakowego z pliku, program odczytuje znak/znaki od miejsca, gdzie znajduje się kursor czytania/zapisu.
Gdy wymagany jest zapis jakiegoś znaku lub łańcucha znakowego do pliku, program zapisuje znak/znaki do miejsca, gdzie znajduje się kursor czytania/zapisu.

Poniżej dołożyłem *getch()* i *getche()* z **conio.h** bo tematycznie te funkcje pasują do reszty z **stdio.h**.

- **getc(stdin**/pointer źródła**)** - odczytuje pierwszy znak wprowadzony z klawiatury po przyciśnięciu [Enter]
- **putc(**znak, **stdout**/pointer docelowy**)** - pokazuje na ekranie wprowadzony do programu znak
  stdin/**stdout** można zastąpić pointerami do innych źródeł, ale te są zdecydowanie najważniejsze
- Jest tylko w conio.h: **getch()** - odczytuje znak wprowadzony z klawiatury (bez [Enter]) ale nie pokazuje go na ekranie - wykorzystywane zawsze, gdy wstrzymuje się widok ekranu zanim na ekranie ma się pojawić coś innego.
- Jest tylko w conio.h: **getche()** - odczytuje znak wprowadzony z klawiatury (bez [Enter]) i pokazuje go na ekranie - wykorzystywane zawsze, gdy swoją odpowiedź chcemy zobaczyć, np.

```
Porzucić program (Tak/Nie) : T
```

- **getchar(**znak/znaki**)** - odczytuje pierwszy znak wprowadzony z klawiatury po przyciśnięciu [Enter]
- **putchar(**znak**)** - pokazuje na ekranie wprowadzony do programu znak
- **gets(**znaki**)** - odczytuje łańcuch znaków z klawiatury
- **puts(**znaki**)** - pokazuje na ekranie wprowadzony do programu znak łańcuch znaków

- **fgetc(**pointer źródła**)** - odczytuje pierwszy znak wprowadzony ze źródła danych (klawiatury, pliku, bufora...) po przyciśnięciu [Enter]
- **fputc(**znak, pointer docelowy**)** - wpisuje znak do miejsca docelowego (ekranu, pliku, bufora...)
- **fgets(**bufor, liczba bajtów, pointer źródła**)** - odczytuje łańcuch znaków w ilości 'liczba bajtów' ze źródła danych (np. pliku) i daje go do bufora skąd jest dostępny do dalszego procesu
- **fputs(**bufor, pointer docelowy**)** - wpisuje ciąg znaków do miejsca docelowego z miejsca składowania danych (bufora/łańcucha znaków przez jego nazwę - np. nazwa = "Janek" - lub bezpośrednio)

- **ungetc(**znak, pointer docelowy**)** - umieszcza znak z powrotem do 'pointer docelowy'

Dane są odczytywane od miejsca ustawienia kursora czytania.
Dane są zapisywane od miejsca ustawienia kursora zapisu i przykrywają tam istniejący tekst.

○ getc(stdin), putc(znak, stdout), znak=getch(), znak=getche(), getchar(znak/znaki), putchar(znak), gets(znaki), puts(znaki)

Test dla **getchar()** wykonałem osobno ponieważ nie może on być wykonany bezpośrednio po **getc()** - funkcje te działają podobnie.

```c
// Operacje wejścia/wyjścia
#include<stdio.h>                   // ten temat
#include<locale.h>                  // dla 'setlocale()'

char znak;

int main(void)
{  setlocale(LC_CTYPE, "Polish");  // polskie znaki

   printf("\n - Bezpośredni odczyt i zapis pojedynczego znaku i łańcucha znakowego - \
\n");

   printf("\n Test na getchar(): "); znak = getchar();
   // printf(" Przyjąłem znak %c", znak);
   printf(" Przyjąłem znak: ");
   putchar(znak);
   return 0;
```

```
}
```

Występujący w tym kodzie samotny \ (*backslash*) na końcu linii "formatu", tuż przed znakiem [Enter], oznacza kontynuację tej linii i używany jest aby złamać linię dla łatwiejszego odczytu kodu (aby w jednej linii kod nie był za długi). W tym wypadku zarówno znak \ jak i znak [Enter] są ignorowane przez kompilator:

```
            "Borland \
      International"
```

jest tym samym co:

```
      "Borland International"
```

Przykładowy wynik działania programu:

```
 - Bezpośredni odczyt i zapis pojedynczego znaku i łańcucha znakowego -

 Test na getchar(): Artur
 Przyjąłem znak A
--------------------------------
Process exited after 8.7 seconds with return value 0
Press any key to continue . . . _
```

Podobnie jest w przypadku **gets()** - wcześniejsza obecność **getc()** i **getchar()** przeszkadza w otrzymaniu poprawnego wyniku.

```
// Operacje wejścia/wyjścia
#include<stdio.h>                   // ten temat
#include<locale.h>                  // dla 'setlocale()'

char znaki[15];

int main(void)
{  setlocale(LC_CTYPE, "Polish");  // polskie znaki

   printf("\n - Bezpośredni odczyt i zapis pojedynczego znaku i łańcucha znakowego - \
\n");

   printf("\n Test na gets(): "); gets(znaki);
   printf(" Przyjąłem ciąg znaków ");
   puts(znaki);
   return 0;
}
```

