

NAZWA PRZEDMIOTU  
**Programowanie strukturalne**

**Temat wykładu 1.**

**Struktura języka C. Zmienne, stałe, operatory i wyrażenia.  
Wypisywanie wyników i wprowadzenie danych do programu.**

dr hab. inż. Jerzy Montusiewicz, prof. PL

# 1. Agenda

---

- 1.1. Informacje o wykładzie**
- 1.2. Wprowadzenie do programowania**
- 1.3. Tworzenie i uruchamianie programu**
- 1.4. Struktura języka C**
- 1.5. Zmienne, stałe, operatory, wyrażenia**
- 1.6. Wypisywanie wyników**
- 1.7. Wprowadzenie danych do programu**

# 1.1. Informacje o wykładzie

Program przedmiotu „Programowanie strukturalne” obejmuje 30 godz. wykładów oraz 30 godz. zajęć laboratoryjnych. Został wyceniony na 5 pkt. ECTS.

Przedmiot kończy się egzaminem, który odbędzie się w sesji egzaminacyjnej. Forma egzaminu: pisemna, test zawierający pytania typu otwartego, wielokrotnego wyboru, łączenia oraz pytania otwarte. Nie będzie pisania kodów programu. Zaliczenie od 51% zdobytych punktów, co 10% o pół stopnia wyżej.

## **Cele przedmiotu:**

- poznanie podstaw programowania strukturalnego z wykorzystaniem przykładu języka C,
- praktyczna nauka posługiwania się specyficznymi mechanizmami programowania w języku C,
- poznanie metod korzystania z biblioteki standardowej.

# 1.1. Informacje o wykładzie

**Wymagania wstępne w zakresie wiedzy, umiejętności i innych kompetencji:**

- algebra liniowa,
- analiza matematyczna,
- język angielski – stopień podstawowy.

**Efekty uczenia się w zakresie wiedzy:**

- Ma podstawową wiedzę o programowaniu strukturalnym i elementach języka C służących do strukturyzacji programów;
- Zna zaawansowane elementy programowania strukturalnego, takie jak wskaźniki, złożone typy danych i dynamiczna alokacja pamięci;
- Zna wysokopoziomowe i niskopoziomowe operacje wejścia-wyjścia i metody ich formatowania;
- Zna bieżący standard języka i wprowadzone przezeń nowe mechanizmy programistyczne.

# 1.1. Informacje o wykładzie

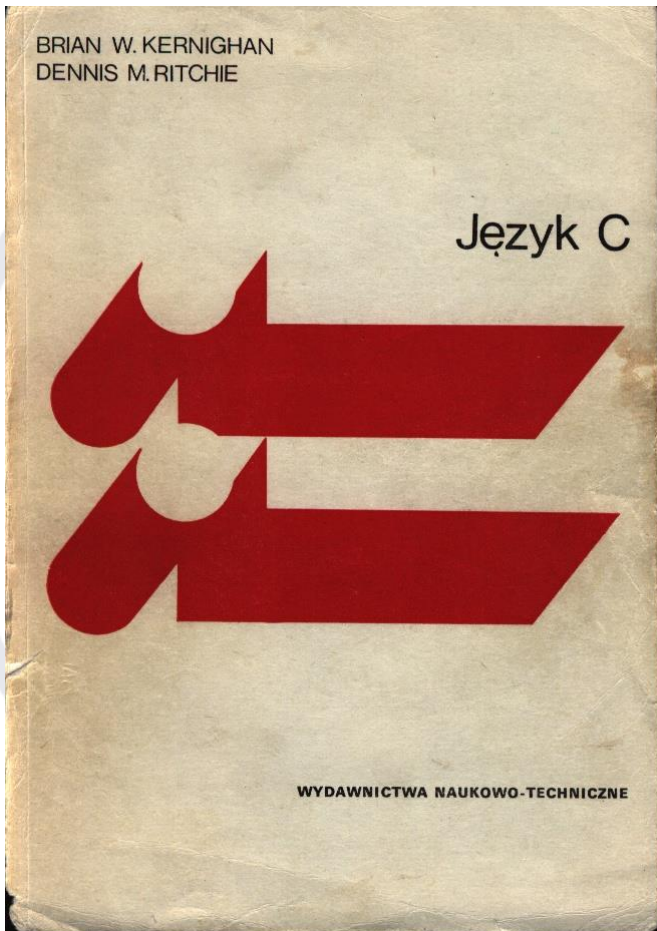
## **Efekty uczenia się w zakresie umiejętności:**

- Potrafi posługiwać się dokumentacją opisującą bibliotekę języka C, wyszukiwać niezbędne informacje w literaturze, także w języku angielskim;
- Potrafi opisać w sposób niesformalizowany wymagania wobec aplikacji o charakterze strukturalnym;
- Potrafi zaprojektować aplikację strukturalną o średnim i dużym stopniu złożoności;
- Potrafi wybrać i zastosować w praktyce właściwy sposób organizacji prac programistycznych, w tym technikę testowania aplikacji.

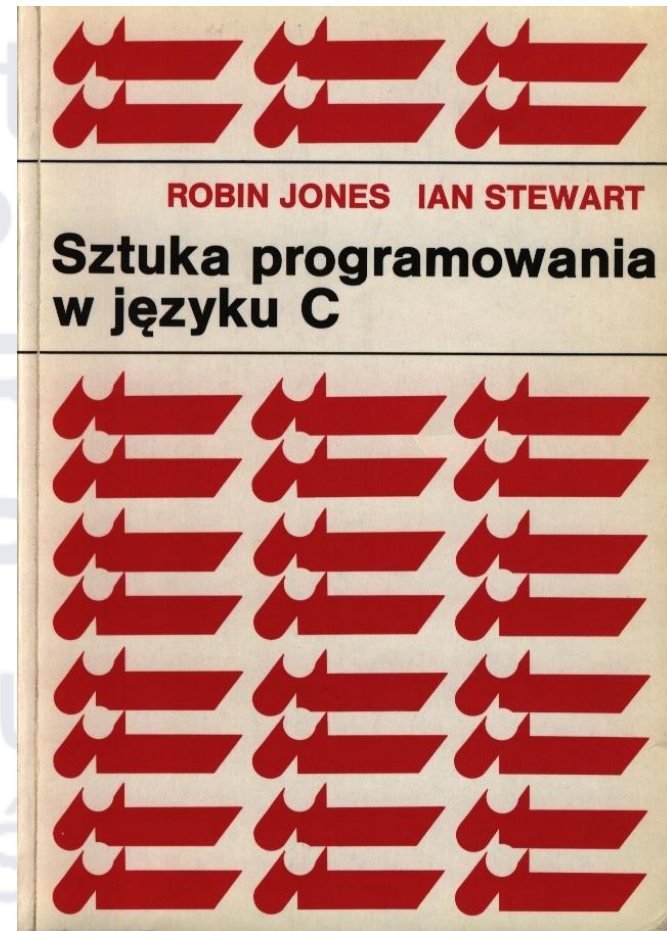
## **Efekty uczenia się w zakresie kompetencji społecznych:**

- Potrafi podejmować odpowiedzialne decyzje projektowe, również w obszarze pozatechnicznym;
- Potrafi myśleć kreatywnie w trakcie analizy i projektowania aplikacji.

# 1.1. Informacje o wykładzie, podręczniki



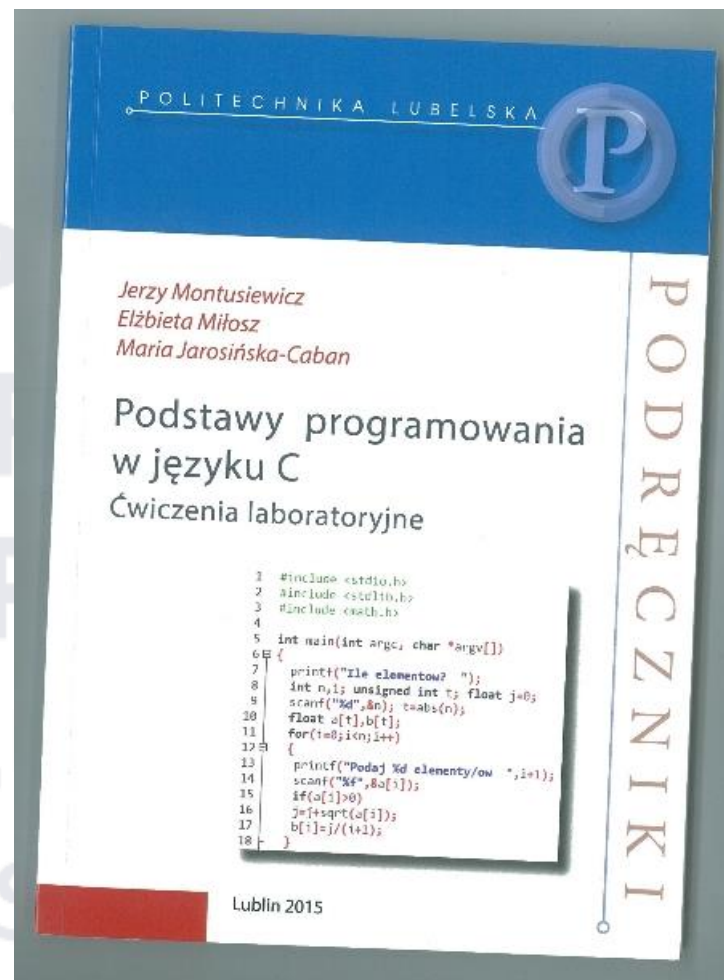
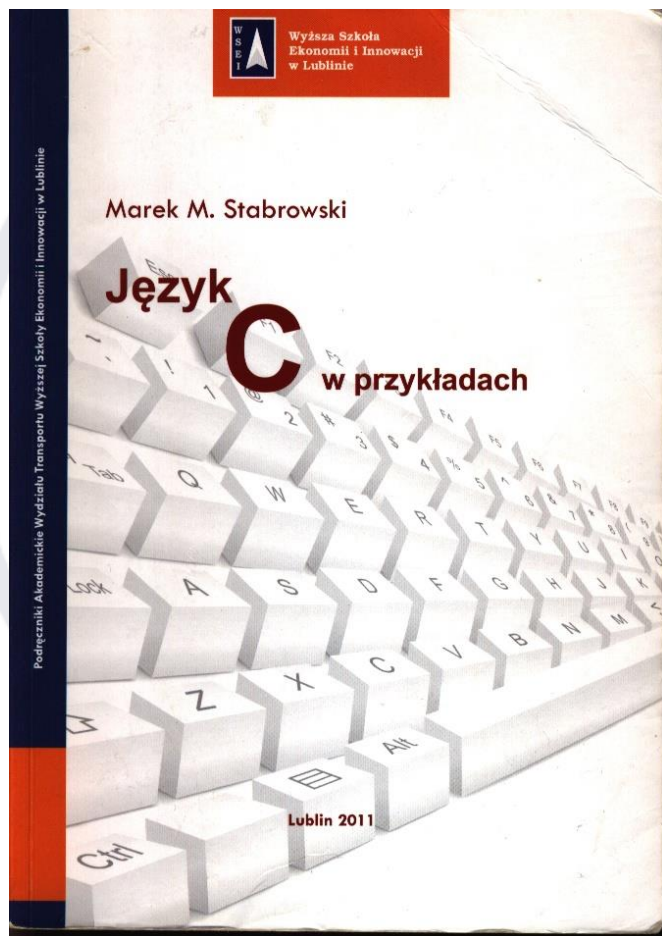
1978; The C Programming Language  
Pierwszy podręcznik



1987; The Art. of C Programming /  
WNT W-wa 1992



# 1.1. Informacje o wykładzie, podręczniki



**Podręcznik dostępny w wersji elektronicznej  
w bibliotece cyfrowej Politechniki Lubelskiej**

## 1.2. Wprowadzenie do programowania

### Podział języków:

- **języki niskiego poziomu**; składnia zbliżona do języka wewnętrznego maszyny cyfrowej (procesora); jednej instrukcji elementarnej języka odpowiada zazwyczaj jedna operacja elementarna procesora;
  - język maszynowy (zapis w postaci liczb);
  - język assemblerowy (składnia przyjazna);

```
0000: CD 28 00 A0 00 9A EE FE 1D F0 4F 03 06 1C 8A 03 .....
0010: 06 1C 17 03 06 1C BF 06 01 01 01 00 02 FF FF FF .....
0020: FF FF FF FF FF FF FF FF FF FF FF FF E1 1B 4E 01 .....
0030: C6 20 A4 00 10 00 A2 21 FF FF FF FF 00 00 00 00 .....
0040: 05 00 00 00 00 00 00 00 00 00 4F 06 1C 00 00 00 .....
0050: CD 21 CB 00 00 00 20 20 20 20 20 EE 10 35 C1 B0 .....
0060: 00 00 00 00 00 20 20 20 20 20 FF FF FF FF FF FF .....
0070: CA A1 45 20 A1 C0 CA 00 00 00 00 B2 B2 F0 32 00 .....
```

Kod w języku maszynowym, adres 1. komórki pamięci oraz wartości (system szesnastkowy)

Kod w języku assemblerowym

```
1:  .model small      ;komentarz
2:  .stack
3:  .data
4:  a db 3ah
5:  b db 19
6:  c db ?
7:  .code
8:  start:
9:  mov ax, @data      ;komentarz
10: mov ds., ax
11: mov al., a
12: mov ah, b
13: add ah, al..
14: mov c, ah
15: end
```



