Politechnika Wrocławska

# Wstęp do programowania

## INP001213Wcl

rok akademicki 2018/19

semestr zimowy

# Wykład 11

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

A-1 p. 411B

Politechnika Wrocławska

# Plan prezentacji (1)

- Obsługa łańcuchów znakowych
  - `getchar()`, `putchar()`
  - `ctype`, `string`
  - `puts()`, `gets()`

Na podstawie:

- G. Perry, D. Miller, *Język C. Programowanie dla początkujących*

Politechnika Wrocławska

# Plan prezentacji (2)

- Wskaźnik na plik
- Dostęp do plików
  - Odczyt plików tekstowych
  - Zapis plików tekstowych
- Formatowanie danych wyjściowych
- Formatowane dane wejściowe
- Argumenty linii poleceń

Na podstawie:

- G. Perry, D. Miller, *Język C. Programowanie dla początkujących*

Politechnika Wrocławska

# Funkcje `getchar()`, `putchar()`

- Funkcja **`getchar()`** pobiera pojedynczy znak
- Funkcja **`putchar()`** drukuje pojedynczy znak

# Funkcje getchar(), putchar()

Start here | 01_putchar.c

```c
/*01_putchar.c*/
/*Wstep do programowania*/
/*na podstawie: G. Perry, D. Miller,
Język C Programowanie dla poczatkujacych, Helion, 2014*/
/*Program ilustruje wykorzystanie funkcji
putchar() do wypisania napisu*/

#include <stdio.h>
#include <string.h>

main(){
    int i;
    char msg[] = "Programowanie w C jest fajne!";
    for(i=0; i < strlen(msg); i++){
        putchar(msg[i]);
    }
    putchar('\n');
    return 0;
}
```

# Funkcje getchar(), putchar()

Start here | 02_getchar_putchar.c

```c
/*02_getchar_putchar.c*/
/*Wstep do programowania*/
/*na podstawie: G. Perry, D. Miller,
Jezyk C Programowanie dla poczatkujacych, Helion, 2014*/
/*Program ilustruje wykorzystanie funkcji
getchar() i putchar()*/

#include <stdio.h>
#define N 25

main(){
    int i;
    char msg[N];

    printf("Wpisz maksymalnie 25 znakow i nacisnij klawisz Enter...\n");

```

# Funkcje getchar(), putchar()

Start here | 02_getchar_putchar.c

```
16
17      for(i = 0; i < N; i++){
18          msg[i] = getchar(); //pobranie pojedynczego znaku
19          if(msg[i] == '\n'){ //jesli pobrany znak jest znakiem nowej linii
20              break;          //zakoncz wczytywanie
21          }
22      }
23      i--;
24
25      putchar('\n');          //dodatkowe przelamanie linii
26
27      //wypisanie napisu wspak
28      for( ; i >= 0; i--){    //poczatkowa wartosc zmiennej i
29          putchar(msg[i]);    // zostala ustalona w poprzedniej petli
30      }
31
32      putchar('\n');
33
34      return 0;
35  }
```

Politechnika Wrocławska

# Funkcje `getchar()`, `putchar()`

- funkcja **`getchar()`** buforuje dane wejściowe
- dopóki użytkownik na naciśnie klawisza **`Enter`** dane może poprawić dane
- znak nowej linii pozostaje w buforze, o ile nie zostanie usunięty

# Funkcje getchar(), putchar()

Start here | 03_getchar_bufor.c

```c
/*03_getchar_bufor.c*/
/*Wstep do programowania*/
/*na podstawie: G. Perry, D. Miller,
Jezyk C Programowanie dla poczatkujacych, Helion, 2014*/
/*Program ilustruje problem ze znakiem
nowego wiersza pozostajacym w buforze*/

#include <stdio.h>

main(){
    char inicjalImienia, inicjalNazwiska;

    printf("Podaj inicjal imienia.\n");
    inicjalImienia = getchar();

    printf("Podaj inicjal nazwiska.\n");
    inicjalNazwiska = getchar();

    printf("Twoje inicjaly to %c. %c.\n",inicjalImienia, inicjalNazwiska);

    return 0;
}
```

Program nie zadziała zgodnie z oczekiwaniem, bo znak nowej linii pozostanie w buforze.