Przykładowy wynik działania programu:

```
 - Bezpośredni odczyt i zapis pojedynczego znaku i łańcucha znakowego -

 Test na gets(): Artur Buczek
 Przyjąłem ciąg znaków Artur Buczek

--------------------------------
Process exited after 30.05 seconds with return value 0
Press any key to continue . . . _
```

```c
// Operacje wejścia/wyjścia
#include<stdio.h>                   // ten temat
#include<conio.h>                   // dla getch() i getche()
#include<locale.h>                  // dla 'setlocale()'

char znak;

int main(void)
{  setlocale(LC_CTYPE, "Polish");  // polskie znaki

   printf("\n - Bezpośredni odczyt i zapis pojedynczego znaku i łańcucha znakowego - \
\n");

   printf("\n Test na getc(): "); znak = getc(stdin);
   // printf(" Przyjąłem znak %c", znak);
   printf(" Przyjąłem znak %c");
   putc(znak, stdout);

   printf("\n\n Test na getch(): "); znak = getch();    // getch() jest w 'conio.h'
   // printf("\n Przyjąłem znak %c", znak);
   printf("\n Przyjąłem znak %c");
   putc(znak, stdout);

   printf("\n\n Test na getche(): "); znak = getche();  // getch() jest w 'conio.h'
   // printf("\n Przyjąłem znak %c", znak);
   printf("\n Przyjąłem znak %c");
   putc(znak, stdout);

   return 0;
}
```

Widać stąd wyraźnie, że:
- ***stdin*** czyli *standard input* to klawiatura
- ***stdout*** czyli *standard output* to ekran monitora

Przykładowe wyniki działania programu:

```
 - Bezpośredni odczyt i zapis pojedynczego znaku i łańcucha znakowego -

 Test na getc(): Ronaldo
 Przyjąłem znak R

 Test na getch():
 Przyjąłem znak p

 Test na getche(): H
 Przyjąłem znak H
--------------------------------
Process exited after 70.49 seconds with return value 0
Press any key to continue . . . _
```

- **fgetc(stdin), fputc(znak, stdout), fgets(znaki), fputs(znaki)**

Utwórz plik `test1.txt` na 'desktopie' o treści:

```
Litwo! Ojczyzno moja! ty jesteś jak zdrowie:
Ile cię trzeba cenić, ten tylko się dowie,
Kto cię stracił. Dziś piękność twą w całej ozdobie
Widzę i opisuję, bo tęsknię po tobie.
```

Plik `test2.txt` będzie utworzony automatycznie.

```c
// Operacje wejścia/wyjścia

#include<stdio.h>                  // ten temat
#include<locale.h>                 // dla 'setlocale()'

char znak;
char bufor[200];

int main(void)
{  setlocale(LC_CTYPE, "Polish");  // polskie znaki

FILE *fp1;
fp1 = fopen("C:/Users/Artur/Desktop/test1.txt", "r");  // zmień identyfikator na swój

FILE *fp2;
fp2 = fopen("C:/Users/Artur/Desktop/test2.txt", "a+"); // zmień identyfikator na swój

  printf("\n - Bezpośredni odczyt i zapis pojedynczego znaku i łańcucha znakowego -
\n");

/************************ fgetc() i fputc()************************/
  // Z klawiatury na ekran
  printf("\n Test na fgetc() czytane z klawiatury: "); znak = fgetc(stdin);
  printf(" Przyjąłem znak %c");
  fputc(znak, stdout);

  // Z pliku (pierwszy znak) do innego pliku
  fseek(fp1, 0, SEEK_SET);        // ustawia kursor czytania/zapisu na początek pliku
  printf("\n\n Test na fgetc(): Czytam pierwszy znak z pliku fp1"); znak = fgetc(fp1);
  printf("\n Przyjąłem znak %c i wpisuję go do pliku fp2 \n\n", znak);
  fputc(znak, fp2);

  fprintf(fp2, "\n\n");           // zrób jedną linię wolną w pliku fp2

/************************ fgets() i fputs()************************/
  fseek(fp1, 5, SEEK_CUR);      // ustaw kursor czytania na początek następnego słowa
  fgets(bufor, 38, fp1);
  printf("fgets() test - tekst w buforze: \n%s\n", bufor );

  fputs(bufor, fp2);
  printf("fputs() test - tekst został wpisany do pliku fp2\n");
  printf("\nfputs() test - ten sam tekst jest tu na ekranie: \n");
  fputs(bufor, stdout);
```

```
  return 0;
}
```

Przykładowe wyniki:

W pliku test2.txt

```
L

 Ojczyzno moja! ty jesteś jak zdrowie
```

Na ekranie:

```
 - Bezpośredni odczyt i zapis pojedynczego znaku i łańcucha znakowego -

 Test na fgetc(): Messi
 Przyjąłem znak M

 Test na fgetc(): Czytam pierwszy znak z pliku fp1
 Przyjąłem znak L i wpisuję go do pliku fp2

fgets() test - tekst w buforze:
 Ojczyzno moja! ty jesteś jak zdrowie
fputs() test - tekst został wpisany do pliku fp2

fputs() test - ten sam tekst jest tu na ekranie:
 Ojczyzno moja! ty jesteś jak zdrowie
--------------------------------
Process exited after 23.29 seconds with return value 0
Press any key to continue . . . _
```

- **ungetc()**

Funkcja **ungetc()** może przyjąć przeczytany wcześniej znak np. funkcją **getc()** i wstawić w miejsce przeczytanego znaku inny znak podyktowany logiką procesu. Poniższy przykład szyfruje dane poprzez wymianę znaków na następujące bezpośrednio po nich, np. "Artur" staje się "Bsuvs" w sekwencji ASCII. **ungetc()** zwraca znak do bufora (pliku lub klawiatury - będzie on przeczytany przez **getc()** jako pierwszy).