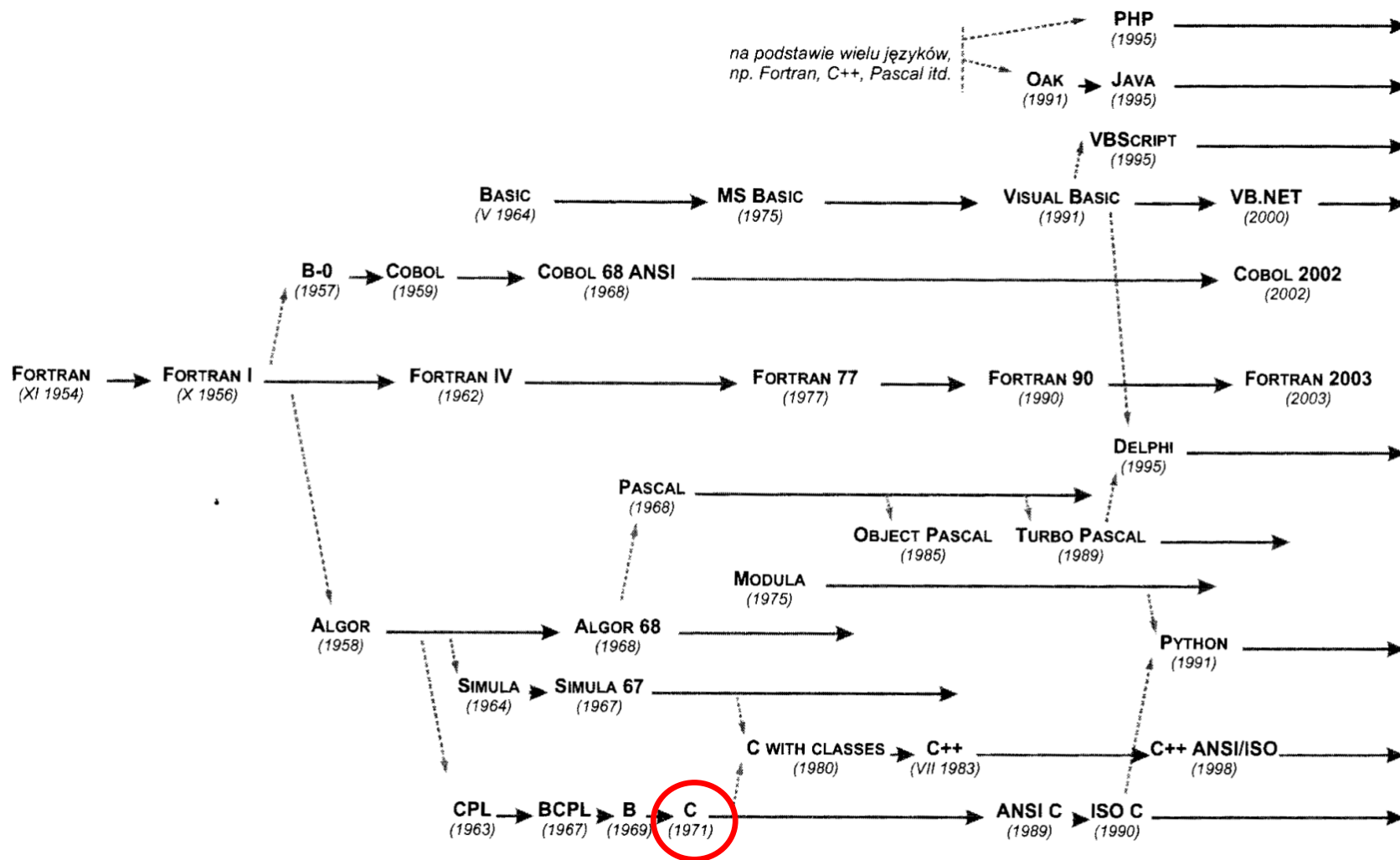
## 1.2. Wprowadzenie do programowania

- **języki wysokiego poziomu;**
  - rozbudowana składnia zbliżona w dużym stopniu do języka naturalnego człowieka;
  - jedna instrukcja elementarna języka realizowana jest przez bardzo dużą liczbę operacji elementarnych procesora.

Podział języków wysokiego poziomu, **programowanie typu:**

- proceduralnego i/lub strukturalnego (imperatywne);
- zorientowanego obiektowo (obektowe);
- specjalizowanego (np. sterowanie przepływem danych, współbieżnego).

# 1.2. Wprowadzenie do programowania



Historia rozwoju wybranych języków wysokiego poziomu

## 1.2. Wprowadzenie do programowania

### Kilka faktów z historii powstania języka C:

- CPL – Combined Programming Language (1963)
- BCPL – Basic CPL (1967)
- B (1969), język interpretowany, Ken Thompson z Bell Labs
- NB – Ken Thompson, Dennis Richie
- C (1971), język kompilowany
- Wprowadzanie przez producentów kompilatorów swoich zmian spoza standardu
- 1989 – wypracowanie standardu C89 (ANSI X<sub>3.159.1989</sub>)
- 1999 – nowy standard C99

Język C nadal jest wykorzystywany, jest dobrą podstawą do opanowywania innych języków. Składnia języka C stała się podstawą dla języków C++, C#, Java

ANSI – American National Standards Institute

## 1.2. Wprowadzenie do programowania

Podział ze względu na samodokumentowanie kodu:

- symboliczne nazwy instrukcji w postaci skrótów, stosowanie komentarzy do opisów;
- opisowe (nazwy zrozumiałe, wyrazy wyjaśniające przeznaczenie i zasadę działania).

Co to są paradygmaty programowania.

Paradygmat programowania określa sposób postrzegania przez programistę procedur tworzących program komputerowy.

Paradygmat programowania to wzorzec postępowania, który w konkretnym etapie rozwoju informatyki lub w poszczególnych obszarach zastosowania, jest częściej używany od innych.

## 1.2. Wprowadzenie do programowania

**Występujące paradygmaty prowadzą do programowania:**

- proceduralnego,
- funkcyjnego,
- obiektowego,
- zdarzeniowego,
- logicznego,
- agentowego lub
- strukturalnego,
- imperatywnego,
- uogólnionego,
- deklaratywnego,
- aspektowego,
- modularnego.

Istnieją języki hybrydowe, które łączą w sobie wiele elementów programowania, np. proceduralnego, obiektowego oraz uogólnionego (np. język C++).



## 1.3. Tworzenie i uruchamianie programu

### Etapy programowania:

1. Określenie celów programu.
2. Projektowanie programu (ALGORYTM).
3. Pisanie kodu.
4. Kompilacja.
5. Uruchomienie programu.
6. Testowanie i usuwanie błędów.
7. Pielęgnowanie i modyfikacja programu.

# 1.3. Tworzenie i uruchamianie programu

**Kod źródłowy** programu w języku C można napisać w dowolnym edytorze tekstu, który zapisuje znaki w kodzie ASCII, np. notatniku.

Program **Dev-C++** jest środowiskiem programistycznym dostępnym bezpłatnie na licencji GPL wykorzystującym kompilator GCC (GNU Compiler Collection), który może pracować na różnych platformach: Linux, DOS, WIN32, OS/2. Pakiety instalacyjne można pobrać ze strony:

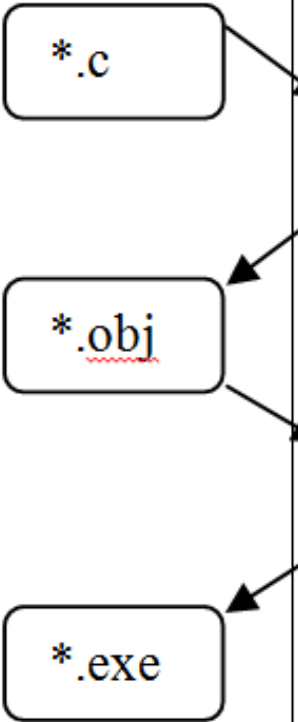
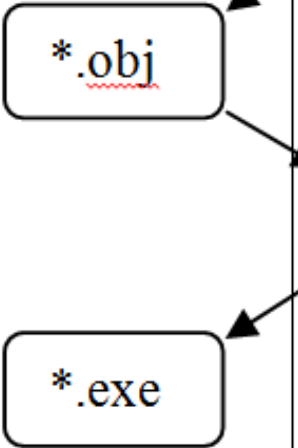
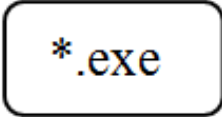
<http://www.bloodshed.net/devcpp.html>.

Cechy:

- wzajemna przenośność kodów utworzonych na platformach Windows oraz Linux,
- istnienie polskiej wersji językowej interfejsu.

**ASCII** – American Standard Code for Information Interchange

## 1.3. Tworzenie i uruchamianie programu

OBIEKT	PROCES	OPIS
 *.c	KOMPILACJA	Plik źródłowy w języku C
		Przetłumaczenie kodu języka C na język wewnętrzny komputera
 *.obj	ŁĄCZENIE	Skompilowana postać pliku źródłowego
		Konsolidacja (linkowanie) połączenie modułów
 *.exe		Wykonywalna postać pliku

## 1.3. Tworzenie i uruchamianie programu

**Kompilator** pozwala na przekształcenie kodu źródłowego do kodu wynikowego pozwalający na bezpośrednie uruchomienie go na komputerze:

- sprawdza poprawność składni napisanego kodu (np. brak średnika czy niewłaściwa nazwa).
- wykrywa błędy niepozwalające na wygenerowanie kodu wynikowego.
- generuje listę błędów wraz z opisem pozwalającym na łatwiejszą korektę kodu źródłowego.
- tworzy ostrzeżenia **typu warning**, dotyczące błędów logicznych.

**Etapy procesu kompilacji:**

- prekompilacja,
- właściwa kompilacja,
- optymalizacja kodu asemblera,
- asemblacji.

## 1.3. Tworzenie i uruchamianie programu

---

**Linker (konsolidator)** odpowiada za operacje zmierzające do utworzenia pliku binarnego z kodem wykonywalnym:

- połączenie skompilowanych wersji kodów źródłowych ze wskazanymi w nich funkcjami niezdefiniowanymi (z przyłączonych bibliotek),
- przypisanie kodu maszynowego do ustalonych adresów,
- wygenerowanie wersji wykonywalnej programu, gotowej do uruchomienia.



## 1.4. Struktura języka C

**Programowanie strukturalne** polega na podziale kodu źródłowego programu na:

- funkcje – są blokiem instrukcji wywoływane w programie pojedynczym poleceniem,
- bloki – grupa instrukcji traktowanych jako jedna całość zawierająca się pomiędzy { ... }, wykorzystanie instrukcji przypisania, wyboru (struktury kontrolne) i pętli. Bloki usytuowane są hierarchicznie, mogą zawierać inne bloki.

Takie programowanie znacznie zwiększa przejrzystość kodu, ułatwia to jego czytelność. Rezygnacja z instrukcji skoków bezwarunkowych.

Język C nie posiada słów kluczowych odpowiedzialnych za obsługę wejścia i wyjścia. Zbudowano zestaw *Bibliotek Standardowych* zawierające funkcje wykonujące wczytywanie i zwracanie danych, modyfikowanie zmiennych, operacje na plikach i inne.

## 1.4. Struktura języka C

Program zbudowany jest z modułów-funkcji będących pewnymi względnie zamkniętymi ciągami instrukcji.

***typ\_zwracany** – to typ zmiennej, np. całkowita – int, zmiennoprzecinkowa – float, double zwracana na zewnątrz przez funkcję (instrukcja return).*

**typ\_zwracany nazwa\_funkcji (lista argumentów)**

{ *Sygnatura/nagłówek funkcji*

deklaracje zmiennych lokalnych;  
czynne instrukcje;  
wywoływane funkcje definiowane;  
wywoływane funkcje biblioteczne

} ***Lista argumentów** zawiera informacje o typach argumentów, ich identyfikatorach (nazwach), oddzielone przecinkami,*

**{ }** – „ciało funkcji”

C używa **32** słów kluczowych (standard **C89**) oraz **5** nowych **C99**.

Wiele operacji realizują funkcje z biblioteki standardowej.

## 1.4. Struktura języka C

Każdy program w C musi zawierać funkcję **main( )**

**Przykład, po włączeniu nowego projektu (Dev-cpp):**

*Od znaku #. hash zapisujemy  
tzw. dyrektywy kompilacyjne.*

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

*Preprocesor wyszukuje dyrektywy i wykonuje  
edycję kodu źródłowego (np. wstawienie,  
zamianę) przed wkroczenie kompilatora*

***stdio** (ang. standard input-output),  
.h – plik nagłówkowy C*

```
/* run this program using the console pauser or add your own getch,  
system("pause") or input loop */
```

```
int main(int argc, char *argv[])  
{  
    system("pause")  
    return 0;
```

```
}
```

*Funkcja powstrzymuje dalsze  
wykonanie programu.*

## 1.5. Zmienne, stałe, operatory, wyrażenia

**Zmienna** – to identyfikator posiadający własną nazwę, która może przechowywać wartość ustalonego typu zajmująca fragment pamięci o ustalonym rozmiarze.

**Nazwa** zmiennej (identyfikator) to ciąg znaków zawsze zaczynający się od litery, mogą występować cyfry i inne znaki, nie można zastosować spacji (używamy znaku podkreślenia).

Zasady powszechnie przyjęte:

- nazwy zmiennych piszemy małymi literami: a2, nowe\_pole,
- nazwy stałych zdefiniowanych jako dyrektywy kompilatora za pomocą #define piszemy kapitalikami: LL2, STARY,
- nazwy funkcji piszemy małymi literami: drukuj(), pisz(), oblicz().

**Język C rozróżnia małe i DUŻE litery.**

**Można używać tylko znaków „angielskich”.**

# 1.5. Zmienne, stałe, operatory, wyrażenia

## Deklaracja zmiennych

```
typ nazwa_zmiennej1, nazwa_zmiennej2;  
float bok1, dal3;
```

Podstawowe typy zmiennych:

**int** → liczby całkowite (4B),

**float** → liczby zmiennopozycyjne (zmiennoprzecinkowe), rzeczywiste (4B),

**double** → liczby zmiennopozycyjne podwójnej precyzji (8B),

**char** → liczby całkowite do przechowywania znaków (1B), zapis znaków ASCII

Typ przekazuje informację w jaki sposób będą na nich wykonywane operacje

Zmiennej w chwili jej deklaracji można nadać wartość początkową.

```
float bok2 = 12.5;
```

Zadeklarowane zmienne bez inicjalizacji mogą zawierać „śmieci”.

Zadeklarowanie zmiennej musi nastąpić przed jej użyciem.

**ASCII** – American Standard Code for Information Interchange



## 1.5. Zmienne, stałe, operatory, wyrażenia

**Stałe** to zmienne, którym nie można przypisać innej wartości w trakcie działania programu. Wartość zostaje ustalona w kodzie programu.

Definicja stałej (1. sposób):

```
const typ nazwa_stalej = wartość;  
const float pi = 3.14;
```

Definicja stałej (2. sposób):

Stosujemy dyrektywy preprocesora,

```
#define NAZWA wartość  
#define PI 3.14
```

Uzyskujemy tzw. **stałą symboliczną (MAKRO)**. Następuje zastąpienie stałej jej wartością w każdym miejscu kodu przed uruchomieniem procesu kompilacji.

