Programowanie w języku C i programowanie obiektowe

dr inż. Bartosz Jędrzejec

5 marca 2025

Podstawowe informacje o prowadzącym

Imię i Nazwisko: **Bartosz Jędrzejec**

Miejsce pracy: **Katedra Informatyki i Automatyki**

Budynek D pokój D202a

Adres e-mail: **bartoszj@kia.prz.edu.pl**

bartoszj@prz.edu.pl

Karta modułu kształcenia

Jest dostępna na stronie Wydziału Elektrotechniki i Informatyki
↪ `https://weii.prz.edu.pl/`
↪ STUDENCI
↪ Plany studiów
↪ 2024/2025
↪ Automatyka i robotyka
↪ Komputerowe systemy sterowania
↪ Programowanie w języku C i programowanie obiektowe

Forma zaliczenia przedmiotu

- Pozytywne zaliczenie 2 części laboratorium warunkuje dopuszczenie do egzaminu
- Pozytywna ocena z egzaminu

Forma zaliczenia przedmiotu

- Pozytywne zaliczenie 2 części laboratorium warunkuje dopuszczenie do egzaminu
- Pozytywna ocena z egzaminu

Zwolnienie z egzaminu

Ocena $\geqslant$ **4.5** z każdej z **2** części laboratorium

Plan wykładu z języka ANSI C

- Pierwszy program
- Podstawowe elementy języka C
  - Zestaw znaków
  - Słowa kluczowe
  - Typy danych
  - Literały i stałe
  - Zmienne
  - Wyrażenia
- Operacje wejścia-wyjścia
- Operatory
- Instrukcje
- Funkcje
- Tablice
- Wskaźniki
- Struktury
- Pliki

Plan wykładu z programowania obiektowego

- Strumieniowe operacje we/wy
- Klasy
- Dynamiczna alokacja pamięci
- Konstruktor, destruktor
- Składnik statyczny klasy
- Konstruktor kopiujący
- Przeciążanie operatorów
- Lista inicjalizacyjna konstruktora
- Dziedziczenie

Literatura

- Kernighan Brian W., Ritchie Dennis M., Język ANSI C, WNT, Warszawa, 1998
- **Jędrzejec B., Sadolewski J., Programowanie w języku C i C++, Oficyna Wyd. Politechniki Rzeszowskiej, 2017**
- Prata S., Język C szkoła programowania, Wydawnictwo Robomatic, Wrocław, 1999
- Delannoy C., Ćwiczenia z języka C, WNT, Warszawa, 1993
- Cormen Thomas H., Leiserson Charles E., Rivest Ronald L., Wprowadzenie do algorytmów, WNT, Warszawa, 1997
- Aho Alfred V., Ullman Jeffrey D., Wykłady z informatyki z przykładami w języku C, Wydawnictwo Helion, Gliwice, 2003

Historia

- Język C został stworzony w **1972** roku przez Denisa Ritchie'go na bazie języka B.

Historia

- Język C został stworzony w **1972** roku przez Denisa Ritchie'go na bazie języka B.
- Brian Kernighan i Dennis Ritchie publikują podręcznik "The C Programming Language". (**1978**)

Historia

- Język C został stworzony w **1972** roku przez Denisa Ritchie'go na bazie języka B.
- Brian Kernighan i Dennis Ritchie publikują podręcznik "The C Programming Language". (**1978**)
- ANSI C (**1989**)

Historia

- Język C został stworzony w **1972** roku przez Denisa Ritchie'go na bazie języka B.
- Brian Kernighan i Dennis Ritchie publikują podręcznik "The C Programming Language". (**1978**)
- ANSI C (**1989**)
- ISO C (**1990**)

Historia

- Język C został stworzony w **1972** roku przez Denisa Ritchie'go na bazie języka B.
- Brian Kernighan i Dennis Ritchie publikują podręcznik "The C Programming Language". (**1978**)
- ANSI C (**1989**)
- ISO C (**1990**)
- Standard C99 (**1999**) - (m.in. funkcje inline, deklaracja zmiennych w dowolnym miejscu programu, nowe funkcje i pliki nagłówkowe, nowe typy wbudowane)

Historia

- Język C został stworzony w **1972** roku przez Denisa Ritchie'go na bazie języka B.
- Brian Kernighan i Dennis Ritchie publikują podręcznik "The C Programming Language". (**1978**)
- ANSI C (**1989**)
- ISO C (**1990**)
- Standard C99 (**1999**) - (m.in. funkcje inline, deklaracja zmiennych w dowolnym miejscu programu, nowe funkcje i pliki nagłówkowe, nowe typy wbudowane)
- Standardy C11 (**2011**) - (m.in. wątki, Unicode) i C18 (**2018**)

Zalety języka C

- język programowania do celów ogólnych

Zalety języka C

- język programowania do celów ogólnych
- szybkość działania

Zalety języka C

- język programowania do celów ogólnych
- szybkość działania
- zwięzły kod źródłowy – niewielki rozmiar programów

Zalety języka C

- język programowania do celów ogólnych
- szybkość działania
- zwięzły kod źródłowy – niewielki rozmiar programów
- przenośność i dostępność na różne platformy

Zalety języka C

- język programowania do celów ogólnych
- szybkość działania
- zwięzły kod źródłowy – niewielki rozmiar programów
- przenośność i dostępność na różne platformy

Wady języka C

- elastyczność języka C wymaga większej odpowiedzialności od programisty

Zalety języka C

- język programowania do celów ogólnych
- szybkość działania
- zwięzły kod źródłowy – niewielki rozmiar programów
- przenośność i dostępność na różne platformy

Wady języka C

- elastyczność języka C wymaga większej odpowiedzialności od programisty
- podatność na błędy (wskaźniki, zarządzanie pamięcią)

Zalety języka C

- język programowania do celów ogólnych
- szybkość działania
- zwięzły kod źródłowy – niewielki rozmiar programów
- przenośność i dostępność na różne platformy

Wady języka C

- elastyczność języka C wymaga większej odpowiedzialności od programisty
- podatność na błędy (wskaźniki, zarządzanie pamięcią)
- nie posiada wbudowanych bibliotek do typowych struktur danych i algorytmów

```
/*Program p1.c
  Obliczanie objetosci walca */
#include <stdio.h>
void main()
{
    float promien, wysokosc, objetosc;
    promien = 3.3;
    wysokosc = 44.4;
    objetosc = 3.1415926 * promien * promien * wysokosc;
    printf("Objetosc walca = %f",objetosc);
}
```

```
Objetosc walca = 1519.010254
```

```
//Program p1.c
//Obliczanie objetosci walca
#include <stdio.h>
void main()
{
    float promien, wysokosc, objetosc;
    promien = 3.3;
    wysokosc = 44.4;
    objetosc = 3.1415926 * promien * promien * wysokosc;
    printf("Objetosc walca = %f",objetosc);
}
```

```
Objetosc walca = 1519.010254
```

```
//Program p1.c
//Obliczanie objetosci walca
#include <stdio.h>
void main()
{
    float promien, wysokosc, objetosc;
    promien = 3.3;
    wysokosc = 44.4;
    objetosc = 3.1415926 * promien * promien * wysokosc;
    printf("Objetosc walca = %f",objetosc);
}
```

```
Objetosc walca = 1519.010254
```

```
//Program p1.c
//Obliczanie objetosci walca
#include <stdio.h>
void main()
{
    float promien, wysokosc, objetosc;
    promien = 3.3;
    wysokosc = 44.4;
    objetosc = 3.1415926 * promien * promien * wysokosc;
    printf("Objetosc walca = %f",objetosc);
}
Objetosc walca = 1519.010254
```

```
//Program p1.c
//Obliczanie objetosci walca
#include <stdio.h>
void main(void)
{
    float promien, wysokosc, objetosc;
    promien = 3.3;
    wysokosc = 44.4;
    objetosc = 3.1415926 * promien * promien * wysokosc;
    printf("Objetosc walca = %f",objetosc);
}
Objetosc walca = 1519.010254
```

```
//Program p1.c
//Obliczanie objetosci walca
#include <stdio.h>
int main()
{
    float promien, wysokosc, objetosc;
    promien = 3.3;
    wysokosc = 44.4;
    objetosc = 3.1415926 * promien * promien * wysokosc;
    printf("Objetosc walca = %f",objetosc);
    return 0;
}
Objetosc walca = 1519.010254
```

```
//Program p1.c
//Obliczanie objetosci walca
#include <stdio.h>
main()
{
    float promien, wysokosc, objetosc;
    promien = 3.3;
    wysokosc = 44.4;
    objetosc = 3.1415926 * promien * promien * wysokosc;
    printf("Objetosc walca = %f",objetosc);
    return 0;
}
Objetosc walca = 1519.010254
```

```
//Program p1.c
//Obliczanie objetosci walca
#include <stdio.h>
main(int argc, const char* argv[])
{
    float promien, wysokosc, objetosc;
    promien = 3.3;
    wysokosc = 44.4;
    objetosc = 3.1415926 * promien * promien * wysokosc;
    printf("Objetosc walca = %f",objetosc);
    return 0;
}
Objetosc walca = 1519.010254
```

```
//Program p1.c
//Obliczanie objetosci walca
#include <stdio.h>
main(int argc, const char* argv[])
{
    float promien, wysokosc, objetosc;
    promien = 3.3;
    wysokosc = 44.4;
    objetosc = 3.1415926 * promien * promien * wysokosc;
    printf("Objetosc walca = %f",objetosc);
    return 0;
}
```

