

Informatyka 2 (ES1E3017)

Politechnika Białostocka - Wydział Elektryczny
Elektrotechnika, semestr III, studia stacjonarne I stopnia
Rok akademicki 2022/2023

Wykład nr 4 (23.11.2022)

dr inż. Jarosław Forenc

Plan wykładu nr 4

- Pamięć a zmienne w programie
 - zmienne: automatyczne (auto), rejestrowe (register), zewnętrzne (extern), statyczne (static)
 - struktura procesu w pamięci komputera, ramka stosu
- Programy wielomodułowe
- Operacje wejścia-wyjścia w języku C
 - typy standardowych operacji wejścia wyjścia
 - strumienie, standardowe strumienie: stdin, stdout, stderr
- Operacje na plikach
 - otwarcie i zamknięcie pliku
 - operacje znakowe

Pamięć a zmienne w programie

- Ze względu na czas życia wyróżnia się w programie:
 - **obiekty statyczne** - istnieją od chwili rozpoczęcia działania programu aż do jego zakończenia
 - **obiekty dynamiczne** - tworzone i usuwane z pamięci w trakcie wykonania programu
 - automatycznie (bez udziału programisty)
 - kontrolowane przez programistę
- O typie obiektu (**statyczny** lub **dynamiczny**) decyduje klasa pamięci obiektu (ang. storage class)
 - **auto** - zmienne automatyczne
 - **register** - zmienne umieszczane w rejestrach procesora
 - **extern** - zmienne zewnętrzne
 - **static** - zmienne statyczne

Zmienne automatyczne - auto

- Miejsce deklaracji: najczęściej początek bloku funkcyjnego ograniczonego nawiasami klamrowymi { i }
- Pamięć przydzielana automatycznie przy wejściu do bloku i zwalniana po wyjściu z niego
- Zakres widzialności: ograniczony do bloku, w którym zmienne zostały zadeklarowane (**zmienne lokalne**)
- Dostęp do zmiennych z innych bloków możliwy przez wskaźnik
- Jeśli zmienne są inicjalizowane, to odbywa się ona przy każdym wejściu do bloku, w którym zostały zadeklarowane
- Nie ma potrzeby jawnego używania **auto**, gdyż domyślnie zmienne wewnątrz bloków funkcyjnych są lokalne

```
auto int x;
```

Zmienne rejestrowe - register

- Zazwyczaj o miejscu umieszczenia zmiennej automatycznej decyduje kompilator:
 - pamięć operacyjna - wolniejszy dostęp
 - rejestry procesora - szybszy dostęp
- Programista może zasugerować kompilatorowi umieszczenie określonej zmiennej automatycznej w rejestrach procesora
- Najczęściej dotyczy to zmiennych:
 - często używanych
 - takich, dla których czas dostępu jest bardzo ważny

```
register int x;
```

Zmienne zewnętrzne - extern

- Miejsce deklaracji: poza blokami funkcyjnymi, najczęściej na początku pliku z kodem źródłowym
- Pamięć na zmienne jest przydzielana, gdy program rozpoczyna pracę i zwalniana, gdy program kończy się
- Zakres widzialności: globalny - od miejsca deklaracji do końca pliku z kodem źródłowym (**zmienne globalne**)
- Jeśli inna zmienna lokalna, ma taką samą nazwę jak globalna, to lokalna przesłania widoczność zmiennej globalnej
- W większości implementacji języka C zmienne **extern** są automatycznie inicjalizowane zerem
- Etykieta **extern** może być pominięta (chyba, że program składa się z kilku plików z kodem źródłowym)
- Zalecane jest ograniczenie stosowania zmiennych globalnych