## 1.5. Zmienne, stałe, operatory, wyrażenia

W języku C możemy wyróżnić następujące **typy operatorów**:

- arytmetyczne,
- bitowe,
- relacyjne (porównania),
- logiczne,
- inne (dotyczące tablic, wskaźników, struktur pewnych obliczeń)

### Operatory arytmetyczne:

- dodawanie „**+**”,
- odejmowanie „**-**”,
- mnożenie „**\***”,
- dzielenie „**/**”,
- reszta z dzielenia „**%**” (tzw. *dzielenie modulo*),
- Inkrementacja „**++**” i dekrementacja „**--**” (zmiana o 1 lub -1).

**Pamiętaj!** Uzyskany wynik jest powiązany z typem zmiennych.

## 1.5. Zmienne, stałe, operatory, wyrażenia

**Wyrażenia arytmetyczne** – to dowolne wyrażenie typu liczbowego, które może być złożone z liczb, zmiennych, funkcji połączonych operatorami (symbolami działań). W wyrażeniach mogą też występować nawiasy okrągłe „()” decydujące o ostatecznej kolejności wykonywania operacji.

**Czym jest ten znak „=” ?**

**To nie jest znak równości.**

„=” – to jest znak **przypisania** wartości znajdującej się po prawej stronie, zmiennej znajdującej się po lewej stronie, np.:

`nowy = nowy + 5`

## 1.6. Wypisywanie wyników

Język C nie ma słów kluczowych do wypisywania (wyprowadzania) danych (wartości i tekstów). Do tego celu stosowane są funkcje biblioteczne.

Funkcja **printf()** wyprowadza dane na ekran (standardowe urządzenia wyjścia). Znajduje się w pliku nagłówkowym `stdio.h`

```
printf("łańcuch sterujący", arg1, arg2);
```

**łańcuch sterujący** – zawiera tekst, który będzie wypisany na monitorze oraz znaki nie drukowalne, kody do wyprowadzenia wartości zmiennych (np. `%d` – dla zmiennych całkowitych, typ **int**) oraz znaki sterujące wydrukiem (np. `\t` – znak tabulacji).

`" "` – cudzysłów służy do umieszczania tekstu, znaków sterujących oraz kodów formatujących

## 1.6. Wypisywanie wyników

**arg1** – argumenty są oddzielane od siebie przecinkami. W miejsce argumentu można wstawić wyrażenie lub nazwę funkcji którą wywołujemy.

Przykładowe kody do wypisania wartości:

**%d** → zmienna typu **int**,

**%f** → zmienna typu **float**,

Przykładowe znaki sterujące:

**\n** → przejście do nowego rzędu (linii),

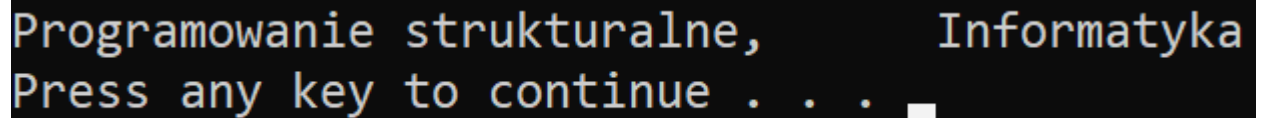
**\t** → znak tabulacji, przesunięcie wydruku w nowe miejsce w tej samej linii.



## 1.6. Wypisywanie wyników, p1.6a

Wyprowadź napisy na monitor z przesunięcie w jednej linii.

```
#include <stdio.h>
#include <stdlib.h>
/* run this program using the console pauser or add your own getch,
system("pause") or input loop */
int main()
{ /* wywołanie funkcji bibliotecznej */
    printf("Programowanie strukturalne,\t Informatyka\n");
    /* łańcuch sterujący bez kodów wyprowadzania danych, występują
znaki sterujące (niedrukowalne) */
    system("PAUSE");
    return 0;
}
```



```
Programowanie strukturalne, Informatyka
Press any key to continue . . . _
```

## 1.6. Wypisywanie wyników, p1.6b

Oblicz średnią arytmetyczną i wyprowadź wartość na monitor.

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

```
{ float a1, a2;
  float averange;
  a1=3.6;
  a2=6.7;
```

```
  averange = (a1+a2)/2;
```

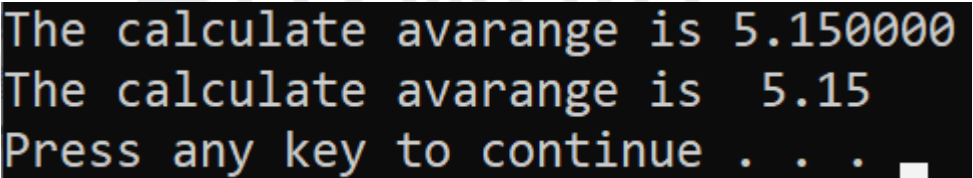
```
  printf("The calculate avarange is %f \n", averange);
```

```
  printf("The calculate avarange is %5.2f \n", averange);
```

```
  system ("pause");
```

```
  return 0;
```

```
}
```



```
The calculate avarange is 5.150000
The calculate avarange is  5.15
Press any key to continue . . .
```

*%5.2f – modyfikacja kodu %f,  
zdeklarowano 5 pól na całą liczbę  
w tym 2 pola na część dziesiętną*

## 1.7. Wprowadzenie danych do programu

Do wprowadzania danych do programu ze standardowego urządzenia (klawiatury) stosujemy funkcję **scanf()** znajdująca się w pliku nagłówkowym **stdio.h**

```
scanf("łańcuch sterujący", &arg1, &arg2);
```

**" "** – cudzysłów służy do umieszczania kodów formatujących  
**kody formatujące** – muszą być zgodne z typem zmiennych umieszczonych w liście argumentów (np. **%f** – dla zmiennych zmiennopozycyjnych, typ **float**).

Rodzaj separatora między kodami formatującymi musi być zastosowany przy wprowadzaniu danych z klawiatury (np. **"/" "**, **"," "** **""**).

**&** – operator adresowy (ampersant), umożliwia wprowadzenie wartości do komórki pamięci komputera.

## 1.7. Wprowadzenie danych do programu, p1.7a

Oblicz średnią arytmetyczną, dane wprowadź z klawiatury i wyprowadź obliczoną wartość na monitor.

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

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

```
int main()
```

```
{ float a1, a2;
```

```
printf("Wpisz dwie liczby rzeczywiste (.../...)\n");
```

```
scanf("%f/%f", &a1, &a2);
```

```
printf("Obliczona srednia = %f \n", (a1+a2)/2);
```

```
system("pause");
```

```
return 0;
```

```
}
```

*wstawiono wyrażenie*

```
Wpisz dwie liczby rzeczywiste (.../...)
4.5/7.3
Obliczona srednia = 5.900000
Press any key to continue . . .
```

## 1.7. Wprowadzenie danych do programu, p1.7b

Oblicz średnią arytmetyczną, dane wprowadź z klawiatury i wyprowadź obliczoną wartość na monitor.

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

```
int main()
```

```
{ int a1, a2;
```

```
float averange;
```

```
printf("Wpisz dwie liczby calkowite (.. , ..) \n");
```

```
scanf("%d,%d", &a1, &a2);
```

```
averange = (a1+a2)/2;
```

```
printf("Pierwsza liczba = %d \n", a1);
```

```
printf("Druga liczba = %d \n", a2);
```

```
printf("Srednia = %5.2f \n", averange);
```

```
return 0;
```

```
}
```

```
Wpisz dwie liczby calkowite (.. , ..)
2,11
Pierwsza liczba = 2
Druga liczba      = 11
Srednia =  6.00
```

*Poprawna wartość  
wynosi 6.5*

*Obliczenia w dziedzinie  
liczb całkowitych*

Materiały zostały opracowane w ramach projektu  
*„Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga”*,  
umowa nr **POWR.03.05.00-00-Z060/18-00**  
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020  
współfinansowanego ze środków Europejskiego Funduszu Społecznego



NAZWA PRZEDMIOTU  
**Programowanie strukturalne**

**Temat wykładu 2.**

**Wprowadzenie do algorytmiki. Schematy Nassi-Schneidermana.  
Programy imperatywne.**

dr hab. inż. Jerzy Montusiewicz, prof. PL

## 2. Agenda

---

- 2.1. Wprowadzenie do algorytmiki.**
- 2.2. Algorytmy opisowe.**
- 2.3. Schematy blokowe.**
- 2.4. Schematy Nassi-Schneidermana (N/S).**
- 2.5. Programy imperatywne.**

## 2.1. Wprowadzenie do algorytmiki

Co to jest algorytm? Przykłady definicji.

Algorytm **to skończony, uporządkowany ciąg** określonych działań lub czynności koniecznych do wykonania określonego zadania.

W języku potocznym algorytm oznacza **przepis** (zbiór koniecznych czynności uporządkowanych we właściwej kolejności), dzięki któremu możemy sprawnie przygotować posiłek lub wymienić zużyte baterie w urządzeniu technicznym.

Postać prezentowania algorytmów.

- **opisowa**,
  - o **lista kroków**,
  - o **pseudojęzyk** (ang. pseudocode),
- **graficzna**,
  - o **schemat blokowy** (ang. flowcharts),
  - o **schematy Nassi-Schneidermana** (schematy N/S).

## 2.2. Algorytmy opisowe

Cel przygotowywania algorytmów.

**Przygotowanie algorytmu pozwala na sprawną implementację zaproponowanego rozwiązania na wybrany język programowania.**

Konstrukcje stosowane w pseudojęzyku.

- **zdanie proste**

**przypisz** średniej **wartość** zero

**czytaj** x

**pisz** wynik

- **zdanie decyzyjne**

**jeżeli** warunek **to** zdanie

lub

**jeżeli** warunek **to** zdanie1

**w przeciwnym przypadku**

zdanie2

## 2.2. Algorytmy opisowe

- **zdanie iteracyjne** podczas gdy  
podczas gdy warunek **wykonuj** zdanie
- **zdanie iteracyjne** powtarzaj  
powtarzaj zdanie **aż** warunek
- **zdanie iteracyjne** dla  
dla lista sytuacji **wykonuj** zdanie
- **zdanie wybierz**  
**wybierz** przełącznik z  
wartość1: zdanie1  
wartość2: zdanie2 (...) **w innym przypadku** zdanie\_domyślne
- **zdanie grupujące** {...} lub **begin ... end**  

{	<b>begin</b>
zdanie1	zdanie1
zdanie2	zdanie2
...	...
}	<b>end</b>

## 2.2. Algorytmy opisowe, p2-2a

Obliczanie średniej arytmetycznej dwóch liczb (zdanie proste).

### Lista kroków:

1. Wczytaj dwie liczby  $a$  i  $b$
2. Dodaj do siebie te liczby i wynik podziel przez 2  $\rightarrow (a+b)/2$
3. Wyświetl wynik. Zakończ algorytm.

### Pseudokod:

1. Czytaj dwie liczby  $a$  i  $b$
2. Przypisz średniej wartość:  $srednia = (a+b)/2$
3. Pisz  $srednia$ .



## 2.2. Algorytmy opisowe, p2-2b

**Obliczanie średniej ocen studenta (zdanie iteracyjne typu **dla**).**  
Student kierunku Inżynieria logistyki w czasie sesji zimowej zdaje  $n$  egzaminów. Obliczyć średnią ocen w sesji.

### **Lista kroków:**

1. Wczytaj liczbę egzaminów  $n$
2. Wyzeruj zmienną  $s \rightarrow s=0$
3. Powtarzaj  $n$  razy:
  - wczytaj kolejną ocenę  $x$
  - dodaj  $x$  do dotychczas obliczonej sumy  $s \rightarrow s=s+x$
4. Oblicz wartość średniej  $\rightarrow s=s/n$
5. Wyświetl wynik  $s$ . Zakończ algorytm.

## 2.2. Algorytmy opisowe, p2-2b

**Obliczanie średniej ocen studenta (zdanie iteracyjne typu **dla**).**

Student kierunku *Inżynieria logistyki* w czasie sesji zimowej zdaje  $n$  egzaminów. Obliczyć średnią ocen w sesji.

### **Pseudokod:**

1. Czytaj  $n$
2. Przypisz  $s$  wartość zero  $\rightarrow s=0$
3. Dla zmiennej  $i$  zmieniającej się od 1 do  $n$  wykonuj:
  - {czytaj  $x$
  - przypisz  $s$  wartość poprzednią  $+x \rightarrow s=s+x$  }
4. Przypisz  $s$  wartość  $\rightarrow s=s/n$
5. Pisz  $s$ .

## 2.3. Schematy blokowe

Wśród algorytmów graficznych wyróżniamy:

- schematy blokowe,
- schematy Nassi-Schneidermana.

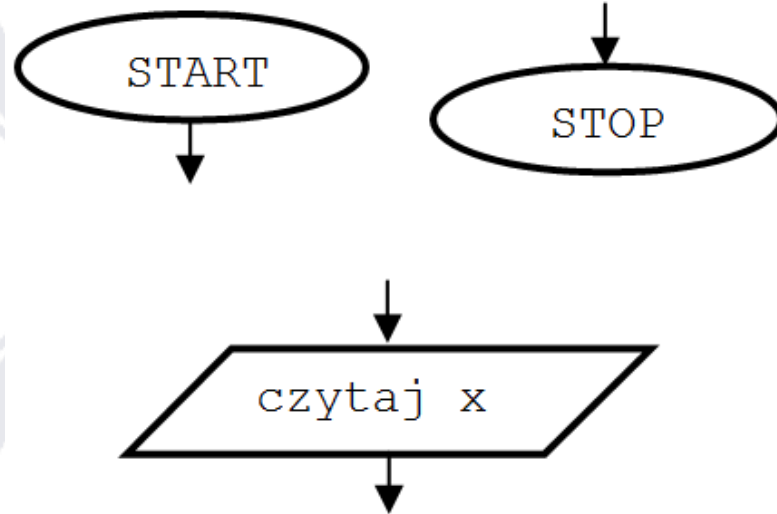
### **Schemat blokowy:**

- to narzędzie prezentujące w graficznej postaci kolejność działań (czynności) w tworzonym algorytmie,
- składa się figur geometrycznych (np. prostokąt, romb, owal) prezentujących różne działania,
- zawiera strzałki wskazujące powiązania między elementami schematu oraz kierunek przepływu informacji.