Objetosc walca = 1519.010254

```
//Program p1.c
//Obliczanie objetosci walca
#include <stdio.h>
main(int argc, const char* argv[])
{
    float promien, wysokosc, objetosc;
    promien = 3.3;
    wysokosc = 44.4;
    objetosc = 3.1415926 * promien * promien * wysokosc;
    printf("Objetosc walca = %f",objetosc);
    return 0;
}
Objetosc walca = 1519.010254
```

```
//Program p1.c
//Obliczanie objetosci walca
#include <stdio.h>
main(int argc, const char* argv[])
{
    float promien, wysokosc, objetosc;
    promien = 3.3;
    wysokosc = 44.4;
    objetosc = 3.1415926 * promien * promien * wysokosc;
    printf("Objetosc walca = %f",objetosc);
    return 0;
}
```

Objetosc walca = 1519.010254

Zestaw znaków

- duże litery alfabetu łacińskiego (od A do Z)

Zestaw znaków

- duże litery alfabetu łacińskiego (od A do Z)
- małe litery alfabetu łacińskiego (od a do z)

Zestaw znaków

- duże litery alfabetu łacińskiego (od A do Z)
- małe litery alfabetu łacińskiego (od a do z)
- cyfry (od 0 do 9)

Zestaw znaków

- duże litery alfabetu łacińskiego (od A do Z)
- małe litery alfabetu łacińskiego (od a do z)
- cyfry (od 0 do 9)
- znaki specjalne `! * + \ " < # ( = | { > % ) ~ ; } / ^ - [ : , ? & _ ] ‘ .`

Zestaw znaków

- duże litery alfabetu łacińskiego (od A do Z)
- małe litery alfabetu łacińskiego (od a do z)
- cyfry (od 0 do 9)
- znaki specjalne `! * + \ " < # ( = | { > % ) ~ ; } / ^ - [ : , ? & _ ] ‘ .`
- znaki dodatkowe jak `@`, `$` uwzględniane w łańcuchach znaków lub w komentarzach

Zestaw znaków

- duże litery alfabetu łacińskiego (od A do Z)
- małe litery alfabetu łacińskiego (od a do z)
- cyfry (od 0 do 9)
- znaki specjalne `! * + \ " < # ( = | { > % ) ~ ; } / ^ - [ : , ? & _ ] ‘ .`
- znaki dodatkowe jak `@`, `$` uwzględniane w łańcuchach znaków lub w komentarzach
- kombinacje znaków, jak np. `\b`, `\n`, `\t` tzw. *Escape-sekwencje* reprezentujące jeden znak

Słowa kluczowe

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

„Niestandardowe” słowa kluczowe

ada	far	near	entry
asm	fortran	pascal	huge

Typy podstawowe

`int`	reprezentuje liczbę całkowitą
`char`	reprezentuje pojedynczy znak (np. literę alfabetu)
`float`	reprezentuje liczbę rzeczywistą (pojedynczej precyzji)
`double`	reprezentuje liczbę rzeczywistą (podwójnej precyzji)

Typy podstawowe

`int`	reprezentuje liczbę całkowitą
`char`	reprezentuje pojedynczy znak (np. literę alfabetu)
`float`	reprezentuje liczbę rzeczywistą (pojedynczej precyzji)
`double`	reprezentuje liczbę rzeczywistą (podwójnej precyzji)

Modyfikatory

`short`	oznacza liczbę krótką
`long`	oznacza liczbę długą
`signed`	oznacza liczbę ze znakiem
`unsigned`	oznacza liczbę bez znaku

Uzupełnienie typów podstawowych

`void`	oznacza brak wartości zwracanej przez funkcję
`enum`	typ wyliczeniowy - rozszerzenia języka C
wskaźnikowy `*`	reprezentuje zmienne typu wskaźnikowego

Uzupełnienie typów podstawowych

`void`	oznacza brak wartości zwracanej przez funkcję
`enum`	typ wyliczeniowy - rozszerzenia języka C
wskaźnikowy `*`	reprezentuje zmienne typu wskaźnikowego

Możliwy do zadeklarowania zestaw typów danych

`char`, `unsigned char`

`int`, `unsigned`, `signed`, `short`, `unsigned short`, `long`, `unsigned long`

`float`, `double`, `long double`

Literały i stałe

1. **literały całkowitoliczbowe**
   - **dziesiętne**
     zestaw znaków: 0 1 2 3 4 5 6 7 8 9 + -
     przykłady: `125, -89, 0, +23`
   - **ósemkowe**
     zestaw znaków: 0 1 2 3 4 5 6 7 + - (pierwszym znakiem jest zero)
     przykłady: `0125, 0, 072, -012`
   - **szesnastkowe**
     zestaw znaków: 0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F + - (pierwszymi znakami są zero i małe lub duże x - 0x, 0X)
     przykłady: `0x125, 0xaa, 0X12C, -0xc1b`
2. **literały rzeczywiste**
   zestaw znaków: 0 1 2 3 4 5 6 7 8 9 . + - e E
   przykłady: `1.2, 0., 0.5e3, -125E-2`

Literały i stałe

3. **literały znakowe**
   pojedynczy znak ujęty w apostrof
   zestaw znaków:
   - cały repertuar „widocznych” znaków ASCII
   - escape-sekwencje

   przykłady: `'A', 'b', '+', '0', '\n', '\\'`
4. **literały łańcuchowe**
   ciąg dowolnych znaków ujętych w cudzysłowy
   przykłady: `"To jest lancuch", "A+B=C", "\n tekst"`
5. **stałe symboliczne**
   jest to nazwa zastępująca łańcuch znaków
   `#define NAZWA tekst`
   przykłady:
   `#define PI 3.1415926`,
   `#define WZOR 4*a*b*b`

Zmienne

Zmienna jest to nazwa, której przypisany jest typ danych.

Nazwa jest to ciąg literowo-cyfrowy zaczynający się od litery.

Przykłady deklaracji zmiennych:

```
char znak;
int tab[10], a;
float x=2.5;
int tab[5] = {1, 5, 10, 2, 0};
char tekst[] = "To jest lancuch";
```

Wyrażenia

Wyrażenie to połączenie zmiennych i stałych przy pomocy operatorów.

Przykłady wyrażeń:

```
x = 2 * a * b
```

```
a >= b
```

```
a > b ? a : b
```

```
a > b && a > 0
```

```
a == b || a != c
```

Funkcje wejścia-wyjścia

Funkcje biblioteczne modułu `stdio.h`.

1. Funkcje wyjścia
   - `putchar`
   - `puts`
   - `printf`
2. Funkcje wejścia
   - `getchar`
   - `gets`
   - `scanf`

Funkcja putchar

Funkcja wyprowadzenia pojedynczego znaku.

Ogólna postać:

```
int putchar(int character);
```

Parametr `character` jest wyrażeniem typu całkowitego.

przykłady:

```
char znak='K';
int spacja=32;
putchar('\n');
putchar('A');
putchar(znak);
putchar(spacja);
putchar(znak+'\066'-52);
```

Wynik:

↵
A
K
␣
M

Funkcja puts

Funkcja wyprowadzenia łańcucha znaków.

Ogólna postać:

```
int puts(const char * str);
```

Parametr str może być stałą lub zmienną łańcuchową.

przykłady:

```
char tekst[] = "To jest lancuch";
puts(tekst);
puts("Praktyczna funkcja");
puts("\066\067\x2b\X2A itd.");
```

Wynik:

```
To jest lancuch
Praktyczna funkcja
67+* itd.
```

Funkcja `printf`

Funkcja wyprowadzenia wartości numerycznych, znakowych i łańcuchowych.

Ogólna postać:

```
int printf(const char * format, ...);
```

Parametr `format` stanowi wzorzec wyprowadzanych wyników.

Parametr `...` jest listą argumentów.

Pełny zapis instrukcji formatu:

%[Pole znaku][Szerokość][.Dokładność][Modyfikator]Typ danych

Pole znaku: `+`, `-`, `␣`, `#`, `0`
Szerokość: całkowita ilość znaków wyprowadzanych wartości
Dokładność: ilość miejsc po przecinku dla liczb rzeczywistych i długość łańcucha znaków
Modyfikator: `h` - `short`, `l` - `long`/`double`, `L` - `long double`

Funkcja printf

Typ danych wyprowadzanych wyników:

format	typ
d, i	int (dziesiętnie)
u	unsigned
o	int (ósemkowo)
x,X	unsigned (szestnastkowo)
f, e, E, g, G	float/double
c	char
s	char*
n	int*
p	pointer

Funkcja printf

przykłady:

```
printf("%c %c", 'a', 65);
```
Wynik: `a A`

```
printf("%d %ld", 1977, 650000L);
```
Wynik: `1977 650000`

```
printf("%10d", 1977);
```
Wynik: ␣␣␣␣␣␣1977

```
printf("%010d", 1977);
```
Wynik: `0000001977`

Funkcja printf

przykłady:

```
printf("%d %x %o %#x %#o", 100, 100, 100, 100, 100);
```
Wynik: `100 64 144 0x64 0144`

```
printf("%4.2f %+.0e %E", 3.1416, 3.1416, 3.1416);
```
Wynik: `3.14 +3e+000 3.141600E+000`

```
printf("%*.*f", 7, 2, 10.1234);
```
Wynik: `␣␣10.12`

```
printf("%s", "Lancuch");
```
Wynik: `Lancuch`

Funkcja getchar

Funkcja wczytania pojedynczego znaku.