Zmienne statyczne - static

- Miejsce deklaracji: w bloku funkcyjnym jako automatyczne lub poza blokami funkcyjnymi, jako globalne
- Istnieją przez cały czas wykonywania programu, nawet po zakończeniu bloku funkcyjnego, w którym zostały zadeklarowane
- Zakres widzialności: zależny od sposobu deklaracji (automatyczne lub globalne)
- Zmienne **static** są automatycznie inicjalizowane zerem
- Mogą być inicjalizowane podczas deklaracji (tylko stałą wartością), inicjalizacja jest wykonywana tylko raz, podczas kompilacji programu

```
static int x = 10;
```

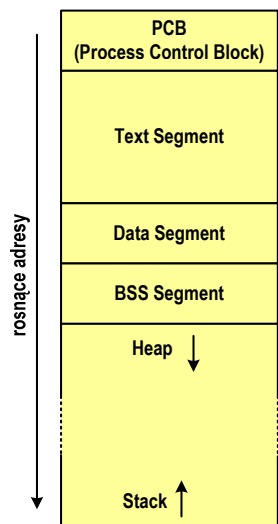
Klasy pamięci zmiennych

```
int a; /* extern int a; - zmienna globalna */
void foo();

int main(void)
{
    int b; /* auto int b; - zmienna lokalna */
    register float a; /* zmienna automatyczna, rejestrowa */
    foo(); foo(); foo();
    return 0;
}

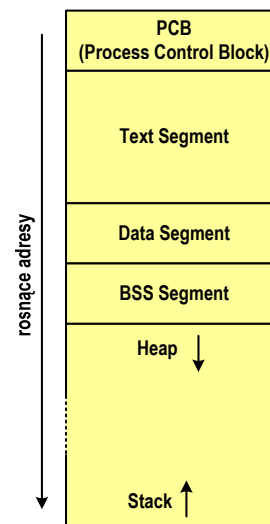
void foo()
{
    static int c = 1; /* zmienna statyczna */
    {
        double a; /* zmienna lokalna */
    }
    c++;
}
```

Struktura procesu w pamięci komputera



- **PCB** - blok kontrolny procesu
 - obszar pamięci operacyjnej zarezerwowany przez system operacyjny do zarządzania procesem
- **Text Segment**
 - kod programu czyli instrukcje w postaci binarnej
- **Data Segment**
 - zmienne globalne i statyczne zainicjalizowane niezerowymi wartościami
- **BSS Segment** (Block Started by Symbol)
 - zmienne globalne i statyczne domyślnie zainicjalizowane zerowymi wartościami

Struktura procesu w pamięci komputera



- **Heap** - sterta
 - obszar zmiennych dynamicznych
 - pamięć w obszarze sterty przydzielana jest funkcjami `calloc()` i `malloc()`
- **Stack** - stos
 - zmienne lokalne (automatyczne)
 - parametry funkcji i adresy powrotu z funkcji (stack frame)

Zmienne w pamięci komputera

```

int a; /* BSS Segment */
void foo();

int main(void)
{
    int b; /* Stack */
    float *a; /* Stack */
    a = (float *) malloc(400); /* Heap - 400 bajtów */
    return 0;
}

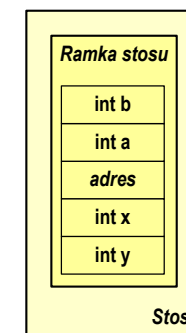
void foo()
{
    static int c = 1; /* Data Segment */
    {
        double a; /* Stack */
    }
    c++;
}
    
```

Ramka stosu (stack frame)

- Każde wywołanie funkcji powoduje odłożenie na stosie tzw. **ramki stosu**

```

void fun(int x, int y)
{
    int a, b;
}
    
```