## 2.3. Schematy blokowe, podstawowe elementy

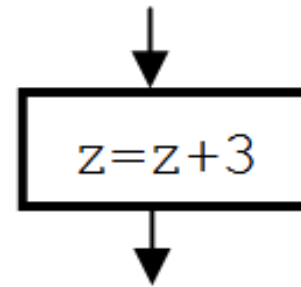
**Bloki  
graniczne**

**Blok  
wejścia-wyjścia**

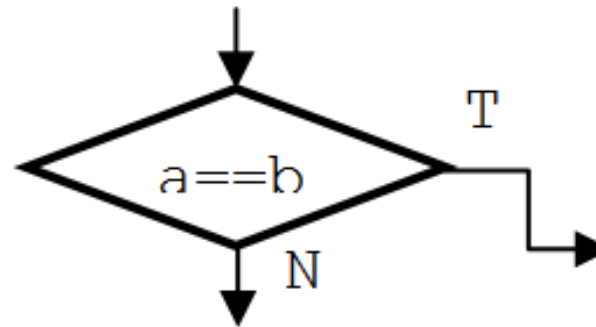


## 2.3. Schematy blokowe, podstawowe elementy

**Blok  
operacyjny**



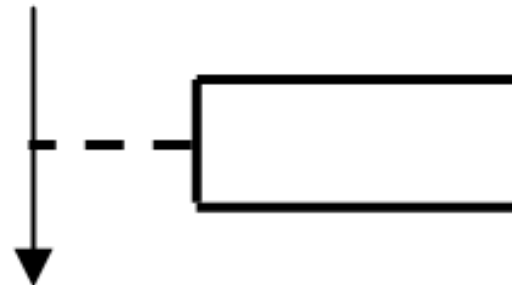
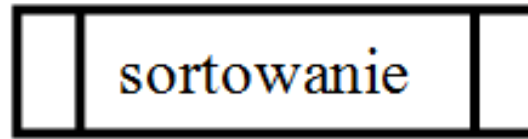
**Blok  
decyzyjny,  
warunkowy**



## 2.3. Schematy blokowe, podstawowe elementy

**Blok  
podprogramu  
(funkcji)**

**Blok  
komentarza**

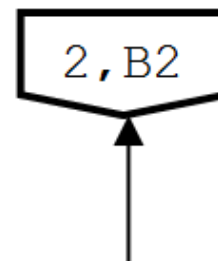
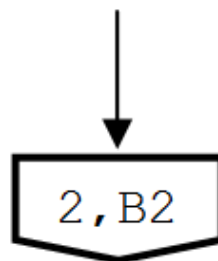
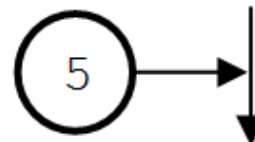
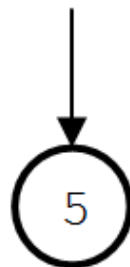




## 2.3. Schematy blokowe, podstawowe elementy

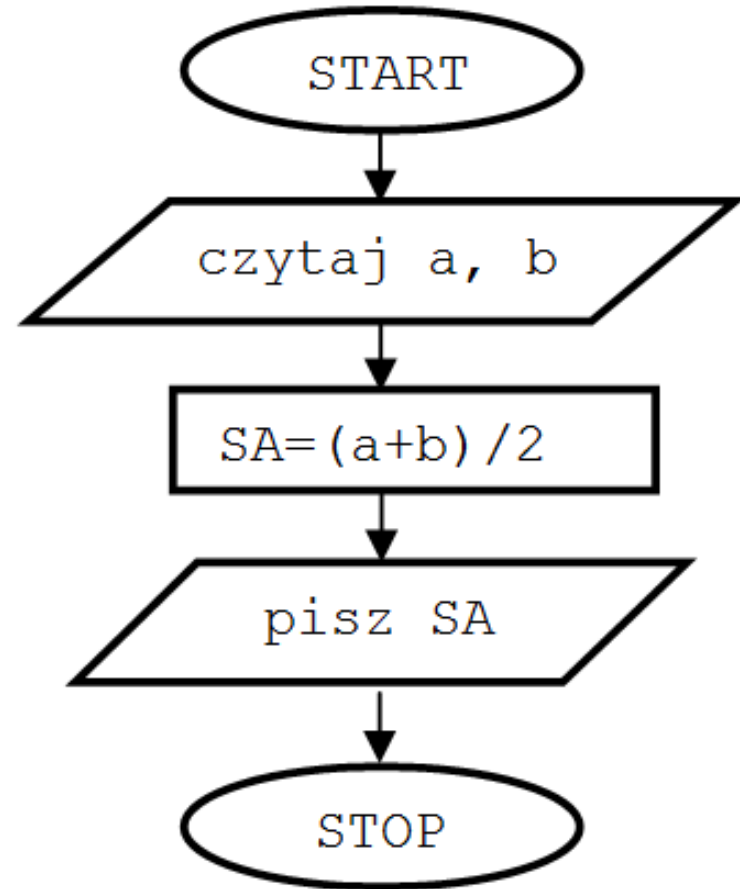
**łącznik  
wewnątrzstronicowy  
(wewnętrzny)**

**łącznik  
międzystronicowy  
(zewnętrzny)**



## 2.3. Schematy blokowe, p2-3a

Obliczanie średniej arytmetycznej dwóch liczb (algorytm liniowy).



## 2.4. Schematy Nassi-Schneidermana (N/S)

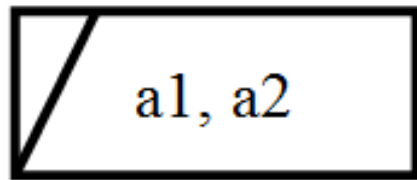
Graficzne przekazanie kolejności działań w przygotowanym algorytmie.

### Charakterystyka:

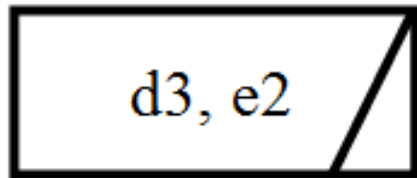
- pokazuje wprost główną ideę algorytmu – jego strukturę,
- nie występują strzałki,
- domyślny kierunek powiązania elementów przebiega z góry na dół.

Bardziej zwarta forma niż schemat blokowy.

## 2.4. Schematy (N/S), podstawowe elementy

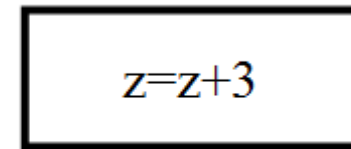


**Blok  
wejścia**

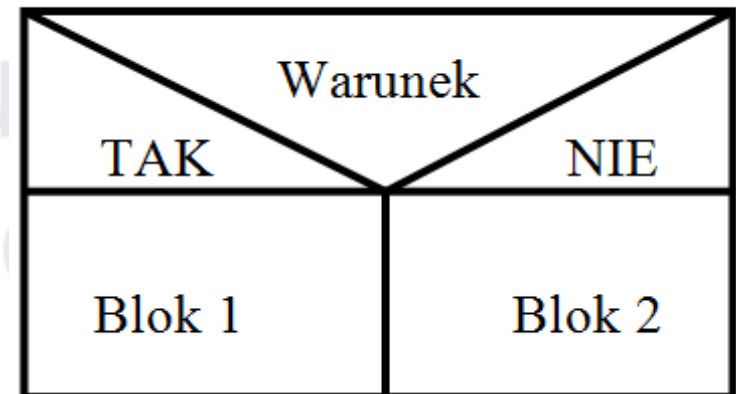


**Blok  
wyjścia**

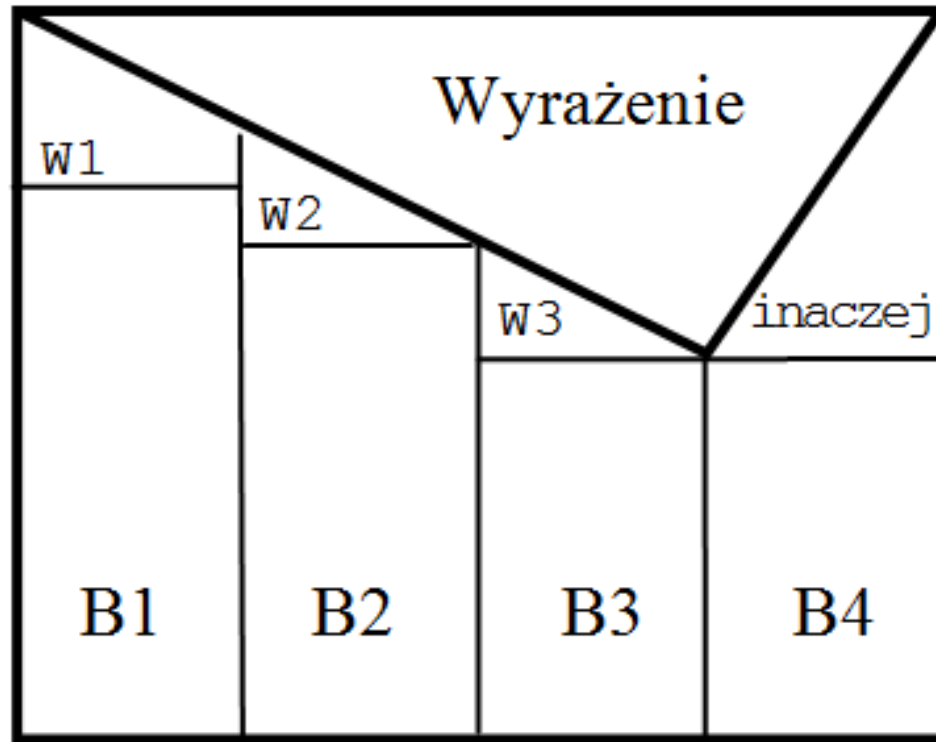
**Blok  
operacyjny**



**Blok  
decyzyjny**



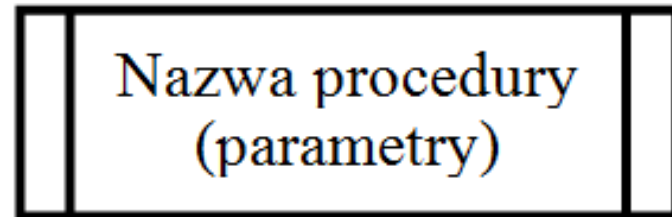
## 2.4. Schematy (N/S), podstawowe elementy



**Blok wyboru  
z szeregu  
możliwości**

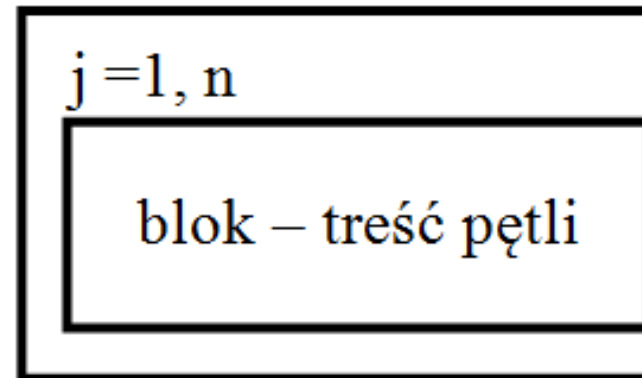
## 2.4. Schematy (N/S), podstawowe elementy

**Blok wywołania  
procedury**



**Blok iteracyjny (BI)  
pętla**

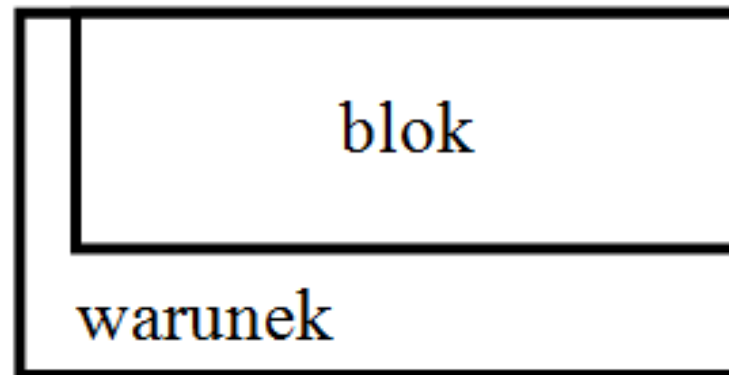
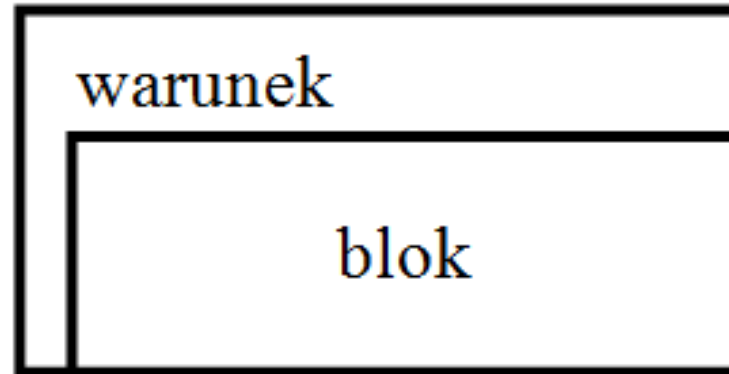
*z istniejącą  
zmienną  
sterującą*



## 2.4. Schematy (N/S), podstawowe elementy

Bl, podczas gdy  
*Dopóki*  
    <warunek>  
wykonuj  
    <blok>

Bl, powtarzaj  
*Powtarzaj*  
    <blok>  
aż do  
    <warunek>





## 2.5. Programy imperatywne

---

**Programowanie imperatywne** – to programowanie, które opisuje proces wykonywania jako sekwencję instrukcji zmieniających stan programu. Wyraża żądania jakichś czynności do wykonania. Programy imperatywne składają się z ciągu komend do wykonania przez komputer.

To najprostsza forma pisania kodu programu. Wyższą organizacją programu jest programowanie strukturalne i proceduralne. W programowaniu imperatywnym nie piszemy własnych funkcji.