Ogólna postać:

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

Funkcja zwraca kod wczytanego znaku po akceptacji klawiszem ↵

przykład:

```
char znak;
znak=getchar();      fflush(stdin);
```

Bliźniacze funkcje w module `conio.h`, nie potrzebują akceptacji ↵

```
int getche(void);
```

funkcja wyprowadza znak na ekran

```
int getch(void);
```

funkcja nie wyprowadza znaku na ekran

Funkcja gets

Funkcja wprowadzenia jednej linii tekstu zakończonej ↵.

Ogólna postać:

```
char * gets(char * str);
```

Parametr `str` jest zmienną łańcuchową.

Funkcja nie sprawdza długości wczytanego łańcucha.
przykład:

```
char tekst[10];
gets(tekst);
```

Bliźniacza funkcja do obsługi plików:

```
char * fgets (char * str, int num, FILE * stream);
```

przykład:

```
fgets(tekst, 10, stdin);
```

Funkcja `scanf`

Funkcja wprowadzenia danych różnego typu.

Ogólna postać:

```
int scanf(const char * format, ...);
```

Parametr `format` stanowi wzorzec wprowadzanych danych.

Parametr `...` jest listą argumentów.

Format dla typów danych jak dla funkcji **`printf`**.
przykłady:

```
int a;
double b;
char tekst[10];
scanf("%d",&a);
scanf("%lf",&b);
scanf("%9[^\n]s",tekst);
```

Operatory

1. **operatory arytmetyczne**:
   - `+` dodawanie
   - `-` odejmowanie
   - `*` mnożenie
   - `/` dzielenie
   - `%` reszta z dzielenia liczb całkowitych

   przykłady:

   `4 + b * b - c`,

   `5 / 2` (wartość 2),

   `5./ 2` (wartość 2.5),

   `5 % 2` (wartość 1)

Operatory

2. **operatory porównania**:
   - `==` równe
   - `!=` różne
   - `<` mniejsze
   - `<=` mniejsze lub równe
   - `>` większe
   - `>=` większe lub równe

Wartością wyrażenia logicznego jest wartość **0** lub **1**.

przykłady:

```
int a=1, b=2, x=0, delta=-2;
```

`delta > 0`	**Wynik:** 0
`a != b`	**Wynik:** 1
`a == 0`	**Wynik:** 0
`x <= 2`	**Wynik:** 1

Operatory

3. **operatory logiczne**:
   - `&&` logiczne AND (i) /iloczyn/
   - `||` logiczne OR (lub) /alternatywa/
   - `!` logiczne NOT (nie) /negacja/

Wartością wyrażenia logicznego jest wartość **0** lub **1**.

przykłady:

```
int a=1, b=2, c;
c = (a>1) && (b=3);      Wynik: a=1, b=2, c=0
c = (a<=1) && (b=3);     Wynik: a=1, b=3, c=1
c = (a<=1) || (b=3);     Wynik: a=1, b=2, c=1
c = (a>1) || (b=3);      Wynik: a=1, b=3, c=1
c = !a;                  Wynik: a=1, c=0
```

Operatory

4. **operatory bitowe**:
   - `&` bitowa koniunkcja (AND)
   - `|` bitowa alternatywa (OR)
   - `^` bitowa różnica symetryczna (XOR)
   - `~` uzupełnienie jedynkowe
   - `<<` przesunięcie w lewo
   - `>>` przesunięcie w prawo

przykłady:

```
unsigned int k=0x55, m=0x33;
  k    0x55   01010101
  m    0x33   00110011
```

Operatory

4. **operatory bitowe**:
   - `&` bitowa koniunkcja (AND)
   - `|` bitowa alternatywa (OR)
   - `^` bitowa różnica symetryczna (XOR)
   - `~` uzupełnienie jedynkowe
   - `<<` przesunięcie w lewo
   - `>>` przesunięcie w prawo

przykłady:

```
unsigned int k=0x55, m=0x33;
  k    0x55   01010101
  m    0x33   00110011
 k&m   0x11   00010001
 k|m   0x77   01110111
```

Operatory

4. **operatory bitowe**:
   - `&` bitowa koniunkcja (AND)
   - `|` bitowa alternatywa (OR)
   - `^` bitowa różnica symetryczna (XOR)
   - `~` uzupełnienie jedynkowe
   - `<<` przesunięcie w lewo
   - `>>` przesunięcie w prawo

przykłady:

```
unsigned int k=0x55, m=0x33;
  k    0x55   01010101
  m    0x33   00110011
 k^m   0x66   01100110
 ~m    0xcc   11001100
```

Operatory

4. **operatory bitowe**:
   - `&` bitowa koniunkcja (AND)
   - `|` bitowa alternatywa (OR)
   - `^` bitowa różnica symetryczna (XOR)
   - `~` uzupełnienie jedynkowe
   - `<<` przesunięcie w lewo
   - `>>` przesunięcie w prawo

przykłady:

```
unsigned int k=0x55, m=0x33;
```

k	0x55	01010101
m	0x33	00110011
m<<2	0x66	11001100
k>>3	0x0a	00001010

Operatory

5. **operatory przypisania**:

operator	przykład zapisu	zapis równoważny
=	a = b	a = b
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b
<<=	a <<= b	a = a << b
>>=	a >>= b	a = a >> b
&=	a &= b	a = a & b
^=	a ^= b	a = a ^ b
|=	a |= b	a = a | b

Operatory

5. **operatory przypisania**:

przykłady:

```
int a=1, b=2, c;
a+=b;
c=a*=b+=2;
a>>=2;
```

Wynik: a=3, b=2

Wynik: a=12, b=4, c=12

Wynik: a=3

Operatory

6. **operatory unarne**:
   - `++` inkrementacji (zwiększenia)
   - `--` dekrementacji (zmniejszenia)
   - `-` minus jednoargumentowy
   - `+` plus jednoargumentowy (bez zmiany znaku).

Zapis: `++i; i++;` $\Rightarrow$ `i+=1;` $\Rightarrow$ `i=i+1;`

przykłady:

```
int a=1, b=2, c;
c=a++;
c=++a;
c*=a--;
a=-b;
```

Wynik: a=2, c=1

Wynik: a=3, c=3

Wynik: a=2, c=9

Wynik: a=-2, b=2

Operatory

7. **operator rozmiaru**:
   - `sizeof wyrażenie`
   - `sizeof (nazwa typu)`

   Operator rozmiaru określa liczbę bajtów potrzebnych do zapisu wyrażenia lub typu.

   przykłady:

```
long double r;
sizeof(long double)    Wynik: 10
sizeof r               Wynik: 10
char tekst[]="Taki sobie lancuch";
sizeof tekst           Wynik: 19
char tekst[80]="Taki sobie lancuch";
sizeof tekst           Wynik: 80
```

Operatory

8. **operator konwersji**:
   - `(nazwa typu) wyrażenie`

   przykłady:

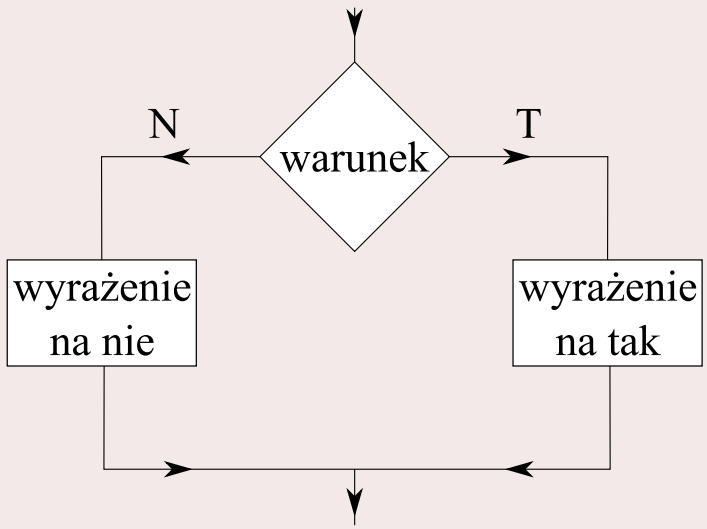
```
int a;
double c=125.75;
a=(int) c;
(double) 5/2
```

Wynik: a=125

Wynik: 2.5

Operatory

9. **operator warunkowy**:
   - `warunek ? wyrażenie na tak : wyrażenie na nie`



przykład:

```
max=a>b?a:b;
```

Wynik: max=większa z dwóch liczb a i b

Operatory

10. **operator przecinkowy**:
    - `w1,w2,w3,...,wn`

    Kompilator oblicza kolejno wartości wyrażeń w1, w2, w3,...,wn i przyjmuje za wynik całego wyrażenia wartość wn.

    przykłady:

    `pi=(3,14);` **Wynik:** pi=14

    `pi=3,14;` **Wynik:** pi=3

Operatory

11. **operatory wskazywania**:
    - `&` adresu
    - `*` adresowania pośredniego
    - `.` składowej
    - `->` wskaźnikowy składowej

Instrukcje

Instrukcja jest to ciąg znaków, składniowo poprawny, zakończony średnikiem.