# Funkcje getchar(), putchar()

Start here | 04_getchar_bufor.c

```c
/*04_getchar_bufor.c*/
/*Wstep do programowania*/
/*na podstawie: G. Perry, D. Miller,
Jezyk C Programowanie dla poczatkujacych, Helion, 2014*/
/*Program ilustruje problem ze znakiem
nowego wiersza pozostajacym w buforze*/

#include <stdio.h>

main(){
    char inicjalImienia, inicjalNazwiska;

    printf("Podaj inicjal imienia.\n");
    inicjalImienia = getchar();
    getchar();

    printf("Podaj inicjal nazwiska.\n");
    inicjalNazwiska = getchar();
    getchar();

    printf("Twoje inicjaly to %c. %c.\n",inicjalImienia, inicjalNazwiska);
    return 0;
}
```

Rozwiązaniem problemu jest usunięcie znaku nowej linii z bufora.

# Funkcje getchar(), putchar()

Start here | 05_getchar_bufor.c

```c
/*05_getchar_bufor.c*/
/*Wstep do programowania*/
/*na podstawie: G. Perry, D. Miller,
Jezyk C Programowanie dla poczatkujacych, Helion, 2014*/
/*Program ilustruje problem ze znakiem
nowego wiersza pozostajacym w buforze*/

#include <stdio.h>

main(){
    char inicjalImienia, inicjalNazwiska;

    printf("Podaj swoje inicjaly.\n");
    inicjalImienia = getchar();
    inicjalNazwiska = getchar();
    getchar();

    printf("Twoje inicjaly to %c. %c.\n",inicjalImienia, inicjalNazwiska);
    return 0;
}
```

Program prosi o podanie inicjałów razem.
Do usunięcia pozostanie jeden znak nowej linii.

Politechnika Wrocławska

# Biblioteka `ctype`

- do sprawdzenia do jakiej klasy znaków należy wczytany znak można wykorzystać funkcje z biblioteki **`ctype`**

Politechnika Wrocławska

# Biblioteka ctype

Start here | 06_isdigit.c

```c
/*06_isdigit.c*/
/*Wstep do programowania*/
/*Program sprawdza,
czy wprowadzony znak jest cyfra*/

#include <stdio.h>

main(){
    char znak;
    printf("Wpisz znak: ");
    znak = getchar();

    if( znak >= '0' && znak <= '9')
       printf("Wpisany znak jest cyfra.\n");
    else{
       printf("Wpisany znak nie jest cyfra.\n");
    }

    return 0;
}
```

# Biblioteka ctype

Start here | 06_isdigit.c | 07_isdigit.c

```c
/*07_isdigit.c*/
/*Wstep do programowania*/
/*Program sprawdza,
czy wprowadzony znak jest cyfra*/

#include <stdio.h>
#include <ctype.h>

main(){
    char znak;
    printf("Wpisz znak: ");
    znak = getchar();

    if( isdigit(znak) )
        printf("Wpisany znak jest cyfra.\n");
    else{
        printf("Wpisany znak nie jest cyfra.\n");
    }

    return 0;
}
```

Politechnika Wrocławska

# Biblioteka `ctype`

- `isalnum(c)`
- `isalpha(c)`
- `isdigit(c)`
- `islower(c)`
- `isupper(c)`
- i inne ...