## 2.5. Programy imperatywne, p2-6a

**Przelicz stopy na metry. Wartość wprowadzamy z klawiatury.**

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

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

```
int main()
```

```
{ int feet;
```

```
float meters;
```

```
printf ("Wprowadz liczbe stop (int) : ");
```

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

Operator adresowy "&".

```
/* Konwersja stop na metry */
```

```
meters = feet*0.3048;
```

```
printf ("\n%3d stop to %5.2f metra/ow\n\n", feet, meters);
```

```
system("PAUSE");
```

```
return 0;
```

```
}
```

```
Wprowadz liczbe stop (int) : 5
```

```
5 stop to 1.52 metra/ow
```

```
Press any key to continue . . .
```

## 2.5. Programy imperatywne, p2-6b

Napisz program wyświetlający informacje o autorze programu, obliczający pole koła i wypłatę dla 1 pracowników. Wartości wprowadzamy z klawiatury.

```
int main() { int ROK=1; int lg; // deklaracje
float PREMIA=0.20, PI=3.14159; float r, pole, stawka, wypłata;
printf("*****\n"); // instrukcje
printf("Programowanie liniowe\n");
printf("Autor programu: %s, kierunek: %s, rok:%d\n", "John Brown",
"Informatyka", ROK);
printf("=====\n"); Dane będą wprowadzane w tej samej linii
printf("Podaj promień koła "); scanf("%f",&r);
pole=PI*r*r; printf("Pole koła o promieniu %0.2f =%0.2f\n",r, pole);
printf("*****\n");
printf("Pracownik1\n"); printf("Podaj liczbe godzin "); scanf("%d",&lg);
printf("Podaj stawke "); scanf("%f",&stawka);
wypłata= lg*stawka+ lg*stawka*PREMIA;
printf("Wypłata = %0.2f\n",wypłata);
printf("=====\n");
system("PAUSE");
return 0; }
```

## 2.5. Programy imperatywne, p2-6b

Napisz program wyświetlający informacje o autorze programu, obliczający pole koła i wypłatę dla 1 pracowników. Wartość wprowadzamy z klawiatury.

REZULTAT

```
*****
Programowanie liniowe
Autor programu: John Brown, kierunek: Informatyka, rok:1
=====
Podaj promień koła  3
Pole koła o promieniu 3.00 =28.27
*****

Pracownik1
Podaj liczbe godzin  40
Podaj stawke  17.5
Wypłata = 840.00
Press any key to continue . . .
```

Materiały zostały opracowane w ramach projektu  
*„Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga”*,  
umowa nr **POWR.03.05.00-00-Z060/18-00**  
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020  
współfinansowanego ze środków Europejskiego Funduszu Społecznego

NAZWA PRZEDMIOTU  
**Programowanie strukturalne**

**Temat wykładu 3.**  
**Funkcje standardowe i funkcje własne. Prototypy funkcji.**

dr hab. inż. Jerzy Montusiewicz, prof. PL

## 3. Agenda

---

**3.1. Struktura programu z wielu funkcji.**

**3.2. Funkcje standardowe.**

**3.3. Prototypy funkcji.**

**3.4. Funkcje własne.**

Zintegrowany  
Program  
Rozwoju  
Politechniki  
Lubelskiej -  
część druga



## 3.1. Struktura programu z wielu funkcji

Program zbudowany z wielu funkcji składa się z:

- funkcji `main()`,
- funkcji bibliotecznych (znajdują się w standardowych bibliotekach języka C), konieczność podania pliku nagłówkowego,
- funkcji własnych (zdefiniowanych przez programistę).

Dobra funkcja – to funkcja realizująca jedno dobrze określone zadanie.

Funkcje stosujemy ze względu na:

- możliwość/konieczność podziału dużego problemu numerycznego na zadania mniejsze. Uzyskujemy zadania proste, łatwiejsze do przygotowania i testowania,
- Możliwość wielokrotnego użycia do nowych celów (zbudowanie własnej biblioteki)

## 3.1. Struktura programu z wielu funkcji, p3-1a

**Przykład**, program zawiera funkcję własną **linia()** oraz funkcję biblioteczną **fabs()** obliczającą wartość bezwzględna liczby.

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

```
#include <math.h>
```

```
void linia2(){  
printf("\n+++++\n");}
```

```
float main()
```

```
{ int war2;  
printf("Podaj wartość ujemna \n");  
scanf("%d", &war2);  
printf("argument %d, abs = %lf\n", war2, fabs(war2));  
linia2();  
return 0;  
}
```

```
Podaj wartość ujemna  
-17  
argument -17, abs = 17.000000  
+++++
```

## 3.2. Funkcje standardowe

Funkcje standardowe zostały pogrupowane w pewne grupy. Aby można się nimi posługiwać należy dyrektywą `#include` dołączyć wybrany plik nagłówkowy, np. `#include <math.h>` , `#include "moje.h"`,

`< . . . >` – oznacza, że plik będzie poszukiwany w katalogu systemowym, który został utworzony podczas instalacji środowiska programistycznego,

`" . . . "` – oznacza, że plik będzie poszukiwany w pierwszej kolejności w bieżącym folderze, a później w katalogu systemowym.

**Ważne!** Pamiętać przy przenoszeniu napisanego programu na inny zestaw komputerowy o skopiowaniu przygotowanych plików nagłówkowych.

## 3.2. Funkcje standardowe

Grupy wybranych funkcji standardowych:

- funkcje matematyczne **<math.h>**  
asin(x), acos(x), atan(x), sin(x), cos(x), tan(x), exp(x), log(x),  
pow(x,y), sqrt(x), ceil(x), floor(x), fabs(x), fmod(x,y), ...
- funkcje łańcuchowe **<string.h>**  
strlen(), strcat(), strcmp(), strcpy(), ...
- funkcje znakowe **<ctype.h>**  
tolower(), toupper(), isalpha(), isdigit(), isalnum(), ...
- funkcje ogólnego użytku **<stdlib.h>**  
abs(), rand(), qsort(), ...

## 3.2. Funkcje standardowe, przykłady

### Funkcja fabs( )

`#include <math.h>`

**double fabs( double arg );**      |arg|

*Opis:* Funkcja fabs() zwraca wartość bezwzględną z liczby *arg*.

### Funkcja pow( )

`#include <math.h>`

**double pow( double baza, double exp );**       $\text{baza}^{\text{exp}}$

*Opis:* Funkcja pow( ) zwraca wartość *bazy* podniesioną do potęgi *exp*. Błąd dziedziny wystąpi jeśli *baza* równa jest 0 a *exp* jest mniejszy lub równy 0.

Błąd dziedziny wystąpi również jeśli *baza* jest ujemna a *exp* nie jest liczbą całkowitą. W przypadku przepełnienia występuje błąd zakresu.

## 3.2. Funkcje standardowe, p3-2a

Obliczenie wartości przy zastosowaniu funkcji `pow(baza,exp)` z `math.h`.

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

```
#include <math.h>
```

```
float main()
```

```
{ double baza, exp;
```

```
    printf("Podaj podstawe i wykladnik (...)\n");
```

```
    scanf("%lf,%lf", &baza, &exp);
```

```
    printf("baza %4.2lf, wykladnik = %4.2lf\n", baza, exp);
```

```
    printf("Wynik baza^wykladnik = %6.2lf\n", pow(baza,exp));
```

```
    system("pause");
```

```
    return 0;
```

```
}
```

```
Podaj podstawe i wykladnik (...)\n2,4\nbaza 2.00, wykladnik = 4.00\nWynik baza^wykladnik = 16.00\nPress any key to continue . . .
```

```
Podaj podstawe i wykladnik (...)\n-1.5,-2\nbaza -1.50, wykladnik = -2.00\nWynik baza^wykladnik = 0.44\nPress any key to continue . . .
```

## 3.2. Funkcje standardowe

### Funkcja `sin( )`

`#include <math.h>`

`double sin( double arg );`      `sin(arg)`

*Opis:* Funkcja `sin()` zwraca wartość sinusa argumentu `arg`, gdzie `arg` podany jest w radianach.

### Funkcja `sqrt( )`

`#include <math.h>`

`double sqrt( double num );`      `num0.5`

*Opis:* Funkcja `sqrt()` zwraca pierwiastek kwadratowy z liczby `num`. Jeśli liczba jest ujemna, występuje błąd dziedziny.



## 3.2. Funkcje standardowe, p3-2b

Obliczenie wartości przy zastosowaniu funkcji  $\sin(\arg)$  z `math.h`,  
`double sin(arg)`, wartość `arg` w radianach.

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

```
#include <math.h>
```

```
float main()
```

```
{ double kat, radian;
```

```
printf("Podaj kat w stopniach \n");
```

```
scanf("%lf,%lf", &kat);
```

```
radian=M_PI*kat/180;
```

```
printf("kat w stopniach = %4.2lf,\nkat w radianach = %4.2lf\n", kat, \radian);
```

```
printf("Wynik: sinus    = %6.2lf\n", sin(radian));
```

```
system("pause");
```

```
return 0;
```

```
}
```

```
Podaj kat w stopniach
```

```
30
```

```
kat w stopniach = 30.00,
```

```
kat w radianach = 0.52
```

```
Wynik: sinus    = 0.50
```

```
Press any key to continue . . .
```

```
Podaj kat w stopniach
```

```
90
```

```
kat w stopniach = 90.00,
```

```
kat w radianach = 1.57
```

```
Wynik: sinus    = 1.00
```

### 3.3. Prototypy funkcji

**Prototypy funkcji** pozwalają kompilatorowi dokładnie sprawdzić czy używane funkcje są poprawnie.

Prototyp funkcji musi pojawić się przed użyciem funkcji (najlepiej umieszczać je na początku programu).

Prototyp funkcji to nagłówek funkcji zakończony średnikiem, np.

**typ\_zwracany nazwa(lista argumentów);**

Funkcja zwraca wartość do miejsca jej wywołania zgodnie z zadeklarowanym typem (np.: **int**, **float**, **char**, **double**).

Funkcja może mieć również typ **void** – wtedy nie zwraca żadnej wartości.

### 3.3. Prototypy funkcji

Lista argumentów, jeśli zawiera identyfikatory zmiennych, to zawsze muszą być poprzedzone informacjami o ich typach, np.:

**int f(int x, int y, float z);** ← **dobrze**

**źle:** **int f(int x, y, float z);**  
brak typu zmiennej **y**

Wersja uproszczona. Podczas definiowania prototypu funkcji nie musimy podawać nazw zmiennych.

**int f(int, int, float);** ← **dobrze**

Lista argumentów może być pusta. W takim przypadku możemy wpisać słowo kluczowe **void** lub zostawić puste miejsce między nawiasami, np.: **float pole(void);** lub **int pr();**

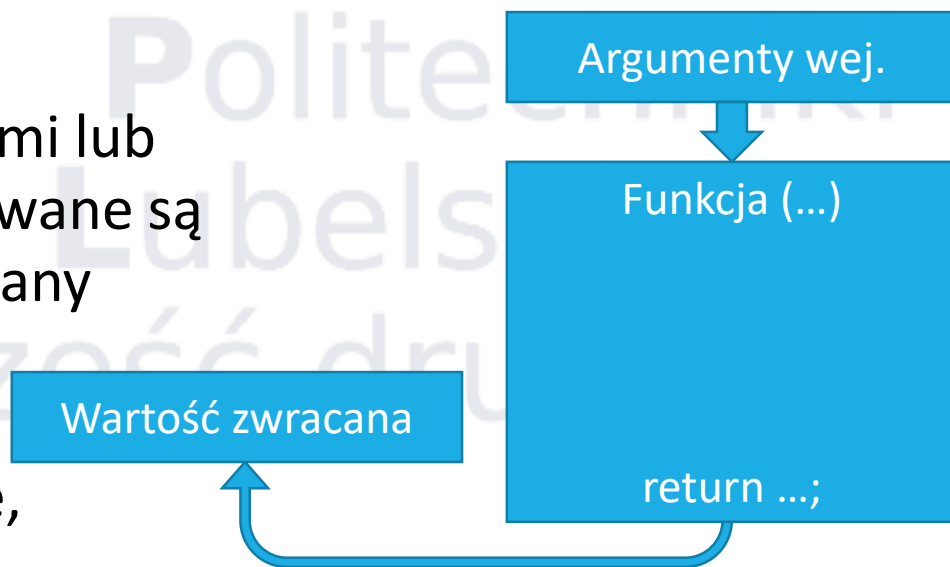
## 3.4. Funkcje własne

Zadaniem funkcji jest przetworzenie przekazanych do niej argumentów i zwrócenie pewnej określonej wartości.

Szkielet funkcji własnej ma postać:

```
typ_zwracany nazwa_funkcji (lista argumentów)  
{ ciało/treść funkcji  
  return zmienna/wyrażenie;  
}
```

Gdy argumenty nie są wskaźnikami lub tablicami to informacje przekazywane są tylko w jednym kierunku. Wywołany program pracuje na przekazanych kopiach wartości. Argumenty nie są modyfikowane, przekazywanie „przez wartość”.



## 3.4. Funkcje własne, instrukcja return

Instrukcja **return**:

- nie występuje w funkcjach typu **void**,
- zwraca na zewnątrz tylko jedną wartość,
- wartość może być podana
  - bezpośrednio (np. 0 , 5.4),
  - w postaci nazwy zmiennej (np. pole, produkt),
  - makra (np. NULL),
  - wyrażenia (np.  $a*b$ ),
  - wartości zwracanej przez inną funkcję,
- zapewnia wyjście z funkcji (powrót do miejsca wywołania funkcji) – to działanie realizowane jest również domyślnie,
- tylko jedna funkcja return może być aktywna, zależy to jaką ścieżkę działania zostanie zrealizowana przez instrukcje wyboru.

## 3.4. Funkcje własne

Pisanie programów z funkcjami własnymi powinno przebiegać następująco:

- zdefiniować prototypy funkcji (określić jej typ zwracany, listę argumentów – ich typy),
- prototypy umieszczamy po poleceniach preprocesora, na zewnątrz ciała funkcji `main()`,
- zdefiniować ciało funkcji `main()`,
- właściwie wywołać funkcje własne (typy argumentów muszą być zgodne z deklaracjami zastosowanymi w prototypach),
- zdefiniować kolejno funkcje własne; funkcje własne definiowane są na zewnątrz ciała funkcji `main()` oraz na zewnątrz innych funkcji własnych.

## 3.4. Funkcje własne, p3-4a

Program z jedną funkcją własną typu void

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

```
void hello(void);
```

*Podano prototyp funkcji*

```
int main ( ) {
```

```
/* Aktywacja/wywołanie funkcji "hello()". */
```

```
hello ();
```

```
System("pause,,");
```

```
return 0; } /* -- koniec main -- */
```

```
void hello ()
```

```
{ printf ("\nKatedra Informatyki\n");
```

*Drukowanie, bez podania kodu formatującego*

```
printf ("Rok akademicki 2019/2020\n");
```

```
}
```

*Pusta lista argumentów  
i brak zwracania  
czegokolwiek,  
słowo kluczowe*

**void** (ang. puste, nic)

```
Katedra Informatyki
Rok akademicki 2019/2020
Press any key to continue . . . _
```



## 3.4. Funkcje własne, p3-4b

Program, który składa się z trzech funkcji:

- funkcji main(),
- funkcji drukującej napis,
- funkcji do obliczenia wyrażenia.