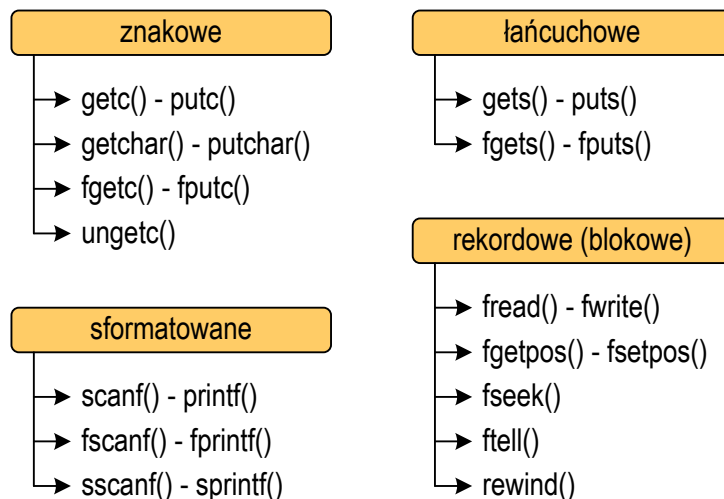
Programy wielomodułowe

(Przykład w Visual C++ 2008)

Operacje wejścia-wyjścia w języku C

- Operacje wejścia-wyjścia nie są elementami języka C
- Zostały zrealizowane jako funkcje zewnętrzne, znajdujące się w bibliotekach dostarczanych wraz z kompilatorem
- **Standardowe** wejście-wyjście (strumieniowe)
 - plik nagłówkowy `stdio.h`
 - duża liczba funkcji, proste w użyciu
 - ukrywa przed programistą szczegóły wykonywanych operacji
- **Systemowe** wejście-wyjście (deskryptorowe, niskopoziomowe)
 - plik nagłówkowy `io.h`
 - mniejsza liczba funkcji
 - programista sam obsługuje szczegóły wykonywanych operacji
 - funkcje bardziej zbliżone do systemu operacyjnego - działają szybciej

Typy standardowych operacji wejścia-wyjścia



Strumienie

- Standardowe operacje wejścia-wyjścia opierają się na **strumieniach** (ang. `stream`)
- Strumień jest pojęciem abstrakcyjnym - jego nazwa bierze się z analogii między przepływem danych, a np. wody
- W strumieniu dane płyną od źródła do odbiorcy
- Użytkownik określa źródło i odbiorcę, typ danych oraz sposób ich przesyłania
- Strumień może być skojarzony ze zbiorem danych znajdujących się na dysku (plik) lub zbiorem danych pochodzących z urządzenia znakowego (klawiatura)
- Niezależnie od fizycznego medium, z którym strumień jest skojarzony, wszystkie strumienie mają podobne właściwości