1. **instrukcja złożona**:
   Zapis:
   ```
   {
     ciąg deklaracji
     ciąg instrukcji
   }
   ```
   przykład:
   ```
   {
     int a=2, b=-3;
     printf("Miejsce zerowe = %g", -(double)b/a);
   }
   ```

Instrukcje

2. **instrukcja przypisania:**
   Zapis:
   `zmienna operator przypisania wyrażenie;`
   przykład:

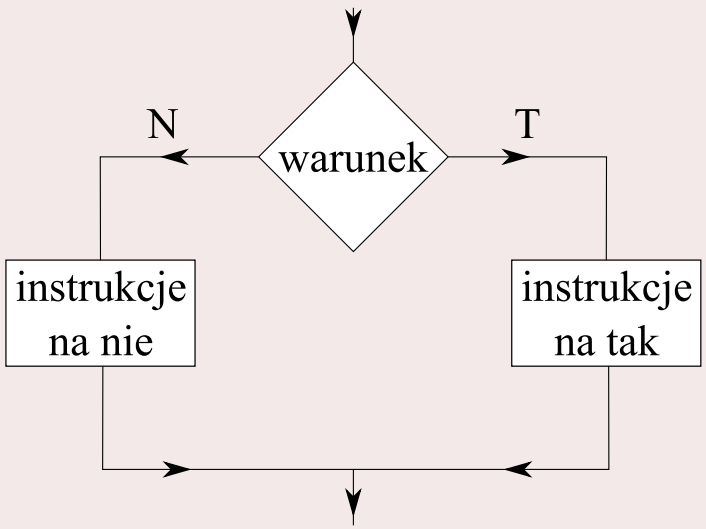
```
v=3.1415926*r*r*h;
a=b=c=d=5;
a+=b+=c+=5;
```

Instrukcje

3. **instrukcja warunkowa `if`:**

   Zapis pełny:

```
if (warunek)
{
  lista instrukcji na tak
}
else
{
  lista instrukcji na nie
}
```

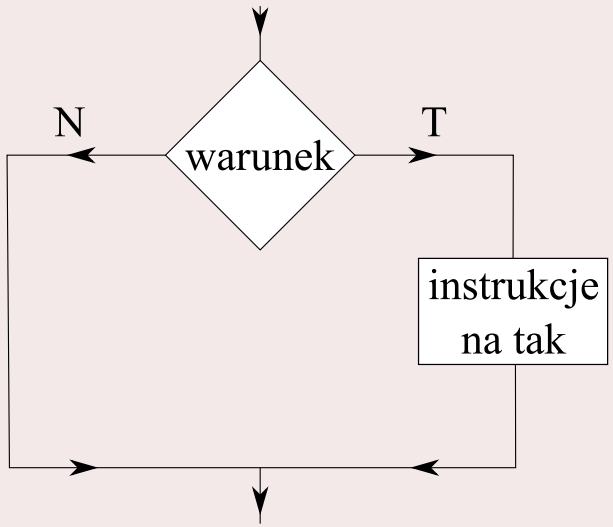


Instrukcje

3. **instrukcja warunkowa `if`:**

Zapis skrócony:

```
if (warunek)
{
  lista instrukcji na tak
}
```



Instrukcje

3. **instrukcja warunkowa `if`:**

przykłady:

```
if(a>0)
   c=b+2;
else
   c=b-3;

if(a<0)
   a=-a;
```

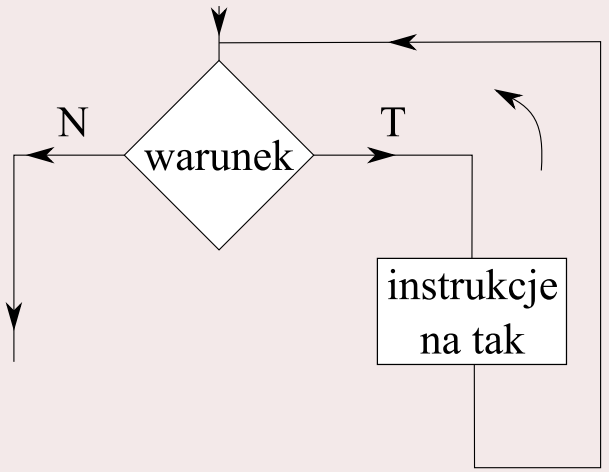
```
if(delta>0)
{
  x1=(-b-sqrt(delta))/(2*a);
  x2=(-b+sqrt(delta))/(2*a);
}
else if(delta==0)
        x1=x2=-b/(2*a);
     else
        printf("Brak rozwiazania");
```

Instrukcje

4. **instrukcja pętli `while`:**

Zapis:

```
while (warunek)
{
  lista instrukcji na tak
}
```



Instrukcje

4. **instrukcja pętli `while`:**

przykłady:

```
k=1;
while(k<=20)
{
  putchar('-');
  k++;
}
```

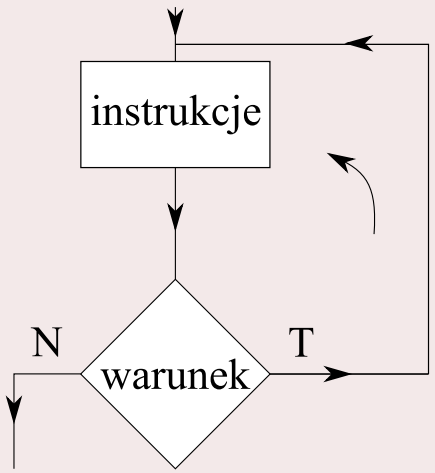
```
i=0;
while((linia[i]=getchar())!='\n')
  i++;
```

Instrukcje

5. **instrukcja pętli `do-while`:**

   Zapis:

```
do
{
  lista instrukcji
}
while (warunek);
```



Instrukcje

5. **instrukcja pętli `do-while`:**

przykłady:

```
do
{
  printf("\nn=");
  scanf("%d",&n);
}
while(n<=0);
```

```
i=1;j=64;
do
  printf("\n%d %d %c",i++,j+i,j+i);
while(i<=6);
```

Wynik:

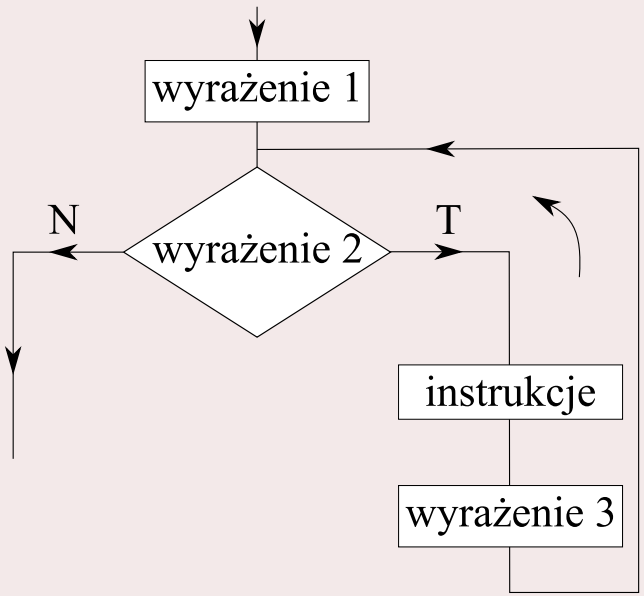
```
1   65   A
2   66   B
     :
6   70   F
```

Instrukcje

6. **instrukcja pętli `for`:**

   Zapis:

```
for(wyr1;wyr2;wyr3)
{
  lista instrukcji
}
```



Instrukcje

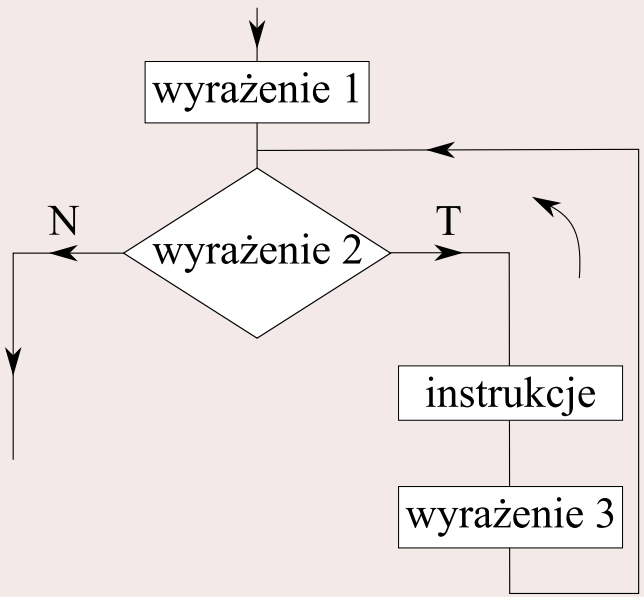
6. **instrukcja pętli `for`:**

Zapis:

```
for(wyr1;wyr2;wyr3)
{
  lista instrukcji
}
```

Odpowiednik pętla `while`:

```
wyr1;
while(wyr2)
{
  lista instrukcji
  wyr3;
}
```



Instrukcje

6. **instrukcja pętli `for`:**

   przykłady:

```
s=0;
for(i=1;i<=n;i++)
{
  printf("\n[%d]=",i);
  scanf("%d",&a);
  s+=a;
}

for(i=0,j=n-1;i<n;i++,j--)
  s+=a[i][i]+a[i][j];
```