- `toupper(c)`
- `tolower(c)`

Politechnika Wrocławska

# Biblioteka `string`

- `strcpy()`
- `strlen()`
- `strcat()`
- i inne …

# Biblioteka string

Start here | 08_string.c

```
1   /*08_string.c*/
2   /*Wstep do programowania*/
3   /*Program demostruje funkcje
4   z biblioteki string*/
5
6   #include <stdio.h>
7   #include <string.h>
8
9   main(){
10      char napis1[14] = "Pro";
11      char napis2[]   = "gra";
12      char napis3[]   = "mowa";
13      char napis4[]   = "nie";
14      char napis5[5];
15
```

# Biblioteka string

Start here | 08_string.c

```
16      /*funkcja strcpy()
17      kopiujemy napis3 do napis5*/
18      strcpy(napis5, napis3);
19      printf(napis5);
20      printf("\n");
21
22      /*funkcja strlen()
23      zwraca dlugosc napisu
24      liczba znaków do znaku '\0'*/
25      printf("napis1: %s ",napis1);
26      printf("ma dlugosc %d\n",strlen(napis1));
27
```

Politechnika Wrocławska

# Biblioteka string

Start here | 08_string.c

```
28      /*funkcja strcat
29      dolacza (konkatenuje)
30      dwa lancuchy
31      (w pierwszym musi byc
32       dostatecznie duzo miejsca*/
33      strcat(napis1, napis2);
34      printf("napis1: %s ",napis1);
35      printf("ma dlugosc %d\n",strlen(napis1));
36
37      strcat(napis1, napis3);
38      strcat(napis1, napis4);
39      printf("napis1: %s ",napis1);
40      printf("ma dlugosc %d\n",strlen(napis1));
41
42      return 0;
43  }
```

# Biblioteka string

```
mowa
napis1: Pro ma dlugosc 3
napis1: Progra ma dlugosc 6
napis1: Programowanie ma dlugosc 13
```

Start here | 08_str

```
28      /*
29      do
30      dw
31      (w
32       dostatecznie duzo miejsca*/
33      strcat(napis1, napis2);
34      printf("napis1: %s ",napis1);
35      printf("ma dlugosc %d\n",strlen(napis1));
36
37      strcat(napis1, napis3);
38      strcat(napis1, napis4);
39      printf("napis1: %s ",napis1);
40      printf("ma dlugosc %d\n",strlen(napis1));
41
42      return 0;
43  }
```

Politechnika Wrocławska

# Funkcje puts() i gets()

Start here | 09_puts_gets.c

```
1  /*09_puts_gets.c*/
2  /*Wstep do programowania*/
3  /*Program demostruje funkcje
4  puts() i gets()*/
5
6  #include <stdio.h>
7
8  main(){
9      char napis[1000];
10
11     puts("Podaj napis.");
12
13     gets(napis);
14     puts(napis);
15
16     return 0;
17 }
18
```

Politechnika Wrocławska

# Wskaźnik do pliku

- Dostęp do plików w programie uzyskuje się z wykorzystaniem wskaźnika do pliku.
- W bibliotece **`stdio`** zdefiniowano strukturę **`FILE`** służącą do przechowywania informacji o plikach.

- Przykładowa deklaracja

```
FILE* fptr;
```

Politechnika Wrocławska

# Otwieranie pliku

- Do otwierania plików służy funkcja **`fopen()`** z bibliteki **`stdio`**.
- Funkcja **`fopen()`** pobiera dwa argumenty:
  - nazwę pliku,
  - tryb dostępu do pliku.
- Funkcja **`fopen()`** zwraca wskaźnik na plik.

# Otwieranie pliku - przykład

Start here | 01_otwieranie_pliku.c

```c
/*01_otwieranie_pliku.c*/
/*Wstep do programowania*/
#include <stdio.h>

int main()
{
    FILE* fptr;
    fptr = fopen("dane.txt","w");
    /*wlasciwa czesc programu*/
    fclose(fptr);
    return 0;
}

```

Politechnika Wrocławska

# Zamykanie pliku

- Do zamykania plików służy funkcja **`fclose()`** z bibliteki **`stdio`**.
- Funkcja **`fclose()`** pobiera jeden argument:
  - wskaźnik plik.
- Funkcja **`fclose()`** zwraca liczbę całkowitą.