Strumienie

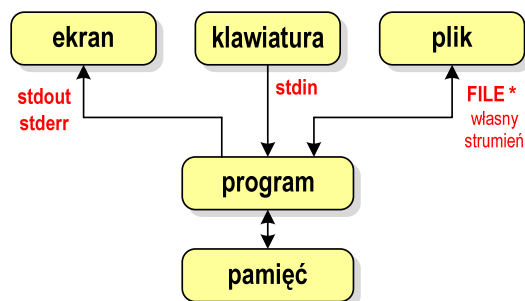
- Strumienie reprezentowane są przez zmienne będące wskaźnikami na struktury typu **FILE** (definicja w pliku **stdio.h**)

```
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 tej struktury

Strumienie

- Współpraca programu z „otoczeniem”



- Standardowe funkcje wejścia-wyjścia mogą:
 - domyślnie korzystać z określonego strumienia (**stdin**, **stdout**, **stderr**)
 - wymagać podania strumienia (własnego, **stdin**, **stdout**, **stderr**)

Strumienie

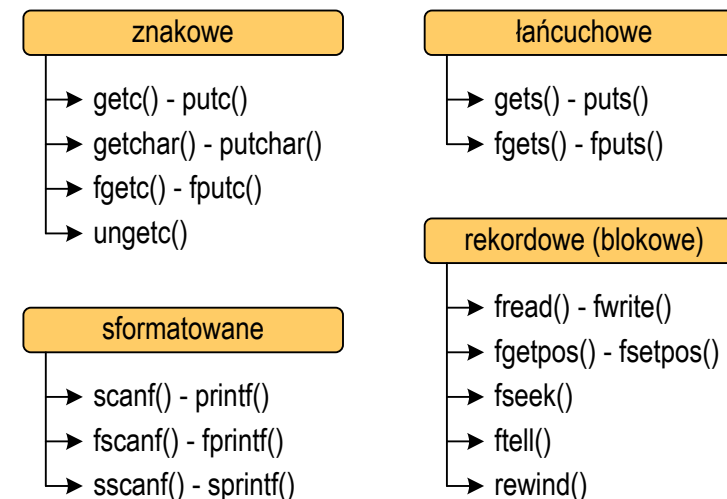
- W każdym programie automatycznie tworzone są i otwierane trzy standardowe strumienie wejścia-wyjścia:

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

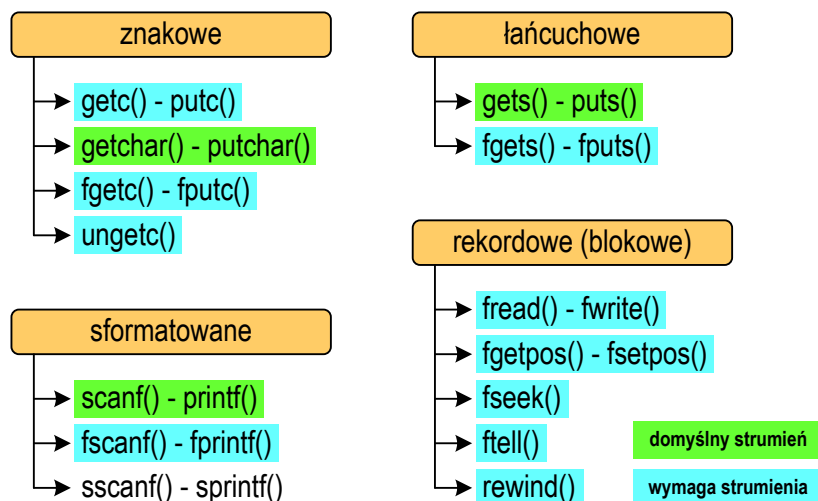
```
_CRTIMP FILE * __cdecl __iob_func(void);  
#define stdin (&__iob_func()[0])  
#define stdout (&__iob_func()[1])  
#define stderr (&__iob_func()[2])
```

- Funkcja **printf()** niejawnie używa strumienia **stdout**
- Funkcja **scanf()** niejawnie używa strumienia **stdin**

Typy standardowych operacji wejścia-wyjścia



Typy standardowych operacji wejścia-wyjścia



Otwarcie pliku - fopen()

FOPEN stdio.h

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

- Otwiera plik o nazwie **fname**, nazwa może zawierać całą ścieżkę dostępu do pliku
- **mode** określa tryb otwarcia pliku:
 - "r" - odczyt
 - "w" - zapis - jeśli pliku nie ma to zostanie on utworzony, jeśli plik istnieje, to jego poprzednia zawartość zostanie usunięta
 - "a" - zapis (dopisywanie) - dopisywanie danych na końcu istniejącego pliku, jeśli pliku nie ma to zostanie utworzony

Operacje na plikach

- Strumień wiąże się z plikiem za pomocą **otwarcia**, zaś połączenie to jest przerywane przez **zamknięcie** strumienia
- Operacje związane z przetwarzaniem pliku zazwyczaj składają się z trzech części

1. Otwarcie pliku (strumienia):

- funkcje: **fopen()**

2. Operacje na pliku (strumieniu), np. czytanie, pisanie:

- funkcje dla plików tekstowych: **fprintf(), fscanf(), fgetc(), fputc(), fgets(), fputs()...**

- funkcje dla plików binarnych: **fread(), fwrite(), ...**

3. Zamknięcie pliku (strumienia):

- funkcja: **fclose()**

Otwarcie pliku - fopen()

FOPEN stdio.h

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

- Otwiera plik o nazwie **fname**, nazwa może zawierać całą ścieżkę dostępu do pliku
- **mode** określa tryb otwarcia pliku:
 - "r+" - uaktualnienie (zapis i odczyt)
 - "w+" - uaktualnienie (zapis i odczyt) - jeśli pliku nie ma to zostanie on utworzony, jeśli plik istnieje, to jego poprzednia zawartość zostanie usunięta
 - "a+" - uaktualnienie (zapis i odczyt) - dopisywanie danych na końcu istniejącego pliku, jeśli pliku nie ma to zostanie utworzony, odczyt może dotyczyć całego pliku, zaś zapis może polegać tylko na dodawaniu nowych danych

Otwarcie pliku - fopen()