Instrukcje

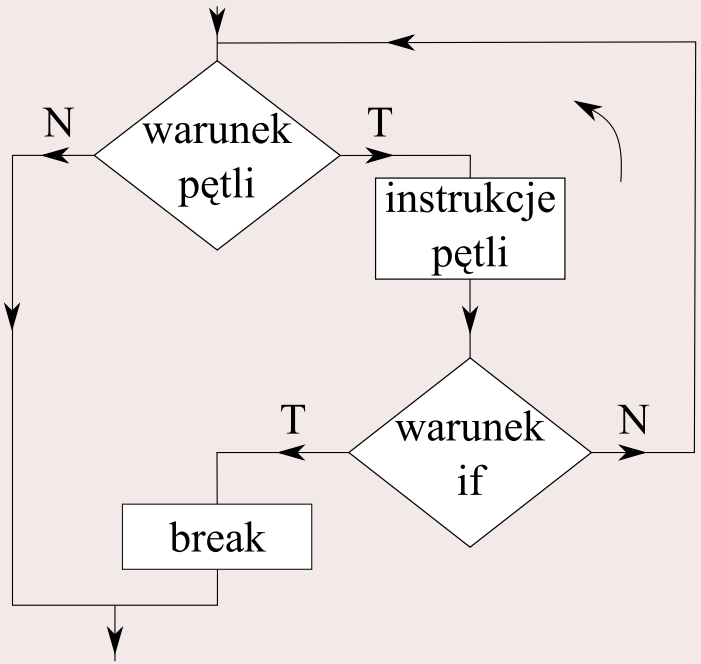
7. **instrukcja zaniechania `break`:**

Zapis:

```
break;
```

przykład:

```
s=0;
while(1)
{
  printf("a=");
  scanf("%d",&a);
  if(a<=0)
    break;
  s+=a;
}
```



Instrukcje

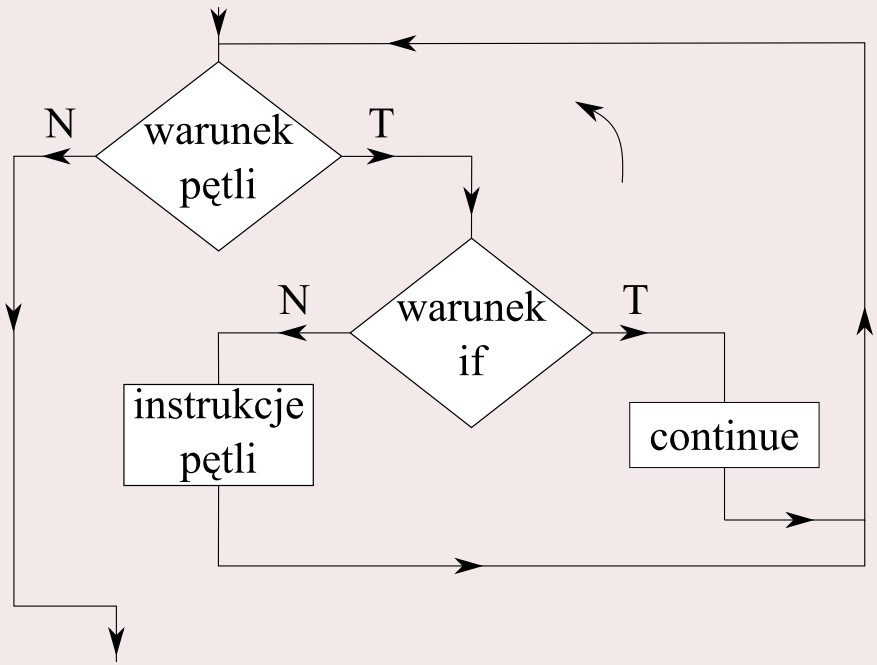
8. **instrukcja kontynuowania `continue`:**

   Zapis:

   ```
   continue;
   ```

   przykład:

   ```
   for(s=0,i=1;i<=n;i++)
   {
     if(i%2)
       continue;
     s+=i;
   }
   ```

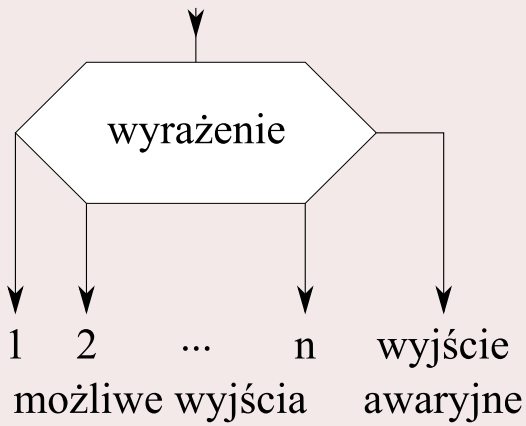


Instrukcje

9. **instrukcja wyboru `switch`:**

Zapis:

```
switch(wyrażenie)
{
  case wyrażenie stałe 1:
       lista instrukcji 1
  case wyrażenie stałe 2:
       lista instrukcji 2
       ...
  case wyrażenie stałe n:
       lista instrukcji n
  default:
       awaryjna lista instrukcji
}
```



Instrukcje

9. **instrukcja wyboru `switch`:**

przykład:

```
switch(wybor)
{
  case 1:
  case 2: printf("Brawo\n");
          break;
  case 3: printf("Tak trzymac\n");
  case 4: printf("Bardzo dobrze\n");
          break;
  default: printf("Milego dnia\n");
}
```

Instrukcje

10. **instrukcja pusta**:

    Zapis:

    ```
    ;
    ```

    Poprawia czytelność programu, nie wykonuje żadnego działania, należy stosować z ostrożnością.

    Przykład błędnego zastosowania:
    `while(1);` – pętla nieskończona

    `for(i=1;i<n;i++);` – pętla zwiększająca tylko i od 1 do n

    `if(n<0);` – wykonuje się tylko instrukcja pusta

Funkcje

Definicja funkcji:

```
typ danych nazwa funkcji(lista parametrów formalnych)
{
  lista zmiennych lokalnych
  lista instrukcji
}
```

typ danych – należy do zbioru typów dozwolonych w języku C
nazwa funkcji – typowa nazwa dla języka C
lista parametrów formalnych – określa zestaw par w postaci: **<typ parametru␣nazwa parametru formalnego>** oddzielonych przecinkami
Jeżeli funkcja nie zwraca wartości to **typ danych** jest `void`.

Funkcje

Przykłady:

```
void podkreslenie (char znak, int n)
{
  int i;
  printf("\n");
  for(i=1;i<=n;i++)
    putchar(znak);
  printf("\n");
}
```

Funkcje

Przykłady:

```
int maks (int n, int t[])
{
  int i, m=t[0];
  for(i=1;i<n;i++)
    if(t[i]>m)
      m=t[i];
  return m;
}
```

Funkcje

Deklaracja funkcji (prototyp):

Jeżeli definicja funkcji występuje później niż wywołanie niezbędna jest deklaracja funkcji.
Zapis:

```
typ danych nazwa funkcji(lista typów parametrów);
```

Przykład:

```
int maks(int n, int t[]);
```

lub

```
int maks(int, int[]);
```

Funkcje

Wywołanie funkcji

Zapis:

```
nazwa funkcji(lista parametrów aktualnych)
```

`lista parametrów aktualnych` – wyrażenia oddzielone przecinkami

- liczba parametrów aktualnych = liczba parametrów formalnych
- typ parametru aktualnego $\leqslant$ typ parametru formalnego
- kolejność parametrów aktualnych zgodna z kolejnością parametrów formalnych

Funkcje

Funkcja w języku C może:

- zwrócić dokładnie 1 wartość
- nie zwrócić żadnej wartości

W języku C parametry do funkcji przekazywane są przez:

- wartość

  Funkcja wywoływana otrzymuje kopie tych parametrów i działa na nich jak na zmiennych lokalnych zainicjowanych wartościami początkowymi.

- wskaźnik

  Funkcja otrzymuje adres parametru i może dokonywać zmian pod adresem.

Funkcje

Przykłady

- Wywołanie funkcji w instrukcji samodzielnej:

```
char znak = '*';
podkreslenie(znak, 10);
```

Wynik:

```
**********
```

- Wywołanie funkcji w wyrażeniu:

```
int tab[] = {5,8,-3,0,1,2,4,5};
printf("\n maksimum=%d",maks(8,tab));
```

Wynik:

```
maksimum=8
```

Funkcje

Rekurencja – funkcja wywołuje samą siebie

Przykład:
definicja matematyczna silni:

$$n! = \begin{cases} 1 & n = 0 \\ n * (n-1)! & n \geqslant 1 \end{cases}$$

zapis w języku C:

```
long int silnia(int n)
{
  if(n<=1)
    return 1;
  else
    return (n*silnia(n-1));
}
```

Funkcje

Funkcje biblioteczne

Przykłady:

Biblioteka <math.h>

`int abs(int i)`	$\lvert \mathbf{i} \rvert$ z `int`
`double fabs(double x)`	$\lvert \mathbf{x} \rvert$ z `double`
`double log(double x)`	Logarytm naturalny
`double log10(double x)`	Logarytm dziesiętny
`double pow(double x,double y)`	Potęga $\mathbf{x}^{\mathbf{y}}$
`double sqrt(double x)`	Pierwiastek $\sqrt{\mathbf{x}}$

Biblioteka <stdlib.h>

```
int system(const char* polecenie)
```

wywołanie komendy systemowej

Funkcje

Funkcje biblioteczne

Przykłady:

Biblioteka `<string.h>`

```
char* strcpy(char* s1,char* s2)
```
kopiuje łańcuch s2 do s1

```
char* strcat(char* s1,char* s2)
```
dołącza łańcuch s2 do końca s1

```
int strcmp(char* s1,char* s2)
```
porównuje dwa łańcuchy, zwraca 0 gdy są identyczne

```
unsigned int strlen(char* s)
```
zwraca długość łańcucha bez znaku NULL

Funkcje

Funkcje biblioteczne

Przykłady:

Deklaracje stałych moduł <math.h>

```
#define M_PI        3.14159265358979323846
#define M_PI_2      1.57079632679489661923
#define M_PI_4      0.78539816339744830916
#define M_1_PI      0.31830988618379067153
```

Funkcje

Funkcje biblioteczne

Włączenie zawartości plików:

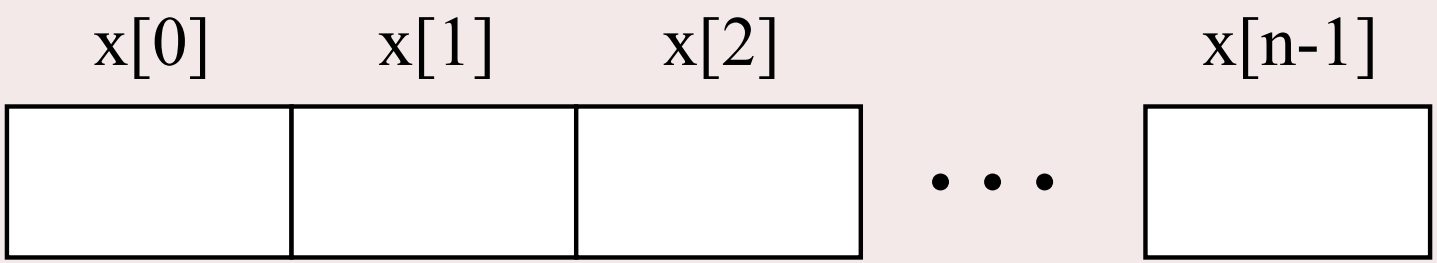
- `#include <nazwa_pliku>`
  Przeglądanie katalogu zbiorów nagłówkowych.
- `#include "nazwa_pliku"`
  Przeglądanie katalogu bieżącego, a potem przeglądanie katalogu zbiorów nagłówkowych.

Tablice

Definicja tablicy:

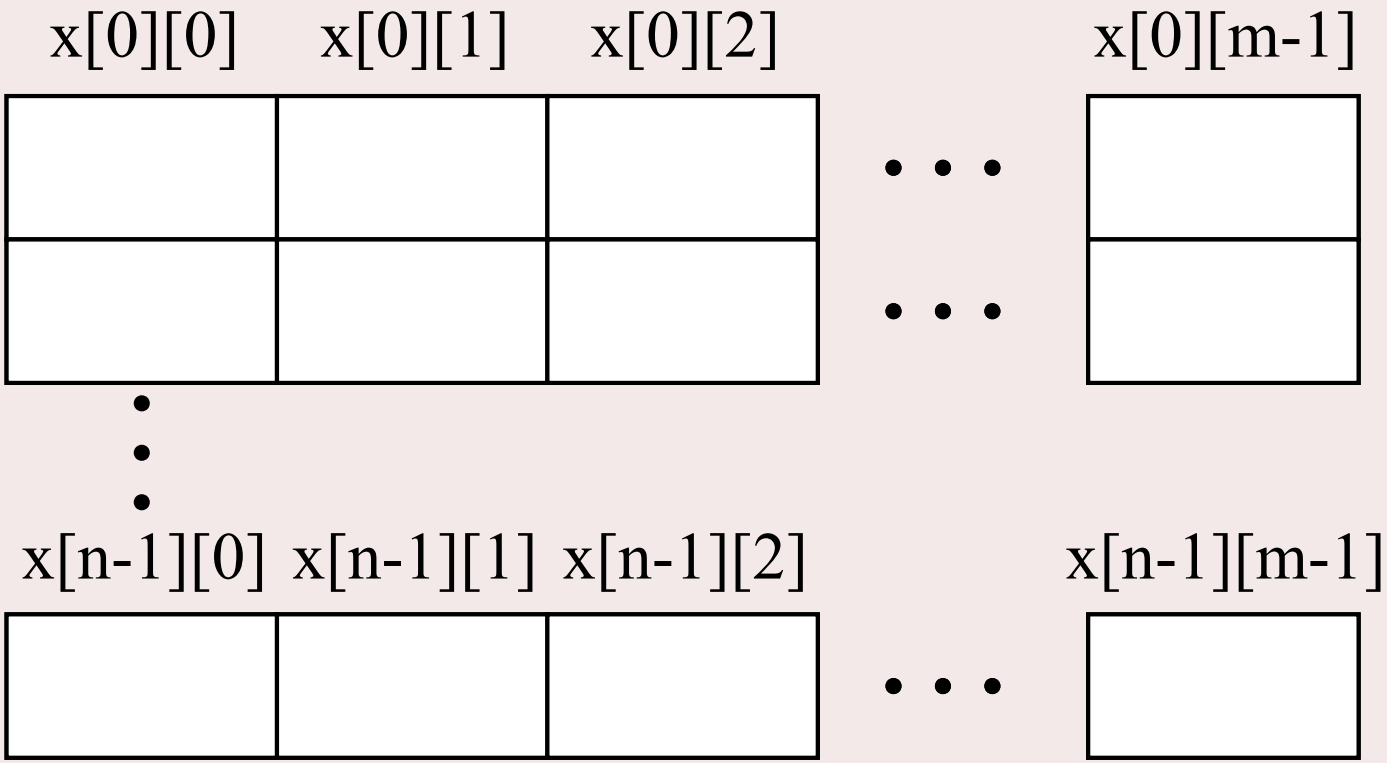
Zbiór elementów tego samego typu składowanych w spójnym obszarze pamięci operacyjnej.
Tablice mogą być jedno lub wielowymiarowe.

Tablica jednowymiarowa n-elementowa.



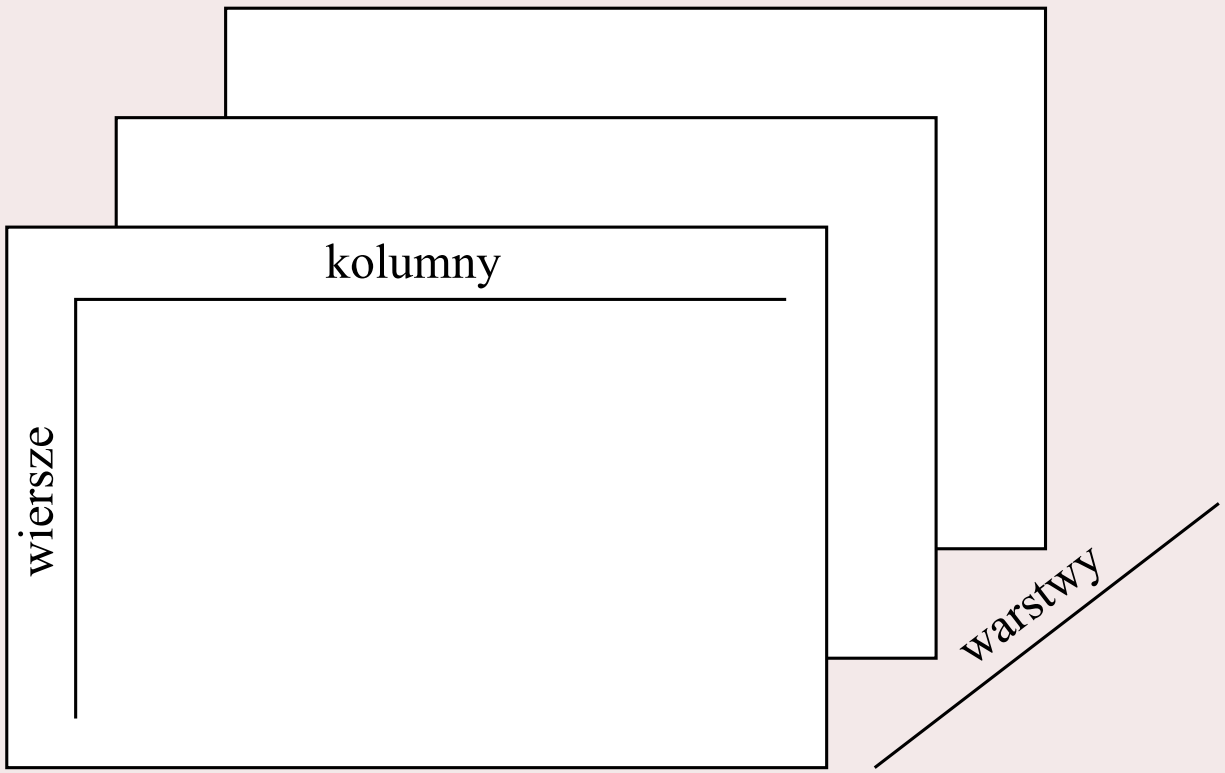
Tablice

Tablica dwuwymiarowa n*m-elementowa.



Tablice

Tablica trójwymiarowa.