Zauważ, że jakkolwiek zmienne 'znak' i 'znak_1' się deklaruje - czy to 'int' czy 'char', można je wydrukować
- jako liczby: `printf("%d", znak);`
- jak i znaki: `printf("%c", znak);`
- czyli: `printf("Znak %c odpowiada liczbie ASCII o numerze %d", znak, znak);`

```
// Operacje wejścia/wyjścia
#include<stdio.h>                 // ten temat
#include<locale.h>                // dla 'setlocale()'

int znak, znak_1;

int main()
{ setlocale(LC_CTYPE, "Polish");  // polskie znaki

  printf("\n - Najbardziej proste szyfrowanie tekstu czyli enigma dla dzieci - \n\n");
```

```c
  FILE *fp;
  fp = fopen("C:/Users/Artur/Desktop/test.txt", "r");  // zmień identyfikator na swój
  if (fp == NULL)
  {  printf("Błąd otwarcia pliku - sprawdź ścieżkę dostępu do pliku");
     return (-1);
  }

  while ( !feof(fp) )
  { znak = getc(fp);   // czyta znak i ustawia kursor czytania na następnym znaku
    if( znak != EOF )
    {  if(znak > '\x20')    // zostaw znak spacji i znaki sterujące w tym [Enter]
          // ungetc() cofa kursor na przeczytany ostatnio znak
          znak_1 = ungetc(znak+1, fp);           // wymień na następny znak ASCII
       else                                 // ungetc(znak-1, fp); odszyfrowuje tekst
          znak_1 = ungetc(znak, fp);

       printf("%c",znak_1);
       // przesuń kursor czytania z pliku do przodu bo ungetc() go cofnął
       znak = getc(fp);
    }
  }
  fclose(fp);
  return 0;
}
```

Wejście (plik `test.txt` na desktopie):

```
Litwo! Ojczyzno moja! ty jesteś jak zdrowie:
Ile cię trzeba cenić, ten tylko się dowie,
Kto cię stracił. Dziś piękność twą w całej ozdobie
Widzę i opisuję, bo tęsknię po tobie.
```

Wynik (na ekranie; zapisem tym można przykryć plik `test.txt`):

```
 - Najbardziej proste szyfrowanie tekstu czyli enigma dla dzieci -

Mjuxp" Pkd{z{op npkb" uz kftufť kbl {espxjf;
Jmf djë us{fcb dfojç- ufo uzmlp tjë epxjf-
Lup djë tusbdj´/ E{jť qjëlopťç ux$ x db´fk p{epcjf
Xje{ë j pqjtvkë- cp uëtlojë qp upcjf/
--------------------------------
Process exited after 1.798 seconds with return value 0
Press any key to continue . . .
```

Sformatowane wejście/wyjście

Wiadomości ogólne

- **scanf()**, **fscanf()**, **sscanf()** - odczytuje sformatowane dane wejściowe ze standardowego wejścia, pliku lub bufora
- **printf()**, **fprintf()**, **sprintf()**, **snprintf()** - wypisuje sformatowane dane do, odpowiednio, standardowego wyjścia, pliku lub bufora (określonego nazwą łańcucha znakowego, poniżej w tej prezentacji `char bufor[] )`

Pierwsza linia to odczyt danych, druga to ich zapis.

scanf przyjmuje dane z klawiatury. Zapis **scanf(**"format", zmienne**)** jest równoważny **fscanf(*stdin*,** "format", zmienne**);**

printf wpisuje dane na ekran. Zapis **printf(**"format", zmienne**)** jest równoważny **fprintf(*stdout*,** "format", zmienne**);**

O ile **scanf** oczekuje danych tylko z klawiatyry a **printf** wypisuje dane tylko na ekran, **fscanf** musi mieć podane źródło danych (klawiatura, plik, bufor, ...) a **fprintf** docelowe ich umiejscowienie.
Standardem dla **fscanf** i **fprintf** jest jednak operowanie na plikach: Wzięcie danych z pliku (**fscanf**) i wpisanie do pliku (**fprintf**).

sscanf i **sprintf** są dedykowane dla odczytu z bufora pamięci, czyli jakiejś zmiennej, zwykle tekstowej (**sscanf**) i zapisu do niego (**sprintf**).

snprintf zapisuje do bufora określoną liczbę pierwszych bajtów łańcucha znaków.

Wiadomości szczegółowe

- **scanf(**"format", zmienne**)** - odczytuje dane wprowadzane z klawiatury (*stdin*, standard input)
- **fscanf(**pointer do pliku odczytu, "format", miejsce na wprowadzone dane**)** - odczytuje dane z pliku i wprowadza je zgodnie z "formatem" do 'miejsca na wprowadzone dane', poniżej: char bufor[];
- **sscanf(**łańcuch znakowy skąd dane będą odczytane - poniżej: char bufor[], "format ingenerujący w dane wejściowe", zmienne**)** - odczytuje dane z łańcucha znakowego, przerabia ten łańcuch zgodnie z "formatem" przyjmując 'zmienne' jako kolejne wyrazy czytanego łańcucha znakowego.
  Zdanie: "Mam 25 lat i 0.5 roku". Przykład formatu "%s %d %s %c %f %s" przyjmuje całe zdanie, bo %s to "Mam", %d to "25", %s to "lat, %c to "i", %f to "0.5", %s to "roku.".
  Format "%s %*d %*s %*c %f %s" przyjmuje "Mam 0.5 roku." bo * (gwiazdka) ignorune zapis pomiędzy spacjami.