```
FOPEN stdio.h  
FILE* fopen(const char *fname, const char *mode);
```

- Zwraca wskaźnik na strukturę **FILE** skojarzoną z otwartym plikiem
- Gdy otwarcie pliku nie powiodło się to zwraca **NULL**
- Zawsze należy sprawdzać, czy otwarcie pliku powiodło się
- Po otwarciu pliku odwołujemy się do niego przez wskaźnik pliku
- Domyślnie plik jest otwierany w **trybie tekstowym**, natomiast dodanie litery **"b"** w trybie otwarcie oznacza **tryb binarny**

Zamknięcie pliku - fclose()

```
FCLOSE stdio.h  
int fclose(FILE *fp);
```

- Zamyka plik wskazywany przez **fp**
- Zwraca **0 (zero)** jeśli zamknięcie pliku było pomyślne
- W przypadku wystąpienia błędu zwraca **EOF**

```
#define EOF (-1)
```

- Po zamknięciu pliku, wskaźnik **fp** może być wykorzystany do otwarcia innego pliku
- W programie może być jednocześnie otwartych wiele plików

Otwarcie pliku - fopen()

- Otwarcie pliku w trybie tekstowym, tylko odczyt

```
FILE *fp;  
fp = fopen("dane.txt", "r");
```

- Otwarcie pliku w trybie binarnym, tylko zapis

```
fp = fopen("c:\\baza\\data.bin", "wb");
```

- Otwarcie pliku w trybie tekstowym, tylko zapis

```
fp = fopen("wynik.txt", "wt");
```

Przykład: otwarcie i zamknięcie pliku

```
#include <stdio.h>  
  
int main(void)  
{  
    FILE *fp;  
  
    fp = fopen("plik.txt", "w");  
    if (fp == NULL)  
    {  
        printf("Bład otwarcia pliku.\n");  
        return (-1);  
    }  
  
    /* przetwarzanie pliku */  
  
    fclose(fp);  
    return 0;  
}
```

Format (plik) tekstowy i binarny

- Przykład zawartości pliku tekstowego (Notatnik):

```
Plik (ang. file) - uporządkowany zbiór danych o skończonej długości,
posiadający szereg atrybutów i stanowiący dla użytkownika systemu
operacyjnego całość. Nazwa pliku nie jest częścią tego pliku,
lecz jest przechowywana w systemie plików.
```

- Przykład zawartości pliku binarnego (Notatnik):

```
MZ.....@.....c.....i...
LI!This program cannot be run in DOS mode...$ {900?XF!XF!XF!0!<X
F!i!i,XF!Z!|=XF!Xg!-XF!t.â;7XF!t.n!>XF!t.+:>XF!Rich?XF!
PE L..Zr.....8 : .....@ + .....|
| ..... + + + + .....e.< .. $!
. t. .... .textbss . + .....r.text
```

- Dane w pliku tekstowym zapisane są w postaci kodów ASCII
- Deklaracja i inicjalizacja zmiennej `x` typu `int`:

```
int x = 123456;
```

- W pamięci komputera zmienna `x` zajmuje 4 bajty:

```
00000000 00000001 11100010 01000000 (2)
```

- Po zapisaniu wartości zmiennej `x` do pliku **binarnego** znajdują się w nim 4 bajty o takiej samej zawartości jak w pamięci komputera

```
00000000 00000001 11100010 01000000 (2)
```

Format (plik) tekstowy i binarny

- Dane w pliku tekstowym zapisane są w postaci kodów ASCII
- Deklaracja i inicjalizacja zmiennej `x` typu `int`:

```
int x = 123456;
```

- W pamięci komputera zmienna `x` zajmuje 4 bajty:

```
00000000 00000001 11100010 01000000 (2)
```

- Po zapisaniu wartości zmiennej `x` do pliku **tekstowego** znajdzie się w nim 6 bajtów zawierających kody ASCII kolejnych cyfr

```
00110001 00110010 00110011 00110100 00110101 00110110 (2)
```

```
'1' '2' '3' '4' '5' '6' znaki
```

Format (plik) tekstowy i binarny

- Dane w pliku tekstowym zapisane są w postaci kodów ASCII
- Deklaracja i inicjalizacja zmiennej `x` typu `int`:

```
int x = 123456;
```

- W pamięci komputera zmienna `x` zajmuje 4 bajty:

```
00000000 00000001 11100010 01000000 (2)
```

- Po zapisaniu wartości zmiennej `x` do pliku **binarnego** znajdują się w nim 4 bajty o takiej samej zawartości jak w pamięci komputera

```
00000000 00000001 11100010 01000000 (2)
```

Format (plik) tekstowy i binarny

- Elementami pliku tekstowego są **wiersze** o różnej długości
- W systemach DOS/Windows każdy wiersz pliku tekstowego zakończony jest parą znaków:

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

- Załóżmy, że plik tekstowy ma postać:

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

- Rzeczywista zawartość pliku jest następująca:

```
50 69 65 72 77 73 7A 79|20 77 69 65 72 73 7A 20 | Pierwszy wiersz
70 6C 69 6B 75 0D 0A|44 72 75 67 69 20 77 69 65 | plikuDrugi wie
72 73 7A 20 70 6C 69 6B|75 0D 0A|54 72 7A 65 63 | rsz plikuTrzec
69 20 77 69 65 72 73 7A|20 70 6C 69 6B 75 0D 0A| i wiersz pliku
```


Format (plik) tekstowy i binarny

- W systemie Linux każdy wiersz pliku tekstowego zakończony jest tylko jednym znakiem:
 - LF (line feed) - przesunięcie o wiersz, kod ASCII - $10_{(10)} = 0A_{(16)} = '\n'$
- Założmy, że plik tekstowy ma postać:

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

- Rzeczywista zawartość pliku jest następująca:

```
50 69 65 72 77 73 7A 79 20 77 69 65 72 73 7A 20 | Pierwszy wiersz
70 6C 69 6B 75 0A 44 72 75 67 69 20 77 69 65 72 | plikuDrugi wier
73 7A 20 70 6C 69 6B 75 0A 54 72 7A 65 63 69 20 | sz plikuTrzeci
77 69 65 72 73 7A 20 70 6C 69 6B 75 0A | wiersz pliku
```

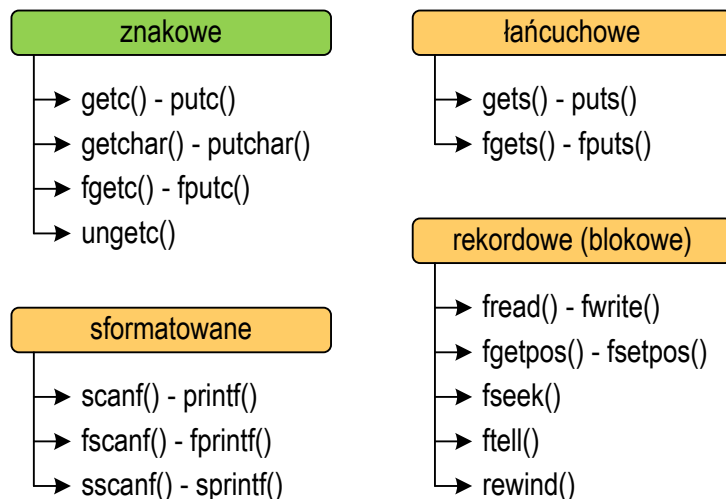
- Pliki **binarne** nie mają ściśle określonej struktury

Tryby otwarcia pliku: tekstowy i binarny