Politechnika Wrocławska

# Otwieranie pliku

## Tryb dostępu

- Tryb dostępu do pliku określany jest łańcuchem znakowym.
- Podstawowe tryby dostępu określają litery:
  - odczyt `"r"`,
  - zapis `"w"`,
  - dołączanie `"a"`,
  - plik binarny `"b"`.
- Litery można ze sobą łączyć.

Politechnika Wrocławska

# Dostęp do pliku

## Błędy

- Jeżeli nie można otworzyć pliku funkcja **`fopen()`** zwraca wskaźnik zerowy **`NULL`**.
- Jeżeli operacja zamknięcia pliku się nie powiedzie funkcja **`fclose()`** zwraca **`EOF`**.

# Dostęp do pliku

## Przykład 1

Start here | 02_otwieranie_nieistniejacego_pliku.c

```c
/*02_otwieranie_nieistniejacego_pliku.c*/
/*Wstep do programowania*/
/*Dostep do pliku.
Otwarcie do odczytu
nieistniejacego pliku.*/
#include <stdio.h>

int main()
{
    FILE* fptr;
    fptr = fopen("danetxt","r");
    printf("%d",(int)fptr);
    /*wlasciwa czesc programu*/
    fclose(fptr);
    return 0;
}
```

# Dostęp do pliku

## Przykład 2

Start here | 03_otwieranie_istniejacego_pliku.c

```c
/*03_otwieranie_istniejacego_pliku.c*/
/*Wstep do programowania*/
/*Dostep do pliku.
Otwarcie do odczytu
istniejacego pliku.*/
#include <stdio.h>

int main()
{
    FILE* fptr;
    fptr = fopen("dane.txt","r");
    printf("%d",(int)fptr);
    /*wlasciwa czesc programu*/
    fclose(fptr);
    return 0;
}
```

# Dostęp do pliku

## Przykład 3

Start here | 04_zapis_do_pliku.c

```
1   /*04_zapis_do_pliku.c*/
2   /*Wstep do programowania*/
3   /*Zapis danych do pliku.*/
4   #include <stdio.h>
5
6   int main()
7   {
8       //deklaracja trzech zmiennych
9       double a, b, c;
10      //i wskaznika do pliku
11      FILE* fptr;
12
```

# Dostęp do pliku

## Przykład 3

Start here | 04_zapis_do_pliku.c

```
13      //wczytanie wartosci zmiennych
14      //ze standardowego wejscia
15      printf("Podaj a: ");
16      scanf("%lf",&a);
17      printf("Podaj b: ");
18      scanf("%lf",&b);
19      printf("Podaj c: ");
20      scanf("%lf",&c);
21
```

# Dostęp do pliku

## Przykład 3

Start here | 04_zapis_do_pliku.c

```
22      //otworzenie pliku do zapisu
23      fptr = fopen("abc.txt","w");
24      //zapis danych sformatowanych do pliku
25      //(tylko jesli plik zostal poprawnie otwarty)
26      if( fptr != NULL ){
27          fprintf(fptr,"a = %g\n",a);
28          fprintf(fptr,"b = %g\n",b);
29          fprintf(fptr,"c = %g\n",c);
30          fclose(fptr);
31      }
32      return 0;
33  }
```

Politechnika Wrocławska

# Zapis danych do pliku

- Do zapisywania danych do pliku można wykorzystać funkcję **`fprintf()`**.
- Funkcja **`fprintf()`** działa analogicznie do funkcji **`printf()`**.
- Pierwszy argument jest wskaźnikiem do pliku, kolejnymi łańcuch kontrolny i dane do wyświetlenia.
- Przykładowo

```
fprintf(fptr,"a = %g\n",a);
```

# Odczyt danych z pliku

- Do czytania danych z pliku można wykorzystać funkcję **`fscanf()`**.
- Funkcja **`fscanf()`** działa analogicznie do funkcji **`scanf()`**.
- Pierwszy argument jest wskaźnikiem do pliku, kolejnymi łańcuch kontrolny i referencje do zmiennych.
- Przykładowo

```
fscanf(fptr,"a = %lf\n",&a);
```