Tablice

Deklaracja i inicjowanie tablicy:

`typ danych nazwa tablicy[lista rozmiarów]={lista wartości};`

Przykłady:

```
int tab[10];
double x[10] = {1.2,0.,-3.7};
int a[]={0,1,2,3,4};
char color[]={'b','i','a','l','y'};
char barwa[]="szary";
int b[2][3]={{100,101,102},{200,201,202}};
int b[2][3]={100,101,102,200,201,202};
int b[2][3]={{100},{200,201}};
```

Wskaźniki

Definicja wskaźnika:

Wskaźnik jest zmienną, której wartością jest adres innej zmiennej.
2 operatory dotyczące wskaźników:

- operator adresu (&)
- operator adresowania pośredniego (*), zwany również operatorem wyłuskania lub dereferencji

Wskaźniki

Deklaracja wskaźnika:

typ danych * nazwa zmiennej wskaźnikowej

Przykłady:

Deklaracja	Komentarz
`int *p;`	p jest wskaźnikiem do obiektu typu `int`
`int **p;`	p jest wskaźnikiem do wskaźnika typu `int`
`int *tab[8];`	tab jest 8-io elementową tablicą wskaźników typu `int`
`char (*p)[8];`	p jest wskaźnikiem do 8-io elementowej tablicy typu `char`
`char *f(void);`	f jest bezparametrową funkcją zwracającą wskaźnik do obiektu typu `char`

Wskaźniki

Deklaracja wskaźnika:

typ danych * nazwa zmiennej wskaźnikowej

Przykłady:

Deklaracja	**Komentarz**
`int (*f)(void);`	`f` jest wskaźnikiem do bezparametrowej funkcji zwracającej wartość typu `int`
`float v;`	deklaracja zmiennej `v` typu `float`
`float *pv=&v;`	deklaracja wskaźnika `pv` i zainicjowanie adresem zmiennej `v`
`int *pu=NULL;`	wyzerowanie wskaźnika

Wskaźniki

Operacje na wskaźnikach:

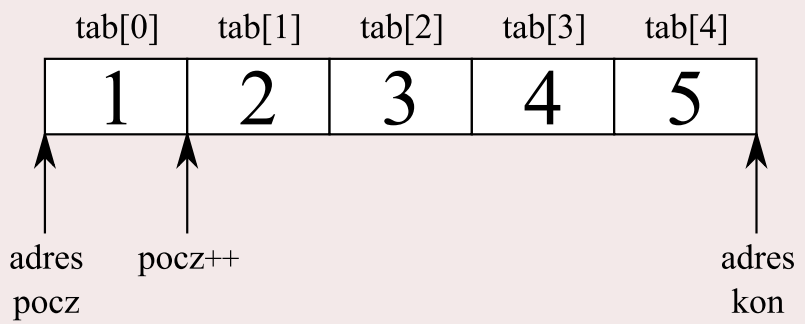
```
int u=13,v,*pu;
pu=&u;   // przypisanie wskaźnikowi pu adresu zmiennej u
v=*pu;   // przypisanie zmiennej v zawartości pod adresem pu

int a[10], x[10][20];
```

Notacja	
indeksowa	**wskaźnikowa**
`a[0]`	`*a`
`a[i]`	`*(a+i)`
`&a[0]`	`a`
`&a[i]`	`a+i`
`x[i][j]`	`*(x[i]+j)`
`x[i][j]`	`*(*(x+i)+j)`

Wskaźniki

Operacje na wskaźnikach:

```
int tab[5]={1,2,3,4,5};
int *pocz, *kon;
int s=0;
pocz=tab;
kon=tab+5;
while(pocz<kon)
{
   s+=*pocz;
   pocz++;
}
```



Wskaźniki

Wskaźniki i funkcje:

```
#include <stdio.h>
void zeruj1(int u, int v);
void zeruj2(int *pu, int *pv);
void main()
{
   int x=13, y=33;
   printf("\nPrzed wywolaniem zeruj1:x=%d y=%d",x,y);
   zeruj1(x,y);
   printf("\nPo wywolaniu zeruj1:x=%d y=%d",x,y);
   zeruj2(&x,&y);
   printf("\nPo wywolaniu zeruj2:x=%d y=%d",x,y);
}
```

Wskaźniki

Wskaźniki i funkcje:

```
void zeruj1(int u, int v)
{
   u=0; v=0;
}
void zeruj2(int *pu, int *pv)
{
   *pu=0; *pv=0;
}
```

Wynik:

```
Przed wywolaniem zeruj1:x=13 y=33
Po wywolaniu zeruj1:x=13 y=33
Po wywolaniu zeruj2:x=0 y=0
```

Wskaźniki

Wskaźniki i funkcje:

```
int analiza(char linia[],int *l,int *c,int *o,int *i)
{
   char znak; int j=0;
   while((znak=toupper(linia[j]))!='\0')
   {
      if(znak>='A'&&znak<='Z') ++*l;
      else if(znak>='0'&&znak<='9') ++*c;
         else if(znak==' '||znak=='\t') ++*o;
            else ++*i;
      ++j;
   }
   return j;
}
```

Wskaźniki

Dynamiczna rezerwacja pamięci

Funkcje z modułu **stdlib.h**

`void* malloc(size_t n)`
zwraca wskaźnik do *n* bajtów niezainicjowanej pamięci lub `NULL`
`void* calloc(size_t n,size_t size)`
zwraca wskaźnik do obszaru mogącego pomieścić tablicę *n* elementów podanego rozmiaru *size*
`void* realloc(void *ptr,size_t size)`
zmienia rozmiar bloku wskazanego przez *ptr* na *size*
`void free(void *ptr)`
zwalnia obszar pamięci wskazany przez *ptr*,
ptr powinien być wartością zwróconą wcześniej przez funkcje: calloc, malloc lub realloc

Wskaźniki

Dynamiczna rezerwacja pamięci

Przykłady:

```
int *x;
x=(int *)malloc(10*sizeof(int));
```

lub

```
x=(int *)calloc(10,sizeof(int));
...
x=(int *)realloc(x,20*sizeof(int));
...
free(x);
```

Wskaźniki

Dynamiczna rezerwacja pamięci

```
#include <stdio.h>
#include <stdlib.h>
#define WIER 8
void main()
{
   int *x[WIER]; //jednowymiarowa tablica wskazników
   int w, k; //indeksy: wiersz,kolumna
   /*przydzielenie pamieci dla tablicy trojkatnej
   liczba elementow w wierszu = numer wiersza+1*/
   for(w=0;w<=WIER-1;w++)
      x[w]=(int*)malloc((w+1)*sizeof(int));
```

Wskaźniki

Dynamiczna rezerwacja pamięci

```
//wypelnienie tablicy liczbami
for(w=0;w<=WIER-1;w++)
   for(k=0;k<=w;k++)
      *(x[w]+k)=100+w+k;
//wydruk tablicy
for(w=0;w<=WIER-1;w++)
{
   printf("\n");
   for(k=0;k<=w;k++)
      printf("%7d",*(x[w]+k));
}
```

Wskaźniki

Dynamiczna rezerwacja pamięci

```
   //zwolnienie pamięci
   for(w=0;w<=WIER-1;w++)
      free(x[w]);
}
```

Wynik:

```
100
101  102
102  103  104
103  104  105  106
104  105  106  107  108
105  106  107  108  109  110
106  107  108  109  110  111  112
107  108  109  110  111  112  113  114
```

Struktury

Definicja struktury:

Struktura składa się z elementów najczęściej różnego typu zwanych składnikami lub polami struktury. Elementy te zgrupowane są pod jedną nazwą.
2 operatory dotyczące struktur:

- operator składowej (.)
- operator wskaźnikowy składowej (->)

```
struct nazewnik struktury
{
zbiór składowych struktury
};
```

Struktury

Definicja struktury:

```
struct nazewnik struktury
{
zbiór składowych struktury
} lista zmiennych;
```

Przykład:

```
struct ksiazka
{
char tytul[40];
char autor[30];
float wartosc;
int numer;
};
```

Struktury

Definicja struktury przy pomocy typedef:

```
typedef struct
{
zbiór składowych struktury
} NAZWA SZABLONU;
```

Przykład:

```
typedef struct
{
char tytul[40];
char autor[30];
float wartosc;
int numer;
} KSIAZKA;
```

Struktury

Deklaracja zmiennych strukturalnych:

Przykłady:

```
struct ksiazka bib, tab[5], *wk;
struct ksiazka k={"W pustyni i w puszczy","Henryk
Sienkiewicz", 100, 1};
struct ksiazka tablica[5]={{"Potop"},{"Ogniem i
mieczem"},{"Pan Wolodyjowski"}};

KSIAZKA bib, tab[5], *wk;
KSIAZKA k={"W pustyni i w puszczy","Henryk
Sienkiewicz", 100, 1};
```

Struktury

Dostęp do elementów struktury:

- zmienna jest strukturą (.)
  nazwa zmiennej.nazwa elementu struktury
  Przykład:
  ```
  gets(bib.tytul);
  if (bib.tytul[0]=='$') break;
  ```
- zmienna jest wskaźnikiem do struktury (->)
  nazwa zmiennej wskaźnikowej->nazwa elementu struktury
  Przykład:
  ```
  if (wk->numer<0) break;
  if ((*wk).numer<0) break;
  ```

Pliki

Definicja pliku:

Plik jest to ciąg danych (zestaw liczb, linii tekstów, struktur) tego samego typu składowanych w pamięci zewnętrznej komputera.

Predefiniowany typ danych o nazwie `FILE`.