```
FILE *fp1, *fp2;
fp1 = fopen("dane.txt", "r"); // lub "rt"
fp2 = fopen("dane.dat", "rb")
```

- Różnice pomiędzy trybem tekstowym i binarnym otwarcia pliku dotyczą innego traktowania znaków **CR** i **LF**
- W trybie **tekstowym**:
 - przy odczycie pliku para znaków **CR, LF** jest tłumaczona na znak nowej linii (**LF**)
 - przy zapisie pliku znak nowej linii (**LF**) jest zapisywany w postaci dwóch znaków (**CR, LF**)
- W trybie **binarnym**:
 - przy odczycie i zapisie para znaków **CR, LF** jest traktowana zawsze jako dwa znaki

Znakowe operacje wejścia-wyjścia



Znakowe operacje wejścia-wyjścia

```
GETC stdio.h
int getc(FILE *fp);
```

- Pobiera jeden znak z aktualnej pozycji otwartego strumienia **fp** i uaktualnia pozycję
- Zmienna **fp** powinna wskazywać strukturę **FILE** reprezentującą strumień skojarzony z otwartym plikiem lub jeden ze standardowo otwartych strumieni (np. **stdin**)
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wartość całkowitą **kodu** wczytanego znaku (typ **int**)
- Jeśli wystąpił błąd lub przeczytany został znacznik końca pliku, to funkcja zwraca wartość **EOF**

Przykład: wyświetlenie pliku tekstowego

```
#include <stdio.h>

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

    fp = fopen("test.txt", "r");
    znak = getc(fp);
    while (znak != EOF)
    {
        printf("%c", znak);
        znak = getc(fp);
    }

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

Przykład: wyświetlenie pliku tekstowego

- Odczytanie i wyświetlenie zawartości pliku tekstowego

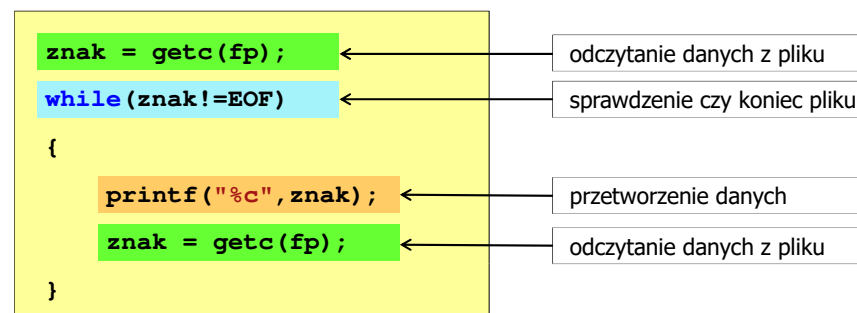
```
znak = getc(fp);
while (znak != EOF)
{
    printf("%c", znak);
    znak = getc(fp);
}
```

można zapisać w krótszej postaci:

```
while ((znak=getc(fp)) != EOF)
    printf("%c", znak);
```

Schemat przetwarzania pliku

- Typowy schemat odczytywania danych z pliku



Znakowe operacje wejścia-wyjścia

```
PUTC stdio.h
int putc(int znak, FILE *fp);
```