- **printf(**"format", zmienne**)** - daje dane wyjściowe na ekran monitora (*stdout*, standard output)
- **fprintf(**pointer miejsca zapisu (zwykle pointer do pliku), "format" zapisu z danymi**)** - zapisuje dane zgodnie z "formatem" zapisu danych do miejsca podanego przez pointer. **fprintf(***stdout*, "format" zapisu z danymi**)** jest równoważny **printf(**"format" zapisu z danymi**)**
- **sprintf(**łańcuch znakowy gdzie dane będą zapisane - poniżej: char bufor[], "format" zapisu z wartościami zmiennych, zmienne**)** - wpisuje dane do łańcucha znakowego (naszego 'bufora') zgodnie z podanym "formatem"
- **snprintf(**łańcuch znakowy gdzie dane będą zapisane - poniżej: char bufor[], ilość bajtów do zapisu, format zapisu, łańcuch znakowy z którego wzięty będzie zapis**)** - wpisuje dane do łańcucha znakowego (naszego 'bufora') zgodnie z podanym "formatem" ale tylko ograniczoną ' ilość bajtów do zapisu '. Funkcja ta jest niedostępna w Turbo C++

Jak traktować zmienną do wydruku, w jakim formacie ją wydrukować? Patrz: "Tablice języka C" --> "Lista specyfikatorów formatu w C".
Ten format poprzedzony jest znakiem % (procentu). Oprócz opisu w tej tabeli, trzeba sobie zdawać sprawę z możliwości modyfikacji tego zapisu. Np. %f wypisuje liczbę zmiennoprzecinkową z sześcioma miejscami po kropce dziesiętnej. Pisząc %3.2f rezerwujemy trzy miejsca dla części całkowitych i dwa dla ułamkowych ale liczba 1000 i większe będą prawidłowo pokazane.

Jak w "formacie" wydruku przedstawić pewne, 'specjalne' znaki lub doprowadzić je do działania w funkcji **printf()** i jej podobnych, np. przejść do następnej linii, cofnąć kursor zapisu itd? Patrz "Tablice języka C" --> "Znaki specjalne (*escape characters*)".
Każdy z tych znaków poprzedzony jest znakiem **** czyli *backslash* mówiącym: Uwaga, następnym znakiem (tylko jednym) jest znak specjalny, nie należy go brać dosłownie, albo ma być tylko wydrukowanym i nie wywoływać nieoczekiwanego działania (np. \" - nie zamykaj łańcucha znaków lecz tylko wydrukuj jeden

znak ") albo, wręcz przeciwnie, jest dedykowany aby takie działanie wykonać (np. \n - symuluj działanie [Enter]).

Utwórz plik na desktopie `test.txt` o treści:
`Programowanie to modelowanie rzeczywistosci.`

```
// Sformatowane wejście/wyjście
#include<stdio.h>                   // ten temat
#include<locale.h>                  // dla 'setlocale()'

int main()
{  setlocale(LC_CTYPE, "Polish");  // polskie znaki
   char nazwa[20];
   float i, j, k;
   char tekst[] = "Lubię bazy danych, język C a szczególnie Python wersja 3.00";
   char bufor[70], bufor_1[70], bufor_2[70];
   char w1[10], w2[10], w3[10], w4[10];    // dla sscanf()
   float w5;                               // dla sscanf()

   FILE *fp;
   // Zmień identyfikator (tu: Artur) na swój
   fp = fopen("C:/Users/Artur/Desktop/test.txt", "r+");        // czytaj i dopisuj

printf("\n ---------------------- scanf() i printf() ----------------------\n");
   // Czytanie z klawiatury do zmiennch bufora
   // 'Wejście' czytane jest tylko do pierwszej spacji a spacja niszczy wejście
   // do drugiego scanf() bo powoduje próbę automatycznego czytania.
   printf("Wpisz wyraz nie przekraczający 20 znaków: ");
   scanf("%s", nazwa);

   printf("\nWpisz trzy liczby zmiennoprzecinkowe.\n");
   printf("Po każdej liczbie może być albo spacja albo [Enter] : ");
   scanf("%f %f %f", &i, &j, &k);

   // Wypisywanie danych na ekran
   printf("\nPrzyjąłem nazwę: %s", nazwa);
   printf("\nPrzyjąłem liczby: %5.3f, %.2f, %f", i, j, k);    // różny format liczb

printf("\n\n ---------------------- fscanf() i fprintf()----------------------\n");
   // Czytanie z pliku do bufora
   fscanf(fp, "%s", bufor);                    // pierwsza spacja zatrzymuje odczyt
   printf("fscanf() w buforze: %s", bufor );
   // Wpisywanie do pliku
   fseek(fp, 0, SEEK_END);                     // dopisz na koniec pliku
   fprintf(fp, tekst);                         // sprawdź wynik w pliku 'test.txt'
   printf("\nSprawdź jak wygląda teraz plik test.txt\n");

printf("\n ---------------------- sscanf() i sprintf()----------------------\n");
   // Czytanie z bufora (zmiennej znakowej)
   printf("Oryginalne zdanie: \n%s \n", tekst);
   // tekst --> Lubię bazy danych, język C a szczególnie Python wersja 3.00
```

```
   sscanf(tekst, "%s   %s    %s    %*s %*c %*c   %*s       %s     %*s   %f", w1, w2,
w3, w4, &w5);

   printf("\nZmodyfikowane zdanie: \n");
   printf("Mam %s %.2f. Nie %s %s %s...", w4, w5, w1, w2, w3);
   // Zdanie przekopiowane do innego bufora
   sprintf(bufor_1, "Co lubię: %s", tekst);
   printf("\n\nW buforze_1: \n%s \n", bufor_1 );  // dla sprawdzenia co jest w buforze

   // Wpisywanie do bufora części łańcucha znaków
   snprintf(bufor_2, 39, "Co lubię najbardziej: %s.", tekst);
   printf("\nW buforze_2: \n%s\n", bufor_2);      // dla sprawdzenia co jest w buforze

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

Przykładowe wyniki (plik `test.txt` teraz na desktopie):

`Programowanie to modelowanie rzeczywistosci.Lubię bazy danych, język C a szczególnie Python wersja 3.00`

Na ekranie:

```
---------------------- scanf() i printf() ----------------------
Wpisz wyraz nie przekraczający 20 znaków: Anastazja

Wpisz trzy liczby zmiennoprzecinkowe.
Po każdej liczbie może być albo spacja albo [Enter] : 123.456 -25.657 8

Przyjąłem nazwę: Anastazja
Przyjąłem liczby: 123.456, -25.66 8.000000

---------------------- fscanf() i fprintf() ----------------------
fscanf() w buforze: Programowanie
Sprawdź jak wygląda teraz plik test.txt

---------------------- sscanf() i sprintf() ----------------------
Oryginalne zdanie:
Lubię bazy danych, język C a szczególnie Python wersja 3.00

Zmodyfikowane zdanie:
Mam Python 3.00. Nie Lubię bazy danych,...

W buforze_1:
Co lubię: Lubię bazy danych, język C a szczególnie Python wersja 3.00

W buforze_2:
Co lubię najbardziej: Lubię bazy danych

--------------------------------
Process exited after 33.38 seconds with return value 0
Press any key to continue . . . _
```

Na koniec 'sformatowanego wejścia/wyjścia' przykład najpopularniejszych operacji: `scanf(), printf(),` 'znaki specjalne' (*escape characters*) takie jak `'\n', '\t', '\b',` formatowanie wydruku w tabeli liczb, omijanie biblioteki `<math.h>` w przypadku pierwiastka liczby...

```
// Sformatowane wejście/wyjście

#include<stdio.h>                   // ten temat
#include<locale.h>                  // dla 'setlocale()'

long int liczba;
long int dzielnik = 2;
int licznik = 0;

int main()
{  setlocale(LC_CTYPE, "Polish");  // polskie znaki, Turbo C++ to nie przeszkadza
   // clrscr();                    // tylko dla Turbo C++
   printf("\n\tNapisz liczbę, którą chcesz rozłożyc na czynniki pierwsze.");
   printf("\n\tLiczba nie może być większa niż 2147483647 (2 147 483 647).\n\n");

   printf("Liczba = ");
   scanf("%d", &liczba);
   printf("Wynik:   %d = ", liczba);
   while (liczba != 1)
   {   if (liczba % dzielnik != 0)
       {  dzielnik = dzielnik + 1;            // dzielnik++;
          // Poniższy zapis to jakby 'niejawny zapis pierwiastka kwadratowego', bo
          //   dzielnik > liczba / dzielnik   <==>   dzielnik > sqrt(liczba)
          if (dzielnik > liczba / dzielnik)   // ostatni dzielnik
          {  if (licznik == 0)
                printf("%ld\n\n\tLiczba %ld jest liczą pierwszą ", liczba, liczba);
             else
                printf(" %ld ", liczba);

             break;
          }
       }
       else
       {  liczba = liczba / dzielnik;
          if (licznik % 5 == 0 && licznik != 0)      // pokaż tylko 5 liczb w linii
             printf("\n\t\t     ");     // to tylko przykład zrobienia tabeli liczb
                                        // tu nie będzie ona idealna
          licznik++;
          printf(" %d *", dzielnik);
       }
   }

   printf("\b ");  // zmaż ostatni znak (* lub spację): cofnij kursor i drukuj spację
   // w ten sposób możesz wydrukować np. powód błędu i po getc() zmazać to zdanie

// printf(\n\n"Naciśnij dowolny klawisz aby zakończyć działanie programu');
// getche();      // te dwie linie są tylko dla Turbo C++
}
```

Przykładowe wyniki:

```
       Napisz liczbę, którą chcesz rozłożyć na czynniki pierwsze.
       Liczba nie może być większa niż 2147483647 (2 147 483 647).

Liczba = 549909360
Wynik:   549909360 = 2 * 2 * 2 * 2 * 3 *
                     3 * 5 * 7 * 7 * 11 *
                     13 * 109
--------------------------------
Process exited after 12.38 seconds with return value 0
Press any key to continue . . . _
```

```
       Napisz liczbę, którą chcesz rozłożyć na czynniki pierwsze.
       Liczba nie może być większa niż 2147483647 (2 147 483 647).

Liczba = 2147483647
Wynik:   2147483647 = 2147483647

       Liczba 2147483647 jest liczbą pierwszą
--------------------------------
Process exited after 10.21 seconds with return value 0
Press any key to continue . . . _
```

Pozycjonowanie kursora odczytu z pliku i wpisu do pliku

Mam definicje tych funkcji w headerze **stdio.h** i nie zawaham się ich użyć w programie:

```
long  _Cdecl ftell(FILE *__stream);
int   _Cdecl fseek(FILE *__stream, long __offset, int __whence);
void  _Cdecl rewind(FILE *__stream);
int   _Cdecl fgetpos(FILE *__stream, fpos_t *__pos);
int   _Cdecl fsetpos(FILE *__stream, const fpos_t *__pos);
```

Tłumacząc to z komputerowego na nasze:

- **ftell(**pointer do pliku**)** - Daj aktualną pozycję kursora zaczynając liczyć od początku, który ma pozycję 0 (zero) w pliku **fp**