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

```
void inscription(); // prototypy funkcji
```

```
int result(int);
```

*Funkcja **result** ma  
argument typu całkowitego*

```
int main(void)
```

```
{ int number;
```

```
    inscription (); // wywołanie funkcji
```

```
    printf("Enter number ");
```

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

```
    printf("Wynik wynosi %d\n", result(number));
```

```
    system("PAUSE");
```

```
    return 0;
```

```
}
```

*Wywołanie funkcji  
wewnątrz funkcji printf*

## 3.4. Funkcje własne, p3-4b

**void** inscription()

```
{ printf("Prosty program z funkcjami własnymi ");  
  printf("w C\n");  
}
```

*Deklaracja zmiennej num  
znajduje się w polu argumentów*

**int** result(int num)

```
{ return 2*num;  
}
```

*Zwrot wartości wyrażenia*

**Wersja 2:** int res2;  
          res2=2\*num;  
          return res2;

```
Prosty program z funkcjami własnymi w C  
Enter number 7  
Wynik wynosi 14  
Press any key to continue . . .
```

## 3.4. Funkcje własne, p3-4c

**Program, który wywołuje funkcje w celu:**

- rysowania linii oddzielających części programu,
- obliczania sumy dwóch różnych liczb,
- drukowania obliczonej wartości.

Wartości zostały wprowadzone z klawiatury w funkcji main().

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

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

```
void Line1(); //prototypy funkcji
```

```
void Line2();
```

```
int sum(int, int);
```

```
void druk(int);
```

## 3.4. Funkcje własne, p3-4c

```
int main()
{
    int a1, b1, s1;
    Line2(); // wywołanie funkcji Line2
    printf("Podaj 2. liczby int (.../...)\n");
    scanf("%d %d", &a1, &b1);
    printf("a1= %d \t b1= %d \n", a1, b1);
    Line1(); // wywołanie funkcji Line1
    s1=sum(a1,b1); // wywołanie funkcji sum
    printf("s1= %d \n", s1);
    Line2();
    printf("%d \t %d \n", a1, b1);
    Line1();
    system("pause");
    return 0;
}
```

```
void Line1()
{ printf("+++++\n"); }
void Line2()
{ printf("----- \n"); }
int sum(int a, int b)
{
    int sum1;
    sum1=a+b;
    a=a+2; b=b+2;
    Line2(); // wywołanie
    printf("a= %d,\tb= %d\n", a,b);
    Line2(); // wywołanie
    return sum1;
}
void druk(int ww)
{ printf("wynik sumy %d\n", ww); }
```

## 3.4. Funkcje własne, p3-4c

W prezentowanym przykładzie pokazano, że zmiana wartości argumentów w funkcji **sum()** (nowe wartości a=5, b=10) nie powoduje zmiany argumentów po powrocie do funkcji **main()**.

Wypisane wartości nadal miały wartości (3 i 8) – takie jak wprowadzone z klawiatury.

```
-----
Podaj 2. liczby int (.../...)
3,8
a1= 3      b1= 8
+ + + + + + + + + +
-----
a= 5,      b= 10
-----
wynik sumy 11
-----
3          8
+ + + + + + + + + +
Press any key to continue . . . _
```

*wartości zmieniono  
w funkcji **sum()***

*wartości pozostają  
niezmienione  
w funkcji **main()***

## 3.4. Funkcje własne, p3-4d

**Program, w którym wywołanie funkcji z argumentem nastąpi w instrukcji 'return'.**

**Program wywołuje funkcje w celu:**

- rysowania linii oddzielających części programu,
- obliczania sumy dwóch różnych liczb,
- drukowania obliczonej wartości.

Wartości zostany wprowadzone z klawiatury w funkcji main().

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

```
int multi(int); // prototypy funkcji
```

```
int result(int);
```

```
int main(void);
```

### 3.4. Funkcje własne, p3-4d

```
{ int number;  
  printf("Enter number ");  
  scanf("%d", &number);  
  printf("Wynik wynosi %d\n", result(number));  
  system("PAUSE");  
  return 0;  
}
```

*wywołanie funkcji **result()***

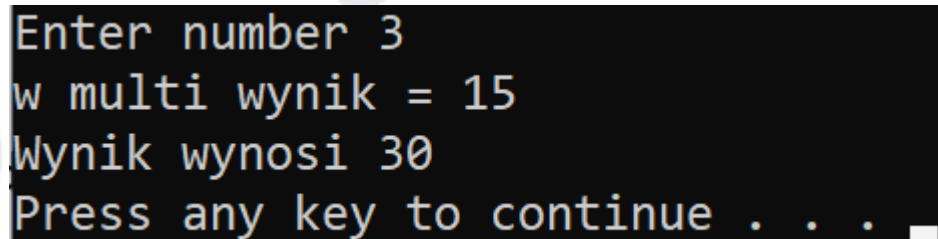
```
int result(int num)
```

```
{ return multi(num)*2; }
```

```
int multi(int a)
```

*wywołanie funkcji  
**multi()***

```
{ int b;  
  b = a*a+2*a;  
  printf("w multi wynik = %d \n", b);  
  return b;  
}
```



```
Enter number 3  
w multi wynik = 15  
Wynik wynosi 30  
Press any key to continue . . . _
```

Materiały zostały opracowane w ramach projektu  
*„Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga”*,  
umowa nr **POWR.03.05.00-00-Z060/18-00**  
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020  
współfinansowanego ze środków Europejskiego Funduszu Społecznego



NAZWA PRZEDMIOTU  
**Programowanie strukturalne**

**Temat wykładu 4.**

**Programowanie z instrukcjami strukturyzującymi: IF, IF ... ELSE, SWITCH.  
Konwersja typów i rzutowanie.**

dr hab. inż. Jerzy Montusiewicz, prof. PL

## 4. Agenda

---

- 4.1. Instrukcja strukturyzująca: IF.**
- 4.2. Instrukcja strukturyzująca: IF...ELSE.**
- 4.3. Instrukcja strukturyzująca: IF...ELSE IF.**
- 4.4. Przełączniki SWITCH i wybór CASE.**
- 4.5. Konwersja typów i rzutowanie.**

## 4.1. Instrukcja strukturyzująca: **if**

Instrukcja **if** należy do grupy instrukcji strukturalizujących program. Określa kolejność wykonywania pewnych operacji.

Zapis instrukcji **if (warunek) instrukcja;**

lub

```
if (warunek)  
{instrukcja1;  
  instrukcja2;  
  ...  
}
```

Spełnienie **warunku** (logiczna prawda, true) powoduje wykonywanie instrukcji lub bloku instrukcji, w przeciwnym razie nastąpi pominięcie.

**Czy te zapisy oznaczają to samo?**

```
if (warunek)      if (warunek);  
instrukcja1;      instrukcja2;
```

**NIE**, w drugim przypadku **instrukcja2** zawsze będzie wykonywana. Warunek dotyczy tylko realizacji instrukcji pustej (przed pierwszym średnikiem).

## 4.1. Instrukcja strukturyzująca: **if**

W wyrażeniach warunkowych stosujemy operatory relacji.

**Relacje binarne** (dwuargumentowe):

- ==** operator sprawdzający równość (**pamiętaj! dwa znaki "="**);
- <** czy pierwszy mniejszy od drugiego;
- >** czy pierwszy większy od drugiego;
- <=** czy pierwszy mniejszy lub równy od drugiego;
- >=** czy pierwszy większy lub równy od drugiego;
- !=** operator sprawdzający nierówność (czy są różne).

## 4.1. Instrukcja strukturyzująca: **if**, p4-1a1

Porównaj wartości dwóch liczby.

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

```
int main ()
```

```
{ int a, b;
```

```
  a = 10; b = 11;
```

```
  printf ("a = %d; b = %d\n", a, b);
```

```
  if(a < b) printf("a=%d jest mniejsze niz b=%d\n", a, b);
```

```
  b = 10;
```

```
  if (a == b)
```

```
  {   printf ("a = %d; b = %d\n", a, b);
```

```
      printf ("a jest rowne b\n");   }
```

```
  system("pause");
```

```
  return 0;
```

```
}
```

```
a = 10; b = 11
a=10 jest mniejsze niz b=11
a = 10; b = 10
a jest rowne b
Press any key to continue . . .
```

## 4.1. Instrukcja strukturyzująca: **if**, p4-1b

### Konwersja podstawy systemu liczbowego.

Zamieniamy liczby w systemie dziesiętnym na liczby w systemie szesnastkowym i na odwrót. System szesnastkowy posiada 16 znaków: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f.

Liczba  $234_{(16)} = 2 \cdot 16^2 + 3 \cdot 16^1 + 4 \cdot 16^0 = 2 \cdot 256 + 3 \cdot 16 + 4 \cdot 1 = 564$

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

```
int main()
```

```
{ int wybor, wartosc;
```

```
    printf("Konwersja:\n");
```

```
    printf("    1: dziesiętne na szesnastkowe\n");
```

```
    printf("    2: szesnastkowe na dziesiętne\n");
```

```
        printf("Wprowadz swój wybór: ");
```

```
        scanf("%d", &wybor); /* adres "wybor" ! */
```

## 4.1. Instrukcja strukturyzująca: **if**, p4-1b

```
if (wybor == 1)
```

```
{ printf("Wprowadz wartosc dziesietna: ");  
  scanf("%d", &wartosc);  
  printf("%d w układzie szesnastkowym jest rowna: %#x\n", wartosc, wartosc);  
}
```

***%x , %X , %#x , %#X –  
kody formatujące  
w systemie szesnastkowym***

```
if (wybor == 2)
```

```
{ printf("Wprowadz wartosc szesnastkowa: ");  
  scanf("%X", &wartosc);  
  printf("%#X w układzie dziesietnym jest rowna: %d\n", wartosc, wartosc);  
}
```

```
system("pause");
```

```
return 0;
```

```
} /*- koniec main -*/
```

Konwersja:

1: dziesietne na szesnastkowe

2: szesnastkowe na dziesietne

Wprowadz swój wybor: 1

Wprowadz wartosc dziesietna: 14

14 w układzie szesnastkowym jest rowna: 0xe

Press any key to continue . . .

## 4.2. Instrukcja strukturyzująca: **if...else**, p4-2a

Ogólniejsza postać instrukcji warunkowej:

**if** (warunek) instrukcja1;

**else** instrukcja2;

Ta postać instrukcji posiada dwie frazy: **if** oraz **else**.

Fraza **else** jest opcjonalna, gdy istnieje, a warunek nie był spełniony (fałsz, false), wykonywana jest **instrukcja2**.

Konstrukcja **if...else** jest bardziej czytelna

```
int wartosc;
```

```
printf("Wprowadz wartosc: ");
```

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

```
if (wartosc < 0) printf("liczba %d jest ujemna\n", wartosc);
```

```
else                printf("liczba %d nie jest ujemna\n", wartosc);
```

```
if(wartosc >= 0)
```

```
Wprowadz wartosc: 0  
liczba 0 nie jest ujemna
```



## 4.2. Instrukcja strukturyzująca: **if...else**, p4-2b

Złożona postać warunku:

Zastosowanie logicznego iloczynu **&&** oraz logicznej sumy **||**.

```
int war;  
printf("Wprowadz wartosc: ");  
scanf("%d", &war); /* adres "wartosc" ! */  
if (war > 10 && war < 30)  
    printf("liczba %d z przedzialu (10 , 30)\n ", war);  
else    printf("liczba %d spoza przedzialu (10 , 30)\n ", war);  
system("pause");  
return 0;  
}
```

```
Wprowadz wartosc: 25  
liczba 25 z przedzialu (10 , 30)  
Press any key to continue . . .
```

## 4.2. Instrukcja strukturyzująca: **if...else**, p4-2c

Fraza **else** odnosi się zawsze do ostatniego **if**, które nie znajduje się w bloku innej frazy.

```
printf(" Bez grupowania w nawiasach\n");
printf(" else zostanie wybrane gdy b == 0.\n");
if (a)    printf("\t a != 0 \n");
if (b)    printf(" drugie if: b != 0 \n\n");
else      printf("else: b == 0 \n\n");
printf(" Grupowanie w nawiasach\n");
printf(" else zostanie wybrane gdy a == 0.\n");
if (a) { if (b) printf(" drugie if: b != 0 \n"); }
else    printf("else: a == 0 \n");
```

Co oznacza ten zapis **if(b)** ?  
Skrótowny zapis warunku **if(b != 0)**.

```
Wprowadz dwie liczby
0 5
Bez grupowania w nawiasach
else zostanie wybrane gdy b == 0.
drugie if: b != 0

Grupowanie w nawiasach
else zostanie wybrane gdy a == 0.
else: a == 0
Press any key to continue . . .
```

```
Wprowadz dwie liczby
0 0
Bez grupowania w nawiasach
else zostanie wybrane gdy b == 0.
else: b == 0

Grupowanie w nawiasach
else zostanie wybrane gdy a == 0.
else: a == 0
Press any key to continue . . .
```

## 4.3. Instrukcja strukturyzująca: **if...else if**, p4-3a

We frazie **else** może występować instrukcja warunkowa **if**. Powstaje tzw. drabinka **if – else – if** opisująca czytelnie wykluczające się nawzajem wybory.

Zamieniamy liczby w systemie dziesiętnym na liczby w systemie ósemkowym i na odwrót. System ósemkowy posiada 8 znaków: 0, 1, 2, 3, 4, 5, 6, 7.

$$\text{Liczba } 234_{(8)} = 2 \cdot 8^2 + 3 \cdot 8^1 + 4 \cdot 8^0 = 2 \cdot 64 + 3 \cdot 8 + 4 \cdot 1 = 156$$

## 4.3. Instrukcja strukturyzująca: **if...else if**, p4-3a

**else if (wybor == 3)**

```
{ printf("Wprowadz wartosc dziesietna: ");  
  scanf("%d", &wartosc);  
  printf("%d w układzie osemkowym jest rowna: %o\n", wartosc, wartosc);  
}
```

*%o – kod formatujący  
w systemie ósemkowym*

**else if (wybor == 4)**

```
{ printf("Wprowadz wartosc osemkowa: ");  
  scanf("%o", &wartosc);  
  printf("%o w układzie dziesietnym jest rowna: %d\n", wartosc, wartosc);  
}
```

**else** printf(" Popelniles blad.\n Wartości z przedziału <3 , 4> \n");

```
Wprowadz swoj wybor: 4  
Wprowadz wartosc osemkowa:  234  
234 w układzie dziesietnym jest rowna: 156
```

## 4.4. Przełączniki **switch** i wybór **case**

Instrukcja przełącznika **switch** jest wieloodgałęziową instrukcją decyzyjną funkcjonalnie podobną do wyboru **if – else – if**.