- Wpisuje **znak** do otwartego strumienia reprezentowanego przez argument **fp**
- Zmienna **fp** powinna wskazywać strukturę **FILE** reprezentującą strumień skojarzony z otwartym plikiem lub jeden ze standardowo otwartych strumieni (np. **stdout**)
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wypisany **znak**
- Jeśli wystąpił błąd, to funkcja zwraca wartość **EOF**

Przykład: zapisanie alfabetu do pliku tekstowego

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

```
int main(void)
```

```
{  
    FILE *fp = fopen("alfabet.txt", "w");  
    for (int i='A'; i<='Z'; i++)  
        putchar(i, fp);  
    fclose(fp);  
    return 0;  
}
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

- Stosując strumień `stdout` można wyświetlić alfabet na ekranie

```
for (int i='A'; i<='Z'; i++)  
    putchar(i, stdout);
```

```
abcdefghijklmnopqrstuvwxyz
```

Znakowe operacje wejścia-wyjścia

GETCHAR

`stdio.h`

```
int getchar(void);
```

- Pobiera znak ze strumienia `stdin` (klawiatura)
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca przeczytany znak (typ `int`)
- Jeśli wystąpił błąd albo został przeczytany znacznik końca pliku, to funkcja zwraca wartość `EOF`

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

Znakowe operacje wejścia-wyjścia

PUTCHAR

`stdio.h`

```
int putchar(int znak);
```

- Wpisuje `znak` do strumienia `stdout` (standardowo ekran)
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wypisany `znak`
- Jeśli wystąpił błąd, to funkcja zwraca wartość `EOF`

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

```
abcdefghijklmnopqrstuvwxyz
```

Przykład: liczba znaków wczytanych z klawiatury

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

```
int main(void)
```

```
{  
    int znak, ile = 0;  
    while ((znak=getchar()) != '\n')  
        ile++;  
    printf("Liczba znakow: %d\n", ile);  
    return 0;  
}
```

```
Ala ma laptopa  
Liczba znakow: 14
```

- Wprowadzane znaki są buforowane do naciśnięcia klawisza `Enter`
- Po naciśnięciu klawisza `Enter` zawartość bufora jest przesyłana do programu i analizowana w nim

Znakowe operacje wejścia-wyjścia

FGETC stdio.h

```
int fgetc(FILE *fp);
```

- Pobiera jeden znak ze strumienia wskazywanego przez **fp**
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca przeczytany znak po przekształceniu go na typ **int**
- Jeśli wystąpił błąd lub został przeczytany znacznik końca pliku, to funkcja zwraca wartość **EOF**

Znakowe operacje wejścia-wyjścia

FPUTC stdio.h

```
int fputc(int znak, FILE *fp);
```

- Wpisuje **znak** do otwartego strumienia reprezentowanego przez argument **fp**
- Jeśli wykonanie zakończyło się poprawnie, to funkcja zwraca wypisany **znak** (typ **int**)
- Jeśli wystąpił błąd, to funkcja zwraca wartość **EOF**

Przykład: liczba wyrazów w pliku

```
#include <stdio.h>

int main(void)
{
    FILE *fp;
    int znak, odstep = 1, ile = 0;

    fp = fopen("test.txt", "r");
    while ((znak = fgetc(fp)) != EOF)
        if (znak == ' ' || znak == '\t' || znak == '\n')
            odstep = 1;
        else
            if (odstep != 0) { odstep = 0; ile++; }
    fclose(fp);
    printf("Liczba slow: %d\n", ile);

    return 0;
}
```

Ala ma laptopa i psa.

Liczba slow: 5

Znakowe operacje wejścia-wyjścia

UNGETC stdio.h

```
int ungetc(int znak, FILE *fp);
```

- Umieszcza **znak** z powrotem w strumieniu wejściowym **fp**

Koniec wykładu nr 4

Dziękuję za uwagę!