  ```
  ftell(fp);
  ```

- **fseek(**pointer do pliku, przesunięcie kursora w stosunku do aktualnej jego pozycji, jeden z trzech sposobów ustawienia kursora**)**, gdzie:
  'przesunięcie (*offset*) kursora w stosunku do aktualnej jego pozycji': 5 oznacza przesunięcie kursora do przodu o 5 pozycji/bajtów a -5 oznacza przesunięcie kursora do tyłu (cofnięcie go) o 5 pozycji/bajtów
  ' jeden z trzech sposobów ustawienia kursora' :
  SEEK_SET - ustaw kursor czytania/zapisu na początek pliku z możliwością przesunięcia go do przodu
  `fseek(fp, 0, SEEK_SET);` - ustaw kursor czytania/zapisu na początek pliku **fp**
  `fseek(fp, 5, SEEK_SET);` - ustaw kursor czytania/zapisu na początek pliku **fp** i pchnij go do przodu o 5 pozycji

SEEK_CUR - ustaw kursor czytania/zapisu z możliwością przesunięcia go do przodu lub do tyłu
`fseek(fp, 5, SEEK_CUR);` - przesuń kursor czytania/zapisu do przudu o 5 pozycji z jego pozycji aktualnej w pliku **fp**

SEEK_END - ustaw kursor czytania/zapisu na koniec pliku z możliwością przesunięcia go do tyłu
`fseek(fp, 0, SEEK_END);` - ustaw kursor czytania/zapisu na koniec pliku **fp**
`fseek(fp, -5, SEEK_END);` - ustaw kursor czytania/zapisu na koniec pliku **fp** i cofnij go o 5 pozycji

- **rewind(**pointer do pliku**)** - Nic mi nie dawaj tylko ustaw kursor czytania/zapisu na początek pliku
`rewind(fp);` to samo co: `fseek(fp, 0, SEEK_SET);`
- **fgetpos(**pointer do pliku, pozycja kursora**)** - Daj aktualną pozycję kursora zaczynając liczyć od początku, który ma pozycję 0 (zero) w pliku
`fgetpos(fp,pozycja_kursora);` - zmienna 'pozycja_kursora' trzyma pozycję kursora w pliku **fp**
- **fsetpos(**pointer do pliku, pozycja kursora**)** - Ustaw kursor w pliku na daną pozycję zaczynając liczyć od początku, który ma pozycję 0 (zero) w pliku
`fsetpos(fp,pozycja_kursora);` - zmienna 'pozycja_kursora' wskazuje miejsce usadowienia kursora w pliku **fp**

Plik na desktopie: `test.txt`
`Programowanie to modelowanie rzeczywistosci.`

```c
// Pozycjonowanie kursora odczytu z pliku i zapisu do pliku

#include<stdio.h>                   // ten temat
#include<string.h>                  // dla strlen()
#include<locale.h>                  // dla 'setlocale()'

int main()
{  setlocale(LC_CTYPE, "Polish");  // polskie znaki

   char bufor[50];

   FILE *fp;
   fpos_t pozycja_kursora;
   // Zmień identyfikator (tu: Artur) na swój
   fp = fopen("C:/Users/Artur/Desktop/test.txt", "r");    // tylko odczyt

printf("\n ------------------ ftell(), fseek() i rewind() ------------------\n");
   // Ustawiam kursor odczytu/zapisu gdzieś w środku tekstu - nieważne gdzie
   // i odczytuję jego pozycję
   fseek(fp, 17, SEEK_SET);
   printf("\nPozycja przesuniętego kursora: %d", ftell(fp) );

   // Ustawiam kursor na początek pliku, przesuwam go o 3 miejsca od aktualnej jego
   // pozycji i odczytuję plik do jego końca
   rewind(fp);
   fseek(fp, 8, SEEK_CUR);           // z początku pliku prrzesuń kursor o 8 znaków
   memset(bufor,0,strlen(bufor));    // wyczyść bufor
   fread(bufor, 50, 1, fp);          // albo   fgets(bufor, 50, fp);
   printf("\n\nW buforze po odczycie przesuniętego kursora, od początku pliku o 8
miejsc: \n%s\n", bufor );            // albo   fputs(bufor, stdout);
```

```
   // Co prawda, kursor już jest na końcu pliku ale dla pewności tam go ustawiam
   // i cofam o 19 pozycji
   fseek(fp, -19, SEEK_END);          // z końca pliku cofnij kursor o 19 znaków
   memset(bufor,0,strlen(bufor));     // wyczyść bufor
   fread(bufor, 50, 1, fp);           // albo   fgets(bufor, 50, fp);
   printf("\nW buforze po odczycie przesuniętego kursora, od końca pliku o 19 miejsc
do tyłu: \n%s\n", bufor );            // albo   fputs(bufor, stdout);

printf("\n -------------------- fgetpos() i fsetpos() --------------------\n");
   // Ustawiam kursor odczytu/zapisu gdzieś w środku tekstu - nieważne gdzie
   fseek(fp, 17, SEEK_SET);
   fgetpos(fp, &pozycja_kursora);
   printf("\nAktualna pozycja kursora: %d", pozycja_kursora);
   // Przeczytaj stąd do końca pliku (maksymalnie 50 znaków)
   memset(bufor,0,strlen(bufor));          // wyczyść bufor
   fread(bufor, 50, 1, fp);                // albo   fgets(bufor, 50, fp);
   printf("\nW buforze: \n%s\n", bufor );  // albo   fputs(bufor, stdout);

   memset(bufor,0,strlen(bufor));          // wyczyść bufor
   pozycja_kursora = 29;
   fsetpos(fp, &pozycja_kursora);
   printf("\nAktualna pozycja kursora: %d", pozycja_kursora);
   // Przeczytaj stąd do końca pliku (maksymalnie 50 znaków)
   fread(bufor, 50, 1, fp);                // albo   fgets(bufor, 50, fp);
   printf("\nW buforze: \n%s\n", bufor );  // albo   fputs(bufor, stdout);

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

Wynik działania programu:

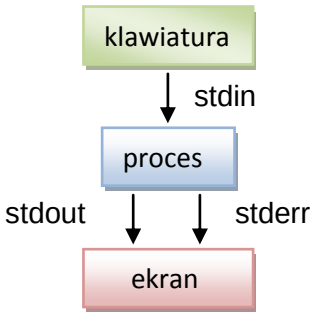
```
 ------------------ ftell(), fseek() i rewind() ------------------

Pozycja przesuniętego kursora: 17

W buforze po odczycie przesuniętego kursora, od początku pliku o 8 miejsc:
wanie to modelowanie rzeczywistości.

W buforze po odczycie przesuniętego kursora, od końca pliku o 19 miejsc do tyłu:
nie rzeczywistości.

-------------------- fgetpos() i fsetpos() --------------------

Aktualna pozycja kursora: 17
W buforze:
modelowanie rzeczywistości.

Aktualna pozycja kursora: 29
W buforze:
rzeczywistości.
```

```
--------------------------------
Process exited after 0.7149 seconds with return value 0
Press any key to continue . . . _
```

Obsługa błędów

Istnieją trzy podstawowe przepływy danych języka C, już zdefiniowane programowo:

- **stdin** - *standard input*, standardowe wejście danych - jeżeli nie określi się inaczej, np. plik, bufor/zmienna tekstowa, itd., to *stdin* oznacza klawiaturę
- **stdout** - *standard output*, standardowe wyjście danych - jeżeli nie określi się inaczej, np. plik, bufor/zmienna tekstowa, itd., to *stdout* oznacza ekran monitora
- **stderr** - *standard error*, standardowe wyjście obsługi błędów - jeżeli nie określi się inaczej, np. plik, bufor/zmienna tekstowa, itd., to *stderr* oznacza ekran monitora



Wychwytywaniem błędów składni języka C zajmuje się szczegółowo plik nagłówkowy `errno.h`.
Jednakże otwarcie pliku, jego modyfikacja i zamknięcie, obsługiwane przez `stdio.h`, w każdym typie programowania, czy to C, Python czy COBOL, musi być sprawdzone, czy zostało wykonane z sukcesem. Dlatego pewne funkcje obsługi błędów znajdują się już w `stdio.h`.

- **clearerr(**pointer do danych**)** - kasuje błędy i unieważnia End-Of-File (EOF, koniec pliku) zmieniając jego liczbę -1 na 0. Sama funkcja *clearerr()* nie podaje żadnej wielkości (*void*).
- Jeżeli wykonamy niedozwoloną instrukcję na źródle danych, np. pliku tylko do odczytu a chcemy wpisać do niego dane albo ten plik nie jest otwarty, itd, wtedy to 'źródło danych' ma przyklejoną etykietę 'błędnego źródła danych' i żadnych innych operacji nie da się na nim wykonać. Dopiero funkcja *clearerr()* zdziera tę etykietę, mówiąc 'to źródło nie jest powodem żadnego błędu' i w ten sposób pozwala na przeprowadzenie korekcyjnej na nim operacji.
- **feof(**pointer do danych**)** - sprawdza czy został osiągnięty koniec pliku
- **ferror(**pointer do danych**)** - sprawdza czy przypadkiem wystąpił błąd operacji na pliku
- **perror(**"*Nasz opis*"**)** - wyświetla zawartość *Nasz opis*, stawia dwukropek i podaje opis odpowiadający bieżącemu błędowi na standardowe wyjście błędów czyli ekran.
  Jeżeli nie chcemy nic mieć w *Nasz opis* i dwukropka, to wystarczy podać: **perror("")**;
  Zdanie opisu błędu, któru podaje *perror()* znajdziesz w: `"Pliki nagłówkowe" --> "errno.h"`

Plik na desktopie: `test.txt`
`Programowanie to modelowanie rzeczywistosci.`

```c
// Obsługa błędów
#include<stdio.h>                  // ten temat
#include<locale.h>                 // dla 'setlocale()'

int main()
```

```
{  setlocale(LC_CTYPE, "Polish");  // polskie znaki

   char znak;
   FILE *fp;
   fp = fopen("C:/Users/Artur/Desktop/Nie_ma_mnie.txt", "r" ); // tego pliku nie ma
   if( !fp )
   {  perror("Błąd otwarcia pliku \'Nie_ma_mnie.txt\'" );
      clearerr(fp);
      fp = fopen("C:/Users/Artur/Desktop/test.txt", "r" );     // tylko do odczytu
      if( !fp )
         perror("Błąd otwarcia pliku \'test.txt\'" );
      else
      {  printf("Plik \'test.txt\' jest otwarty. Można na nim pracować.");
         fwrite("TEKST", 5, 1, fp);  // chcemy do pliku wpisać słowo "TEKST"
         if( ferror(fp) )            // plik jest tylko do odczytu więc będzie błąd
         {  printf("\nBłąd zapisu do pliku \'test.txt\'");
            perror("\nBłąd drukowany przez 'perror' ");
            clearerr(fp);            // zdejmij z pliku 'klątwę' błędu
            znak = getc( fp );
            if( ferror(fp) )         // ciekawe, czy nadal jest błąd na pliku
               printf("Jak widzisz to zdanie to znaczy, że nadal jest błąd na pliku
\'test.txt\'");
            else
               printf("\nPrzeczytany znak: %c", znak);
         }
      }
   }
   else
      printf("Plik \'Nie_ma_mnie.txt\' jest otwarty. Można na nim pracować.");

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

W tym programie *pointer* **fp** 'pointuje' najpierw do pliku na desktopie o nazwie *Nie_ma_mnie.txt*. Skoro nic dobrego z tego nie wychodzi, bo plik nie istnieje, zabieram zabawki (czyli *pointer* **fp**) i idę z nim do innej piaskownicy (z pliku *Nie_ma_mnie.txt* do pliku *test.txt*). *Pointer* (wskaźnik) to zwykła zmienna, nie musi być trwale 'przyklejona' do jednej rzeczy.

Przykładowy wynik działania programu:

```
Błąd otwarcia pliku 'Nie_ma_mnie.txt': No such file or directory
Plik 'test.txt' jest otwarty. Można na nim pracować.
Błąd zapisu do pliku 'test.txt'
Błąd drukowany przez 'perror' : Bad file descriptor

Przeczytany znak: P
--------------------------------
Process exited after 13 seconds with return value 0
Press any key to continue . . . _
```

Dalsze operacje na plikach

- **remove(**ścieżka dostępu do pliku z jego nazwą**)** - usuwa plik
- **rename(**ścieżka dostępu do pliku z jego nazwą, nowa ścieżka dostępu do pliku z tą samą lub inną nazwą**)** - zmienia nazwę pliku i może służyć do przeniesienia go w inne miejsce pod tą samą lub inną nazwą
- **tmpfile()** - zwraca wskaźnik do pliku tymczasowego
  Tworzy plik tymczasowy, np. do przechowania posortowanych danych, z których program główny korzysta i ostatecznie przechowywanie tych danych jest zbyteczne.
  Wraz z końcem procesu pliku głównego, plik tymczasowy zostaje automatycznie usunięty ponieważ dane w nim zawarte są już niepotrzebne.
  Uwaga: Dev C++ może tu stwarzać problemy, ponieważ funkcja ta może starać się utworzyć plik tymczasowy w katalogu systemowym, co wymaga wcześniejszej deklaracji uprawnień administratora.

Wymagane pliki na desktopie (panelu):

test1.txt	- z dowolnym tekstem lub pusty	- będzie usunięty
A.txt	- z dowolnym tekstem lub pusty	- zmieni nazwę na A_nowy.txt
B.txt	- z dowolnym tekstem lub pusty	- będzie przesunięty do Nowy folder/B_nowy.txt
C.txt	- z programem języka C	- zmieni nazwę na C.cpp

Niech C.txt zawiera następujący kod:

```
#include<stdio.h>
int main()
{  printf("To ja, Artur");
   return 0;
}
```

i nowy, pusty folder o nazwie 'Nowy folder'

```
// Operacje na plikach

#include<stdio.h>                  // ten temat
#include<locale.h>                 // dla 'setlocale()'

int main()
{  setlocale(LC_CTYPE, "Polish");  // polskie znaki
   char bufor[] = "To jest test na istnienie pliku tymczasowego.";
   char bufor_1[50];
//    char tmp_bufor[L_tmpnam];

printf("\n ------------------- remove() - usuwanie pliku -------------------\n");
   remove("C:/Users/Artur/Desktop/test1.txt");              // zmień identyfikator na swój
   printf("Plik \'test1.txt\' nie powinien istnieć - popatrz na panel.\n");

printf("\n ---------- rename() - zmiana nazwy pliku i jego miejsca ---------\n");
   rename("C:/Users/Artur/Desktop/A.txt", "C:/Users/Artur/Desktop/A_nowy.txt");
   printf("Plik \'A.txt\' powinien zmienić nazwę na \'A_nowy.txt\' - popatrz na panel.");
   rename("C:/Users/Artur/Desktop/B.txt", "C:/Users/Artur/Desktop/Nowy folder/B_nowy.txt");
   printf("\nPlik \'B.txt\' powinien zostać przesunięty do folderu \'Nowy folder\' pod nazwą
\'B_nowy.txt\'.");
   rename("C:/Users/Artur/Desktop/C.txt", "C:/Users/Artur/Desktop/C.cpp");
   printf("\nPlik \'C.txt\' powinien zmienić nazwę na \'C.cpp\' - popatrz na panel.\n");

printf("\n ------------ tmpfile() - tworzenie pliku tymczasowego -----------\n");
   FILE *tymczasowy = tmpfile();              // utworzenie pliku tymczasowego
   if (tymczasowy == NULL)
   {  printf("Niemożliwe jest utworzenie pliku tymczasowego.\n");
```

```
        printf("Prawdopodobnie potrzebujesz uprawnień administratora.\n");
        return 0;
    }
    fputs(bufor, tymczasowy);
    printf("Tekst został wpisany do pliku \'tymczasowy\'\n");
    rewind(tymczasowy);
    fgets(bufor_1, sizeof bufor_1, tymczasowy);
    printf("Teraz tekst został przepisany z pliku \'tymczasowy\' do bufor_1\n");
    printf("Próba przeczytania tekstu z bufor_1: %s\n", bufor_1);

    return 0;
}
```

Wynik działania programu:

```
 ------------------- remove() - usuwanie pliku -------------------
Plik 'test1.txt' nie powinien istnieć - popatrz na panel.

 ---------- rename() - zmiana nazwy pliku i jego miejsca ---------
Plik 'A.txt' powinien zmienić nazwę na 'A_nowy.txt' - popatrz na panel.
Plik 'B.txt' powinien zostać przesunięty do folderu 'Nowy folder' pod nazwą 'B_nowy.txt'.
Plik 'C.txt' powinien zmienić nazwę na 'C.cpp' - popatrz na panel.

 ------------ tmpfile() - tworzenie pliku tymczasowego -----------
Niemożliwe jest utworzenie pliku tymczasowego.
Prawdopodobnie potrzebujesz uprawnień administratora.

--------------------------------
Process exited after 0.5675 seconds with return value 0
Press any key to continue . . . _
```