**switch** (zmienna)

```
{  
    case stała1:  
        instrukcje;  
        break;  
    case stała2:  
        instrukcje;  
        break;  
    ...  
    ...  
    default:  
        instrukcje;  
}
```

*Zmienna jednego z typów całkowitych i char*

**case** instrukcja porównująca tylko równość wartości tej zmiennej ze swoim argumentem, czyli **stała1**

Po stwierdzeniu równości wykonywane są kolejne instrukcje, także należące do innych instrukcji **case**, aż do napotkania instrukcji **break**.

**break** powoduje przeskok do końca zakresu przełącznika do miejsca po nawiasie **}**.

**default** przypadek domyślny (opcjonalny) obsługuje przypadek gdy zmienna nie jest równa żadnemu wzorcowi typu **stałax**.

## 4.4. Przełączniki **switch** i wybór **case**, p4-4a

Przełącznik **switch**; funkcjonalność zbliżona do drabinki if-else-if.  
Konwersja podstawy systemów liczbowych.

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

```
int main()
```

```
{    char wybor;  
    int wartosc;  
    printf("Konwersja:\n");  
    printf("  1: dziesiętne na szesnastkowe\n");  
    printf("  2: szesnastkowe na dziesiętne\n");  
    printf("  3: dziesiętne na osemkowe\n");  
    printf("  4: osemkowe na dziesiętne\n");  
    printf("Wprowadz swój wybór: ");  
    scanf("%d", &wybor);
```

## 4.4. Przełączniki **switch** i wybór **case**, p4-4a

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

```
switch (wybor) {
```

```
    case 1:
```

```
    printf("Wprowadz wartosc dziesietna: ");
```

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

```
    printf("%d w układzie szesnastkowym =: %#X\n", wartosc, wartosc);
```

```
    // break;
```

```
    case 2:
```

```
    printf("Wprowadz wartosc szesnastkowa: ");
```

```
    scanf("%x", &wartosc);
```

```
    printf("%#x w układzie dziesiętnym =: %d\n", wartosc, wartosc);
```

```
    break;
```

***case 1** to przypadek równości zmiennej **wybor**, wprowadzonej przez **scanf** i odpowiedniej stałej (argumentu instr. **case**)*

*Wykonanie instrukcji do napotkanego **break**.*



## 4.4. Przełączniki **switch** i wybór **case**, p4-4a

### **case 3:**

```
printf("Wprowadz wartosc dziesiętna: ");  
scanf("%d", &wartosc);  
printf("%d w układzie osemkowym =: %o\n", wartosc, wartosc);  
break;
```

### **case 4:**

```
printf("Wprowadz wartosc osemkowa: ");  
scanf("%o", &wartosc);  
printf("%o w układzie dziesiętnym =: %d\n", wartosc, wartosc);  
break;
```

### **default:**

```
printf(" Popelniles blad.\n Wartosc z przedziału od 1 do 4. \n");  
} // -- koniec switch --  
} /* --- koniec main() --- */
```



## 4.4. Przełączniki **switch** i wybór **case**, p4-4a

We frazie **case 1** nie było instrukcji **break**.

Konwersja:

- 1: dziesiętne na szesnastkowe
- 2: szesnastkowe na dziesiętne
- 3: dziesiętne na osemkowe
- 4: osemkowe na dziesiętne

Wprowadz swój wybór: 1

Wprowadz wartość dziesiętną: 100

100 w układzie szesnastkowym jest równa: 0X64

Wprowadz wartość szesnastkową: de

0xde w układzie dziesiętnym jest równa: 222

Program więc przeszedł do realizacji frazy **case 2**.

Do wyprowadzenia wartości w systemie szesnastkowym zastosowano odpowiednio kody: **%#X** oraz **%#x**, stąd wypisanie **0X** lub **0x** oraz znaków systemu szesnastkowego (małe litery, bo małe x).

## 4.4. Przełączniki **switch** i wybór **case**, p4-4b

**Generowanie stałych do wyboru **case** przez pętle 'for'.**

Drukowanie i multiplikowanie wersetów wiersza. Gdy brakuje instrukcji **break** – przejście do następnego przypadku **case**, pomimo braku równości.

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

```
int main()
```

```
{ int run;
```

```
  for (run = 0; run < 7; run++)
```

```
    switch (run)
```

```
    { case 1: printf("Wisława Szymborska: \"Rozmowa z kamieniem\"\n");
```

```
        break;
```

```
        case 2: printf("Pukalam do drzwi kamienia\n");
```

```
            break;
```

```
        case 3:
```

*Drukowanie znaku " wymaga wpisania kodu \".*

## 4.4. Przełączniki **switch** i wybór **case**, p4-4b

```
case 4: printf("- To ja,");  
        printf(" wpusc mnie\n");  
        printf("Chce wejsc do twego wnetrza\n");  
        break;  
case 5: printf("rozejrzec sie dookola.\n");  
case 6: printf("nabrac ciebie jak tchu.\n");  
} // -- koniec switch  
}
```

```
Wisława Szymborska: "Rozmowa z kamieniem"  
Pukalam do drzwi kamienia  
- To ja, wpusc mnie  
Chce wejsc do twego wnetrza  
- To ja, wpusc mnie  
Chce wejsc do twego wnetrza  
rozejrzec sie dookola.  
nabrac ciebie jak tchu.  
nabrac ciebie jak tchu.
```

## 4.5. Konwersja typów i rzutowanie

Konwersja jest dokonywana gdy występują zmienne różnych typów. Polega na tym, że zmienne typu mniej pojemnego są przekształcane w zmienne o większej pojemności., np.:

**char** i **short** promowane są do typu **int**,

**int** do typu **long**,

**float** do typu **double**.

Gdy operator (np. **\***, **/**) wiąże z sobą zmienne **int** i **float (double)** to następuje konwersja **int** w odpowiedni typ **float (double)** [działanie lokalne], np.

**1/3. albo 1./3 → rezultatem jest liczba zmiennoprzecinkowa**

W operacji podstawienia może wystąpić **konwersja odwrotna**, jeśli zmienna docelowa jest mniej pojemna niż rezultat wyrażenia.

**Utrata części informacji.**

## 4.5. Konwersja typów i rzutowanie, p4-5a

**Konwersja przez operator:** iloczyn, iloraz, sumę i różnicę. Konwersja odwrotna.

```
int main()
{ int in1=1, in2=2, win3;
  float fl1=3.5, fl2=2.7, fl3;
  printf("Konwersja zmiennych\n");
  printf("iloczyn int i float = %5.2f\n", in1*fl1);
  printf("iloraz int i float = %5.2f\n", in1/fl1);
  printf("suma int i float = %5.2f\n", in1+fl1);
  printf("roznica int i float = %5.2f\n", in1-fl1);
  printf("\n");
  printf("Konwersja odwrotna\n");
  win3=fl1;
  printf("wypisanie float = %5.2f\n", fl1);
  printf("na int podstawiono float = %d\n", win3);
  return 0;
}
```

```
Konwersja zmiennych
iloczyn int i float = 3.50
iloraz int i float = 0.29
suma int i float = 4.50
roznica int i float = -2.50
```

```
Konwersja odwrotna
wypisanie float = 3.50
na int podstawiono float = 3
```

## 4.5. Konwersja typów i rzutowanie

Kontrolowanie konwersji typów, jej charakteru i kolejności, np.

**temp/3**

Arytmetyka w zbiorze liczb całkowitych daje wynik w tym zbiorze, np.

**temp=7, temp/3 → 2**

Gdy jedna z wartości będzie w zbiorze liczb rzeczywistych to wynik będzie równie w tym zbiorze:

**temp=7, temp/3. → 2.33**

**1.0\*temp/3 → 2.33**

*3 zamieniono na 3.*

*Dodana kropkę dziesiętną.*

*temp zamieniono na wartość rzeczywistą, pomnożono przez 1.0.*

**Konwersja jawna**

**(float)temp/3**

*Jednokrotna zamiana **temp** na wartość rzeczywistą.*

*To działanie wymusza promocję (konwersję automatyczną) liczby **7** na **float**.*

*Po konwersji wykonywane jest dzielenie.*

## 4.5. Konwersja typów i rzutowanie, p4-5b

Czas konwersji, a uzyskany rezultat.

```
int jed=7, dwa=2; double ratio, sum=5.0;
ratio = jed/dwa;
printf("%d / %d (arytmetyka int) \t\t\t= %5.2lf\n", jed, dwa, ratio);
ratio = sum + jed/dwa;
printf("%d / %d (arytmetyka int) + sum \t\t= %5.2lf\n", jed, dwa, ratio);
ratio = jed/(double)dwa;
printf("(konwersja na float 2.zmiennej) %d / %d = %5.2lf\n", jed, dwa, ratio);
ratio = (double)(jed/dwa);
printf("%d / %d (arytmetyka int, potem konwersja) = %5.2lf", jed, dwa, ratio);
```

```
7 / 2 (arytmetyka int)                = 3.00
7 / 2 (arytmetyka int) + sum          = 8.00
(konwersja na float 2.zmiennej) 7 / 2 = 3.50
7 / 2 (arytmetyka int, potem konwersja) = 3.00
```



Materiały zostały opracowane w ramach projektu  
*„Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga”*,  
umowa nr **POWR.03.05.00-00-Z060/18-00**  
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020  
współfinansowanego ze środków Europejskiego Funduszu Społecznego



NAZWA PRZEDMIOTU  
**Programowanie strukturalne**

**Temat wykładu 5.**  
**Programowanie iteracyjne z WHILE, DO ... WHILE, FOR.**  
**Instrukcje skoków.**

dr hab. inż. Jerzy Montusiewicz, prof. PL

## 5. Agenda

---

5.1. Pętla typu **while**.

5.2. Pętla typu **do...while**.

5.3. Pętla typu **for**.

5.4. Instrukcje skoków **continue**, **break**, **go to**.

## 5.1. Pętla typu **while**

Pętle to instrukcje iteracyjne służące do względnie jednolitego przetwarzania zbiorów jednorodnych obiektów lub realizacji powtarzających się w zbliżonej postaci operacji. W języku C mamy trzy rodzaje pętli: **while**, **do...while**, **for**.

W pętli **while** znamy warunek zakończenia obieganania, ale trudno wskazać jakąś zmienną kontrolną lub określić liczbę jej obiegow.

**while** (**warunek**) instrukcja;

**while** (**warunek**) {instrukcja1; instrukcja2; ...}

Charakterystyka:

- warunek zakończenia obieganania pętli sprawdzany jest na początku pętli (podobnie jak w pętli **for**),
- wewnątrz pętli wykonywane są operacje, które w kolejnym obiegu doprowadzą do zakończenia procesu obieganania.

## 5.1. Pętla typu **while**

**warunek** – zbudowany jest z relacji binarnych (**==**, **>**, **<**, **<=**, **>=**, **!=**), a także z iloczynu logicznego (**&&**) lub sumy logicznej (**||**) tych relacji.

**blok** – zawiera instrukcje podstawienia, wyrażenia, wywołania **instrukcji** funkcji do wprowadzania i wyprowadzania danych, funkcji własnych. Instrukcje są ograniczone nawiasami klamrowymi **{ }**, po których nie stawia się średnika.

## 5.1. Pętla typu **while**, p5-1a

**Wypisz zestawienie temperatur Fahrenheita-Celsjusza dla wartości = 0, 40, ..., 280.**

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

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

```
int main() { Zmienne całkowite, zmiennoprzecinkowe
```

```
    int dolna, gorna, krok;
```

```
    float fahr, celsius; Nadanie wartości początkowych
```

```
    dolna = 0; /* dolna granica temperatur */
```

```
    gorna = 280; /* gorna granica */
```

```
    krok = 40; /* rozmiar kroku */
```

```
    fahr = dolna;
```

```
    printf("Przeliczenie stopni \nFahrenheita na Celsjusza\n");
```

```
    while (fahr <= gorna) Arytmetyka float: 5.0
```

```
    { celsius = (5.0/9) * (fahr - 32.0);
```

```
      printf("%6.1f\t %6.2f\n", fahr, celsius);
```

```
      fahr = fahr + krok; } Kod formatujący %f
```

```
    system("pause"); definicja liczby pól:
```

```
    return(0); } 6.1 6.2
```

```
Przeliczenie stopni
Fahrenheita na Celsjusza
  0.0   -17.78
 40.0    4.44
 80.0   26.67
120.0   48.89
160.0   71.11
200.0   93.33
240.0  115.56
280.0  137.78
Press any key to continue
```

## 5.1. Pętla typu **while**, p5-1b1

Przerwanie działania programu po wybraniu klawisza A.

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

```
void czekaj_na_char(void);
```

```
int main() {
```

```
printf("W main przed 'czekaj na char()'\n");
```

```
czekaj_na_char ();
```

```
printf ("W main po wyjściu z 'czekaj na char()'\n");
```

```
system(„pause”);
```

```
return 0; }
```

```
void czekaj_na_char()
```

```
{ char ch;
```

```
ch = '\0'; Należało zainicjować wartość ch.
```

```
printf ("Teraz w 'czekaj na char()'\n");
```

```
printf (" Wyjście po naciśnięciu 'A'\n");
```

```
printf (" Wpisz znak i naciśnij ENTER\n");
```

```
while (ch != 'A') { ch = getchar(); } Warunek sprawdzany na wejściu do pętli.
```

```
printf ("Wyjście: naciśnoles klawisz A\n");
```

```
}
```

W main przed 'czekaj na char()'

Teraz w 'czekaj na char()'

Wyjście po naciśnięciu 'A'

Wpisz znak i naciśnij ENTER

2

&

a

A

Wyjście: naciśnoles klawisz A

W main po wyjściu z 'czekaj na char()'

Press any key to continue . . .