Definicja pliku:

```
FILE *plik;
```

powoduje zarezerwowanie przez kompilator bufora roboczego w pamięci operacyjnej komputera oraz wykreowanie wskaźnika `plik` wskazującego początek tego bufora.

Pliki

Funkcje umożliwiające obsługę pliku:

- otwarcie pliku

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

  `path` – nazwa pliku

  `mode` – parametr określający specyfikę współpracy z podanym plikiem

  Wartości parametru `mode`:

  `r` – istniejący plik otwarty do czytania

  `w` – nowy plik otwarty do pisania

  `a` – istniejący plik otwarty do dopisywania

  `r+` – istniejący plik otwarty do czytania i pisania

  `w+` – nowy plik otwarty do pisania i czytania

  `a+` – istniejący plik otwarty do dopisywania i czytania

Pliki

Funkcje umożliwiające obsługę pliku:

- otwarcie pliku
  ```
  FILE* fopen(const char *path, const char *mode);
  ```

Funkcja `fopen()` zwraca wskaźnik plikowy, który służy jako identyfikator pliku dla innych funkcji we/wy lub `NULL`, gdy nie jest w stanie otworzyć pliku.

Powody błędu:
– zła nazwa pliku,
– brak miejsca na dysku,
– dostęp do pliku jest niedozwolony,
– problem sprzętowy.

Pliki

Funkcje umożliwiające obsługę pliku:

- zamknięcie pliku
  `int fclose(FILE *plik);` lub `int fcloseall(void);`

  Funkcje `fclose()` i `fcloseall()` zwracają 0 jeżeli akcja pomyślnie zakończona lub EOF w przeciwnym wypadku.

  Powody błędu:
  – brak miejsca na dysku,
  – pamięć zewnętrzna została odłączona,
  – wystąpił błąd we/wy.

  Powody stosowania:
  – opróżnia bufor – dane nie są tracone,
  – wstawia znacznik końca pliku,
  – zwolniony wskaźnik można przypisać innemu plikowi.

Pliki

Funkcje umożliwiające obsługę pliku:

Przykład:

Układ programu z obsługą plików

```
#include<stdio.h> //dolaczenie biblioteki systemowej
...
FILE *plik; //deklaracja zmiennej plikowej
...
plik=fopen("C:\\temp.txt", "r+"); //otwarcie pliku
...
fclose(plik); //zamkniecie pliku
...
```

Pliki

Funkcje wprowadzania i wyprowadzania danych:

- zapisanie znaku do pliku
  ```
  int fputc(int c, FILE *plik);
  ```
  Wysyła c do pliku, zwraca c lub EOF
- odczytuje znak z pliku
  ```
  int fgetc(FILE *plik);
  ```
  Odczytuje znak z pliku, zwraca wartość znaku lub EOF.

Pliki

Funkcje wprowadzania i wyprowadzania danych:
Przykład:

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
  FILE *plik_we;
  char nazwa[80];
  int znak;
  printf("\n Wyswietlanie zawartosci pliku");
  printf("\n Podaj nazwe pliku");
  gets(nazwa);
  plik_we=fopen(nazwa,"r+");
```

Pliki

Funkcje wprowadzania i wyprowadzania danych:

Przykład:

```
if(plik_we==NULL)
{
  printf("\n Plik %s nie istnieje",nazwa);
  exit(1); //zamkniecie wszystkich plikow i wyjscie
}
else
{
  while((znak=fgetc(plik_we))!=EOF)
    fputc(znak,stdout);
  fclose(plik_we);
}
}
```

Pliki

Nazwy plików standardowych:

– `stdin` – standardowy plik wejściowy (klawiatura)
– `stdout` – standardowy plik wyjściowy (monitor)
– `stderr` – standardowy plik błędów
– `stdprn` – standardowy plik drukarki

Pliki

Funkcje wprowadzania i wyprowadzania danych:

- zapisanie łańcucha znaków do pliku
  `int fputs(const char *s, FILE *plik);`
  Wysyła łańcuch `s` do pliku, zwraca EOF w przypadku błędu.
- odczytuje łańcuch znaków z pliku
  `char *fgets(char *s, int n, FILE *plik);`
  Odczytuje znaki z pliku i umieszcza w łańcuchu `s`. Odczyt zostaje przerwany po odczytaniu `n-1` znaków lub znaku końca wiersza. Umieszcza na końcu łańcucha `s` znak końca łańcucha. Zwraca wczytany łańcuch lub NULL.
  Jeżeli wczytana liczba znaków jest $<$`n-1` wstawia przed końcem łańcucha znak nowej linii.

Pliki

Funkcje wprowadzania i wyprowadzania danych:
Przykład:

```
FILE *plik
char lancuch[80];
...
while(fgets(lancuch,20,plik)!=NULL)
  printf(lancuch);
...
puts("Podaj lancuch");
gets(lancuch);
fputs(lancuch,plik);
fputs("\n",plik);
```

Pliki

Sformatowane wejście – wyjście:

- sformatowany zapis do pliku
  `int fprintf(FILE *plik, format, ...);`
  Zwraca wartość `int` równą liczbie wyprowadzanych znaków lub EOF w przypadku błędu.
  `format` – format zapisywanych wartości
  `...` – lista zapisywanych zmiennych
- sformatowany odczyt z pliku
  `int fscanf(FILE *plik, format, ...);`
  Zwraca wartość `int` równą liczbie prawidłowo odczytanych danych lub EOF jeżeli koniec zbioru.
  format – format odczytywanych wartości
  ...– lista odczytywanych zmiennych

Pliki

Sformatowane wejście – wyjście:
Przykład:

```
int a=1, s=0, i=0;
while(a)
{
  scanf("%d",&a);
  fprintf(plik,"%d ",a);
}
do
{
  fscanf(plik,"%d",&a);
  s+=a; i++;
} while(a);
```

Pliki

Zapis i odczyt blokowy (pliki binarne):

- zapis blokami do pliku

```
int fwrite(void *wsk, int rozmiar_bloku,
int liczba_blokow, FILE *plik);
```

- odczyt blokami z pliku

```
int fread(void *wsk, int rozmiar_bloku,
int liczba_blokow, FILE *plik);
```

Funkcje **fwrite** i **fread** zwracają liczbę pomyślnie zapisanych bloków.

`wsk` – wskaźnik początku obszaru pamięci operacyjnej
`rozmiar_bloku` – wielkość bloku w bajtach
`liczba_blokow` – ilość zapisywanych/odczytywanych bloków

Pliki

Zapis i odczyt blokowy:
Przykład:

```
double tab[10];
int i, n;
fwrite(tab, sizeof(double), 10, plik);
...
for(i=0;i<n;i++)
  fread(tab+i, sizeof(double), 1, plik);
```

Pliki

Funkcje do swobodnego przetwarzania plików:

- funkcja **rewind** – ustawia wskaźnik pliku na pozycję początkową
  ```
  void rewind(FILE *plik);
  ```
- funkcja **fseek** – ustawia wskaźnik pliku na wybraną pozycję
  ```
  int fseek(FILE *plik, long pozycja, int odniesienie);
  ```
  Funkcja zwraca 0 w przypadku prawidłowego działania.
  `pozycja` – określa liczbę bajtów od miejsca od miejsca określonego przez parametr `odniesienie`
  `odniesienie` – określa 3 możliwe położenia wskaźnika pliku:
  `SEEK_SET` – początek pliku
  `SEEK_CUR` – pozycja bieżąca
  `SEEK_END` – koniec pliku

Pliki

Funkcje do swobodnego przetwarzania plików:
Przykład:

Odczyt i-tej pozycji z pliku:

```
poz=(long)i*sizeof(double);
fseek(plik, poz, SEEK_SET);
fread(&wartosc, sizeof(double), 1, plik);
```

- funkcja `ftell` – wyznacza liczbę bajtów oddzielającą pozycję bieżącą wskaźnika pliku od jego początku
  `long ftell(FILE *plik);`

Przykład:

Wyznaczenie rozmiaru pliku:

```
fseek(plik,0,SEEK_END);
rozmiar=ftell(plik);
```

Pliki

Funkcje dodatkowe do przetwarzania plików:

- funkcja `feof` – bada koniec pliku

  ```
  int feof(FILE *plik);
  ```

  Zwraca 0 jeśli nie osiągnięto końca pliku

  Przykład:

  Przeglądanie zawartości pliku:

  ```
  while(!feof(plik))
  {
  ...
  }
  ```

Pliki

Funkcje dodatkowe do przetwarzania plików:

- funkcja `fflush` – wysyła zawartość buforu do pliku określonego przez wskaźnik (opróżnienie buforu)
  ```
  int fflush(FILE *plik);
  ```
- funkcja `fflushall` – wysyła zawartość wszystkich buforu otwartych strumieni wyjściowych do związanych z nimi plików oraz kasuje zawartość buforów wejściowych
  ```
  int fflushall(void);
  ```

Pliki

Funkcje dodatkowe do przetwarzania plików:

- funkcja **unlink** – usuniecie pliku o określonej nazwie
  ```
  int unlink(char *nazwa_pliku);
  ```
- funkcja **remove** – usuniecie pliku o określonej nazwie
  ```
  int remove(char *nazwa_pliku);
  ```
- funkcja **rename** – zmienia nazwę pliku
  ```
  int rename(char *stara_nazwa,char *nowa_nazwa);
  ```
  Funkcje **unlink**, **remove**, **rename** zwracają 0 przy poprawnym wykonaniu, -1 jeśli wystąpił błąd.