# Dostęp do pliku

## Przykład 4

Start here | 05_odczyt_danych_z_pliku.c

```
1   /*05_odczyt_danych_z_pliku.c*/
2   /*Wstep do programowania*/
3   /*Odczyt danych z pliku.*/
4   #include <stdio.h>
5
6   int main()
7   {
8       //deklaracja zmiennych
9       //i wskaznika do pliku
10      double a, b, c;
11      FILE* fptr;
12
13      //otworzenie pliku do odczytu
14      fptr = fopen("abc.txt","r");
```

# Dostęp do pliku

## Przykład 4

Start here | 05_odczyt_danych_z_pliku.c

```
16      //jesli otwarto plik
17      if( fptr != NULL ){
18          //wczytaj dane
19          fscanf(fptr,"a = %lf\n",&a);
20          fscanf(fptr,"b = %lf\n",&b);
21          fscanf(fptr,"c = %lf\n",&c);
22          fclose(fptr);
23      }
24
25      //wypisz dane na ekran
26      printf("a = %g\n",a);
27      printf("b = %g\n",b);
28      printf("c = %g\n",c);
29
30      return 0;
31  }
```

# Standardowe strumienie danych

- Biblioteka **`stdio`** zapewnia dostęp do trzech standardowych strumieni danych:
  - **`stdin`** (standardowe wejście - klawiatura),
  - **`stdout`** (standardowe wyjście - ekran),
  - **`stderr`** (standardowy strumień błędów - ekran).

# Standardowe strumienie danych

## Przykład

Start here | 06_strumienie.c

```
1   /*06_strumienie.c*/
2   /*Wstep do programowania*/
3   /*Wykorzystanie stdin/stdout.*/
4   #include <stdio.h>
5
6   int main()
7   {
8       double a, b;
9       FILE* fptr;
10
11      //otworzenie pliku do odczytu
12      fptr = fopen("abcd.txt","r");
```

# Standardowe strumienie danych

## Przykład

Start here | 06_strumienie.c

```
14      //jesli nie udalo sie otworzyc pliku
15      if( fptr == NULL )
16          fptr = stdin; //ustaw wskaznik na stdin
17
18      //wczytaj dane
19      fscanf(fptr,"a = %lf\n",&a);
20      fscanf(fptr,"b = %lf",&b);
21
22      //wypisz dane na ekran
23      fprintf(stdout,"a = %lf\n",a);
24      fprintf(stdout,"b = %lf\n",b);
25
26      fclose(fptr);
27
28      return 0;
29  }
```

# Formatowanie danych wyjściowych

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

Łańcuch kontrolny zawiera:

- zwykłe znaki,
- znaczniki konwersji:
  - znacznik zaczyna się znakiem %,
  - znacznik kończy się znakiem określającym typ danych.

# Formatowanie danych wyjściowych

Pomiędzy %, a znakiem określającym typ danych mogą znajdować się (kolejno):

- znak minus (wyrównanie do lewej),
- liczba określająca minimalną szerokość pola,
- kropka oddzielającą szerokość pola od precyzji,
- liczba określająca precyzję:
  - największą dopuszczalną liczbę znaków w ciągu,
  - liczbę cyfr po separatorze dziesiętnym dla liczb zmiennoprzecinkowych,
  - minimalną liczbę cyfr dla liczb całkowitych,
- **`h`** lub **`l`**, jeżeli liczba całkowita ma zostać wypisana jako **`short`** lub **`long`**.