*getchar() – do czytania 1. znaku*

*Gdy ch różne od A to wykonywana jest instrukcja w { }.*

## 5.1. Pętla typu **while**, p5-1b2

**Modyfikacja poprzedniego programu.** Wstawienie instrukcji przypisania wewnątrz pętli *while* i powiązanie z operatorem relacyjnym.

```
void czekaj_na_char()
{ char ch;
  // ch = '\0';
  printf ("Teraz w 'czekaj na char()'\n");
  printf (" Wyjście po naciśnięciu 'A'\n");
  printf (" Wpisz znak i naciśnij ENTER\n");
  // Brak ciała w klamrach { } w tej petli
  while ((ch= getchar()) != 'A');
  printf ("wyjście: naciśnoles klawisz A\n");
}
```

```
W main przed 'czekaj na char()'
Teraz w 'czekaj na char()'
Wyjście po naciśnięciu 'A'
Wpisz znak i naciśnij ENTER
e
)
a
A
wyjście: naciśnoles klawisz A
W main po wyjściu z 'czekaj na char()'
Press any key to continue . . .
```

## 5.1. Pętla typu **while**, p5-1c

### Utwórz wyśrodkowany napis.

Sterowanie pętlą **while** przy użyciu zmiennej kontrolnej. Inicjalizacja zmiennej kontrolnej na zewnątrz pętli, jawna modyfikacja zmiennej wewnątrz instrukcji.

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

```
#include <string.h>
```

**string.h** – dodatkowa biblioteka.

```
void center (int len);
```

```
int main()
```

```
{ char str[80];    str – tablica znakowa.
```

```
  int len;
```

```
  printf ("Wprowadz lancuch: \n\n");
```

```
  gets (str);    gets (str) – wczytywanie napisów (łańcuchów znakowych).
```

```
  printf ("Ponizej wysrodkowany napis \n\n");
```

```
  center(strlen(str));
```

```
  printf (str);
```

**strlen (str)** – określenie długości napisu  
(*netto, bez terminującego bajtu zerowego*).

```
  printf ("\n");
```

```
}
```



## 5.1. Pętla typu **while**, p5-1c

**void center (int len)**      *len - zmienna kontrolna.*

```
{  
    // Napis wysrodkowany  
    len = (80-len)/2;  
    while (len > 0)  
    {  
        printf (" ");  
        len--;  
    }  
}
```

*Wyliczenie liczby pustych znaków po lewej stronie na podstawie długości wprowadzonego łańcucha gets (str) w main().*

*Drukowanie pustych znaków po lewej.*

*Dekrementacja o 1*

Wprowadz lancuch:

Katedra Informatyki

Ponizej wysrodkowany napis

Katedra Informatyki

-----

## 5.2. Pętla typu **do...while**

Pętla **do – while** ma następującą strukturę:

```
do  
  { instrukcja/e  
    instrukcja/e;  
  } while (warunek);
```

Charakterystyka:

- pętla **do...while** będzie obiegana przynajmniej jeden raz,
- warunek obiegania i zakończenia jest sprawdzany za pomocą frazy **while** na końcu pętli,
- liczba obiegów pętli nie jest określona z góry,
- na ogół nie ma zmiennej kontrolnej,
- o liczbie obiegów pętli decyduje wynik warunku,
- warunek zbudowany jest jak w pętli **while**.

## 5.2. Pętla typu **do...while**, p5-2a

**Wprowadzanie liczb większych niż 100.**

Wyjście z programu gdy podana wartość jest mniejsza niż 100.

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

```
int main()
```

```
{
```

```
    int number=0;
```

```
    do
```

```
    {
```

```
        printf("Wprowadz liczbe: ");
```

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

```
        printf ("number = %d\n", number);
```

```
    } while (number >= 100);
```

```
    printf("\nWpisales liczbe %d < 100\n", number);
```

```
}
```

```
Wprowadz liczbe: 123
```

```
number = 123
```

```
Wprowadz liczbe: 432
```

```
number = 432
```

```
Wprowadz liczbe: 12
```

```
number = 12
```

```
Wpisales liczbe 12 < 100
```

## 5.3. Pętla typu **for**

Instrukcja iteracyjna (pętli) **for** pozwala na wykonywanie fragmentu programu dopóki wartość wyrażenia wpisanego w warunek jest różna od zera, co znaczy, że wyrażenie jest prawdziwe. Pętlę **for** stosujemy gdy znana jest liczba obiegów pętli.

```
for (inicjalizacja; warunek; przyrost) instrukcja;  
for (inicjalizacja; warunek; przyrost) {instr1; instr2; ... }
```

- inicjalizacja** – wyrażenie określające wartość początkową,
- warunek** – do jakiej granicznej wartości zastosowanej zmiennej kontrolnej obiegamy pętlę,
- przyrost** – podajemy regułę, jak w kolejnych obiegach pętli należy modyfikować zmienną kontrolną.

W prostych przypadkach zmienna jest zwiększana lub zmniejszana o wartość 1 (inkrementacja / dekrementacja).

## 5.3. Pętla typu **for**

Przyrost nie musi być równy jedności w kolejnych obiegach pętli;  
Przyrost maleje o 1 (dekrementacja):

**for** (run = 10; run >= 0; run--) instrukcja;

Przyrost rośnie o 5:

**for** (run = 0; run <= 100; run=run+5) instrukcja;

wersja uproszczona: run+=5

Zmienna kontrolna (przyrost) jest preinkrementowana:

**for** (run = 0; run <= 100; ++run) instrukcja;

Zmienna kontrolna (przyrost) jest postinkrementowana:

**for** (run = 0; run <= 100; run++) instrukcja;

## 5.3. Pętla typu **for**, p5-3a1

Pętla 'for' w połączeniu z instrukcją 'if'. Operator modulo '%'.  
**Drukuj liczby podzielne przez 3 z przedziału 1 do 42.**

*Operator modulo, '%' wyznaczania resztę z dzielenia dwóch liczb.*

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

```
int main()
```

```
{
```

```
    int num;
```

```
    for (num=1; num<=42; num++)
```

```
        { if (num%3==0) printf ("%d ", num); }
```

```
    printf ("\n");
```

```
    system("pause");
```

```
    return 0;
```

```
}
```

```
3 6 9 12 15 18 21 24 27 30 33 36 39 42
Press any key to continue . . .
```

## 5.3. Pętla typu **for**, p5-3a2

Pętla 'for' w połączeniu z instrukcją 'if'. Operator modulo '%'.  
**Drukuj reszty różne od 0 z dzielenia przez 3 liczb z przedziału 1 do 7**

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

```
int main()
```

```
{
```

```
    int num, reszta;
```

```
    for (num=1; num<=10; num++)
```

```
    { reszta=num%3;
```

```
        if (reszta!=0)
```

```
            printf ("liczba %2d --> reszta %2d\n", num, reszta); }
```

```
    printf ("\n");
```

```
    system("pause");
```

```
    return 0;
```

```
}
```

```
liczba  1 --> reszta  1
liczba  2 --> reszta  2
liczba  4 --> reszta  1
liczba  5 --> reszta  2
liczba  7 --> reszta  1
```

```
Press any key to continue . . . _
```

*Operator modulo, '%' wyznaczania resztę z dzielenia dwóch liczb.*

## 5.3. Pętla typu **for**, p5-3b

Pętla for może mieć więcej zmiennych kontrolnych.

**Wypisz kolejne wartości z przebiegu pętli.**

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

```
int main()
```

*Dwie zmienne kontrolne **ob1** oraz **ob2***

```
{ int ob1, ob2;
```

```
printf("dla petli z dwoma zmiennymi \nkontrolnymi\n");
```

```
for (ob1=0, ob2=0; ob1+ob2 < 15; ++ob1, ++ob2)
```

```
printf("  %d\n", ob1+ob2);
```

*Znak ', ' jest separatorem  
w obszarze pól pętli **for***

```
printf("\n");
```

```
system("pause");
```

```
return 0;
```

```
}
```

```
Petla z dwoma zmiennymi  
kontrolnymi
```

```
6
```

```
8
```

```
10
```

```
12
```

```
14
```

*Przyrost realizowany  
jest na końcu  
każdego obiegu.*



## 5.3. Pętla typu **for**, p5-3c

Wyjście z pętli for przez kontrolowanie innej zmiennej.

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

```
int main()
```

```
{ int ob1;
```

```
char ch1 = ' '; Zmienna typu znakowego ch1.
```

```
printf("Petla z dodatkowym wyjsciem\n");
```

```
printf(" Do zatrzymania nacisnij litera E \n");
```

```
printf(" Klawisz Enter zatwierdza znak\n");
```

```
for (ob1 = 1; ob1<9 && ch1 != 'E'; ob1++)
```

```
{ printf(" %d\n", ob1); && operator
```

```
ch1 = getchar(); iloczynu logicznego. }
```

```
printf ("\n"); Wczytanie znaku
```

```
return 0; z klawiatury.
```

```
}
```

## 5.3. Pętla typu **for**, p5-3c

```
Pętla z dodatkowym wyjściem
Do zatrzymania naciśnij literę E
Klawisz Enter zatwierdza znak

1
w
2
3
^
4
5
t
6
7
y
8
```

```
Do zatrzymania naciśnij literę E
Klawisz Enter zatwierdza znak

1
q
2
3
(
4
5
E
```

## 5.3. Pętla typu **for**, p5-3d

Pętla **for** nie musi mieć zdefiniowanych wszystkich pól: **for ( ; ; )**.  
Wyjście z nieskończonej pętli – instrukcja **break**.

```
int main()
```

```
{ char input;
```

```
printf("Wyjście z nieskończonej petli przez break \n");
```

```
printf("  EXIT: type 'A'\n");
```

```
printf("  Wprowadz znak i naciśnij ENTER\n");
```

```
for ( ; ; ) Pętla nieskończona
```

```
{ input = getchar();
```

```
  if (input == 'A') break; }
```

```
printf ("  Wpisales A\n");
```

```
return 0;
```

```
}
```

*break – przeskok za nawias  
'}' zamykający blok pętli.*

```
Wyjście z nieskończonej petli
przez break
EXIT: type 'A'
Wprowadz znak i naciśnij ENTER
q
3
#
A

Wpisales A
```

## 5.3. Pętla typu **for**, p5-3e

Pętle **for** mogą być zagnieżdżone (we wnętrzu jednej pętli umieszczone są inne pętli). Zagnieżdżenie może być wielokrotne.

Pętla **for** zagnieżdżona. Potęgi (2, 3, 4) liczby od 1 do 9.

```
int main()
{ int i, j, k, temp;
  printf("  i  i^2 i^3 i^4\n\n");
  for (i=1;i<10;i++)
  { for (j=1;j<5;j++)
    { temp = 1;
      for (k=0;k<j;k++)
        temp = temp*i;
      printf("%6d", temp); }
    printf ("\n"); }

  return 0;
}
```

i	i^2	i^3	i^4
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625
6	36	216	1296
7	49	343	2401
8	64	512	4096
9	81	729	6561

## 5.4. Instrukcje skoków: **break**, **continue**, **goto**

Paradygmat programowania strukturalnego nie pozwala stosowania w programach instrukcji niestukturalnych, a właśnie takimi są skoki. Istnieją dwa powody zastosowania instrukcji skoków:

- czasami lepsza czytelność pewnych nielicznych programów,
- konwersja ogromnych bibliotek numerycznych z języka FORTRAN na języki C lub C++ (najłatwiej fortranowskie skoki jest przekonwertować na skoki w języku C).

Instrukcja **break** w przełączniku **switch** była zakończeniem instrukcji w odgałęzieniu **case**. Tak więc **break** wymusza skok poza koniec bloku instrukcji w nawiasach **{ }**.

## 5.4. Instrukcje skoków: **break**, p5-4a

Wyjście z pętli 'for' przez skok 'break', przy wartości równej 8.

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

```
int main()
```

```
{ int pt=1;
```

```
    printf ("Petla for: 1...40;\n break dla == 8\n");
```

```
    for (pt; pt<40; pt++)
```

```
        { if (pt==8) break;
```

```
          printf("%4d\n", pt); }
```

```
    system("pause");
```

```
    return 0;
```

```
}
```

```
Petla for: 1...40;  
break dla == 8
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
Press any key to continue
```

## 5.4. Instrukcje skoków: **continue**, p5-4b

Instrukcja **continue** jest instrukcją skoku do końca bloku, ale nie poza ten blok. Czyli powoduje pominięcie części instrukcji z wnętrza pętli i przejście do następnej iteracji gdy warunek pętli jest spełniony.  
**Skok 'continue' do końca zakresu pętli oraz jej kontynuacja.**

```
int main()
{ // Drukuj tylko wybrane liczby
  int co;
  printf ("Pętla for: 1...17;\n skok \
continue\n dla parzystych liczb\n");
  for (co=0; co<17; co++)
    { if (co%2) continue;
      printf("%4d\n", co); }
  return 0;
}
```

```
Pętla for: 1...17;
skok continue
dla parzystych liczb
 0
 2
 4
 6
 8
10
12
14
16
Press any key to continue
```

## 5.4. Instrukcje skoków: **goto**

Instrukcja **goto** *etykieta* jest najbardziej ogólną instrukcją skoku. Powoduje przeskok do miejsca oznaczonego *etykieta* (adres docelowy) tej instrukcji (jej argument).

```
goto stop  
instrukcja/e
```

```
...
```

```
stop:
```

Etykiety są identyfikatorami zakończone dwukropkiem ':', np.

```
target:
```

```
end:
```

Instrukcja **goto** pozwala na wyskok z głęboko zagnieżdżonych pętli. Należy unikać instrukcji **goto**, ponieważ przy zastosowaniu tej samej etykiety nie jesteśmy w stanie kontrolować miejsca skąd nastąpił skok. Niektórzy uważają tę instrukcję za relik.



## 5.4. Instrukcje skoków: goto, p5-4c

Skok 'goto' do zdefiniowanej etykiety z zagnieżdżonej pętli.

```
#include <stdio.h>
int main()
{ int i, j, k, temp;
  printf(" i  i^2  i^3  i^4\n\n");
  for (i=1;i<10; i++)
  { for (j=1; j<5; j++) { temp = 1;
    for (k=0; k<j; k++)
    { if(i==4) goto stop;
      temp = temp*i;}
    printf("%6d", temp);

    printf ("\n");
  }
  stop:
  printf("\nZakonczono dla i = %d\n", i);
  return 0;
}
```

i	i^2	i^3	i^4
1	1	1	1
2	4	8	16
3	9	27	81
Zakonczono dla i = 4			

Materiały zostały opracowane w ramach projektu  
*„Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga”*,  
umowa nr **POWR.03.05.00-00-Z060/18-00**  
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020  
współfinansowanego ze środków Europejskiego Funduszu Społecznego