# Formatowanie danych wyjściowych

- Szerokość lub precyzja mogą być określone znakiem `*`.
- Szerokość określa wtedy następny argument, który musi być liczbą `int`.
- Przykładowo:

```
printf("%.*s", max, s);
```

# Formatowanie danych wyjściowych

| znak | typ | wypisywany jako |
|---|---|---|
| `d, i` | `int` | liczba dziesiętna |
| `o` | `int` | liczba ósemkowa |
| `x, X` | `int` | liczba szesnastkowa |
| `u` | `int` | liczba dziesiętna bez znaku |
| `c` | `int` | pojedynczy znak |
| `s` | `char *` | znaki ciągu (w liczbie odpowiadającej precyzji lub poprzedzające `\0`) |
| `f` | `double` | [-]*m.dddddd* |
| `e, E` | `double` | [-]*m.ddddddd*e±*xx* lub [-]*m.ddddddd*E±*xx* |
| `g, G` | `double` | jak %e lub %E, jeżeli wykładnik jest mniejszy niż -4 albo większy lub równy precyzji; w p.p. jako %f |
| `p` | `void *` | wskaźnik |
| `%` | `-` | wypisuje znak % |

# Formatowanie danych wyjściowych

```
printf(format,"hello, world");
```

| format | wyjście |
|---|---|
| `:%s:` | `:hello, world:` |
| `:%10s:` | `:hello, world:` |
| `:%.10s:` | `:hello, wor:` |
| `:%-10s:` | `:hello, world:` |
| `:%.15s:` | `:hello, world:` |
| `:%-15s:` | `:hello, world   :` |
| `:%15.10s:` | `:     hello, wor:` |
| `:%-15.10s:` | `:hello, wor     :` |

# Formatowanie danych wyjściowych

- Istnieje jeszcze jedna funkcja podobna do **`printf()`** i **`fprintf()`**:

```
int sprintf(char *string, char *format, ...)
```

- Wynik zapisywany w ciągu **`string`** (rozmiar **`string`** musi być wystarczający).

Politechnika Wrocławska

# Formatowane dane wejściowe

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

- odczytuje znaki ze standardowego wejścia,
- interpretuje je zgodnie ze specyfikacją zawartą w ciągu **`format`**,
- pozostałe argumenty muszą być referencjami do zmiennych.

Politechnika Wrocławska

# Formatowane dane wejściowe

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

- kończy odczytywanie po dojściu do końca ciągu formatowania lub w przypadku, gdy dane wejściowe nie odpowiadają specyfikacji
- zwraca liczbę elementów, które udało się przypisać
- osiągnięcie końca pliku sygnalizuje zwracając **`EOF`**

Politechnika Wrocławska

# Formatowane dane wejściowe

- Istnieje jeszcze jedna funkcja podobna do **`scanf()`** i **`fscanf()`**:

```
int sscanf(char* string, char* format,
...)
```

- funkcja przeszukuje **`string`** zgodnie ze specyfikacją w **`format`**

Politechnika Wrocławska

# Formatowane dane wejściowe

Ciąg formatowania może zawierać:

- spacje i tabulatory (są pomijane),
- zwykłe znaki (z wyjątkiem %), które powinny być dopasowane do znaków drukowalnych w strumieniu danych wejściowych,
- znaczniki konwersji.

Politechnika Wrocławska

# Formatowane dane wejściowe

Znaczniki konwersji:

- znak `%`,
- opcjonalny znak `*` (wstrzymanie przypisania),
- opcjonalna liczba (określająca największą możliwą szerokość pola),
- opcjonalnej litery `h`, `l` lub `L`, która określa szerokość obiektu docelowego,
- znak określający typ danych.

Politechnika Wrocławska

# Formatowane dane wejściowe

| znak | typ argumentu | dane wejściowe |
|---|---|---|
| `d` | `int*` | liczba całkowita dziesiętna |
| `i` | `int*` | liczba całkowita – może być ósemkowa (znak 0 na początku) lub szesnastkowa (znaki 0x lub 0X na początku) |
| `o` | `int*` | liczba całkowita ósemkowa (znak 0 opcjonalny) |
| `x, X` | `int*` | liczba całkowita szesnastkowa (znaki 0x lub 0X opcjonalne) |
| `u` | `unsigned int*` | liczba dziesiętna bez znaku |

**mogą być poprzedzone literą `h` (`short*` zamiast `int*`)**
**lub `l` (`long*` zamiast `int*`)**

Politechnika Wrocławska

# Formatowane dane wejściowe

| znak | typ arg. | dane wejściowe |
|---|---|---|
| `c` | `char *` | znaki; dalsze znaki danych wejściowych (domyślnie 1) są umieszczane we wskazanym miejscu pamięci; pomijanie białych znaków wstrzymane; aby odczytać jeden znak drukowalny, używamy zapisu `%1s` |
| `s` | `char *` | ciąg znaków; wskaźnik do tablicy znaków dostatecznie dużej, aby zapisać ciąg wraz z końcowym znakiem `\0` |
| `e, f, g` | `float *` | liczba zmiennoprzecinkowa z opcjonalnym znakiem, opcjonalną kropką dziesiętną i opcjonalnym wykładnikiem; mogą być poprzedzone literą `l` (`double*` zamiast `float*`) |
| `%` | `-` | znak % |
| znak | typ arg. | dane wejściowe |
| `c` | `char *` | znaki; dalsze znaki danych wejściowych (domyślnie 1) są umieszczane we wskazanym miejscu pamięci; pomijanie białych znaków wstrzymane; aby odczytać jeden znak drukowalny, używamy zapisu `%1s` |

Politechnika Wrocławska

# Argumenty linii poleceń

Start here | 07_linia_polecen.c

```
1  /*07_linia_polecen.c*/
2  /*Wstep do programowania*/
3  /*Program wypisuje zawartosc plikow
4  wymienionych w linii polecen na stdout.
5  Zawiera funkcje do kopiowania plikow.
6  Ilustruje wykorzystanie argumentow
7  linii polecen*/
8
9  #include<stdio.h>
10
11 void filecopy(FILE*, FILE*);
12
13 int main(int argc, char** argv)
14 {
15     //deklaracja wskaznika do pliku
16     FILE *fp;
17
```

Politechnika Wrocławska

# Argumenty linii poleceń

Start here × 07_linia_polecen.c ×

```
18      //jezeli jest jeden argument linii polecen
19      //(nazwa programu)
20      if(argc == 1)
21      {
22          //kopiowane jest standardowe wejscie
23          filecopy(stdin, stdout);
24      }
25
26      //jesli sa jakies argumenty
27      else{
50
51      return 0;
52  }
```

Politechnika Wrocławska

# Argumenty linii poleceń

Start here | 07_linia_polecen.c

```
27      else{
28          //przejdz po wszystkich argumentach
29          while(--argc > 0)
30          {
31              //otworz plik (nazwa pliku jest wskazywana przez kolejny element
32              //w tablicy wskaznikow argv
33              //w razie niepowodzenia
34              if( (fp = fopen(*++argv, "r") ) == NULL )
35              {
36                  //wyswietl komunikat i zakoncz program
37                  printf("nie mozna otworzyc %s\n", *argv);
38                  return 1;
39              }
40              //jesli udalo sie otworzyc
41              else
42              {
43                  //skopiuj plik fp na stdout
44                  filecopy(fp, stdout);
45                  //zamknij plik
46                  fclose(fp);
47              }
48          }
49      }
```

Politechnika Wrocławska

# Kopiowanie plików

Start here | 07_linia_polecen.c

```
54  //funkcja kopiujaca plik ifp (input)
55  //do pliku ofp (output)
56  void filecopy(FILE *ifp, FILE *ofp)
57  {
58      int c;
59      //dopoki ( (znak wczytany z wejscia) jest rozny od EOF )
60      //wypisuj ten znak na wyjscie
61      while( (c = getc(ifp)) != EOF )
62          putc(c,ofp);
63  }
```

Politechnika Wrocławska

# Podsumowanie

- Obsługa łańcuchów znakowych
- Wskaźnik na plik
- Dostęp do plików
  - Odczyt plików tekstowych
  - Zapis plików tekstowych
- Formatowanie danych wyjściowych
- Formatowane dane wejściowe
- Argumenty linii poleceń