

# PODSTAWY PROGRAMOWANIA

## Programowanie strukturalne i proceduralne z przykładami w języku C i C++

( *wykl. dr Marek Piasecki* )

### Literatura:

- do wykładu → **dowolny podręcznik do języka C oraz C++**  
Odsyłacze do rekomendowanych podręczników elektronicznych są podane na [stronie internetowej](#) tego wykładu.

IDE na laboratoriach → **CLion / Falcon C++ / MS Visual Studio / Dev C++**

- **S. Prata** „Szkola programowania, Język C”
- G. Perry, Dean Miller „Język C. Programowanie dla początkujących.”
- **Jerzy Grębosz** “Symfonia C++” tom pierwszy
- **Andrzej Zalewski** “Programowanie w językach C i C++ z wykorzystaniem pakietu Borland C++”
- **S. Lippman** “Podstawy języka C++”
- **K. Jamsa** “Wygraj z C++”

- 
- **Bjarne Stroustrup** “Język C++ “ ← *książka napisana przez twórcę C++*
  - **Robert Sedgewick** “Algorytmy w C ++ “

- 
- **Brian Kernigham, Dennis Ritchie** “Język ANSI C“ ← *trochę historii*

## PROGRAM WYKŁADU

1. Wstęp, **schematy blokowe**, **struktura programu** w języku C/C++  
Typy, operatory i wyrażenia.
2. Operacje wejścia i wyjścia (podejście proceduralne i obiektowe)  
Instrukcje **if**, **if-else**, **switch**. Zagnieżdżanie. Operator ? : .
3. Instrukcje iteracyjne: **while**, **do-while**, **for**.  
Pętle zagnieżdżone. Instrukcje **break** i **continue**.
4. **Tablice** – deklaracja, inicjacja, operator indeksu.  
Tablice w połączeniu z pętlą **for**. Tablice wielowymiarowe.
5. **Wskaźniki** zmiennych, adresy pamięci, arytmetyka wskaźników.  
Związek pomiędzy wskaźnikami a tablicami.
6. **Funkcje** – deklaracja, definicja, przekazywanie parametrów.
7. Funkcje operujące na pamięci: biblioteka <mem.h>  
**Łańcuchy** znaków. Funkcje łańcuchowe <string.h>
8. **Typ strukturalny** – definicja, deklaracja i inicjalizacja zmiennych.  
Zagnieżdżanie struktur. Rozszerzenie struktury o metody składowe.
9. **Obsługa plików** zewnętrznych. Pliki binarne i tekstowe.  
podejście proceduralne – biblioteka <stdio.h>  
podejście obiektowe - klasa fstream
10. **Tablice wskaźników, wskaźniki na tablice**.  
Rzutowanie wskaźników. Dostęp do dowolnego obszaru pamięci.  
Wskaźniki na funkcje.
11. Przykłady różnych kombinacji wskaźników  
**Dynamiczne przydzielanie pamięci**.
12. Rekurencyjne struktury danych  
**Implementacja stosu, kolejki, listy jedno i dwu-kierunkowej**

## PODSTAWOWE POJĘCIA

- Program** – notacja opisująca proces przekształcania *danych wejściowych* w *dane wyjściowe* według pewnego *algorytmu*.
- Dane wejściowe** – informacje dostarczone do programu przez użytkownika, w celu umożliwienia wykonania algorytmu
- Dane wyjściowe** – są generowane przez program i stanowią wyniki działania programu.
- Algorytm** – określa sposób przekształcania danych wejściowych w dane wyjściowe zgodnie z zadaniem. Algorytm składa się z opisu:
- *obiektów* na których wykonywane są działania,
  - *działań* realizujących cel algorytmu,
  - *kolejności* działań.
- Programowanie** – polega na zapisywaniu *algorytmów* w formie *programów* zrozumiałych dla komputera.
- Kod źródłowy** – program napisany w języku takim jak Pascal lub C++, czyli w języku algorytmicznym – czytelny dla programisty,
- Kod wynikowy** – program zapisany jako ciąg rozkazów i danych w kodzie maszynowym procesora (w postaci czytelnej dla komputera), najczęściej w postaci liczb kodu dwójkowego.
- 
- 

### Proces tworzenia ( kodowania? ) programu:

↓ <b>edytor</b>	→	( *.cpp )	<i>kod źródłowy</i>
↓ <b>kompilator</b>	→	( *.obj )	<i>kod wynikowy</i>
↓ <b>linker</b>	→	( *.exe )	<i>kod wynikowy połączony z bibliotekami</i>
↓ <b>debugger</b>	→	(step/watch)	<i>śledzenie działania, usuwanie błędów</i>

---

---

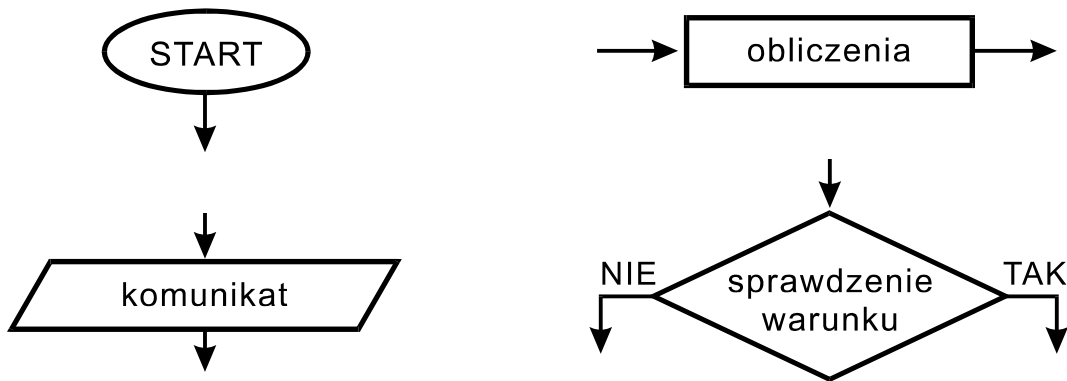
### Język **C++** jest rozszerzeniem języka **C** :

- typy i zmienne referencyjne, unie anonimowe,
- operatory new i delete,
- funkcje przeciążone, funkcje z atrybutem inline,
- domyślne wartości parametrów funkcji,
- przekazywanie parametrów funkcji przez referencję,
- **klasy i obiekty (programowanie obiektowe)**
- wzorce
- obsługa wyjątków

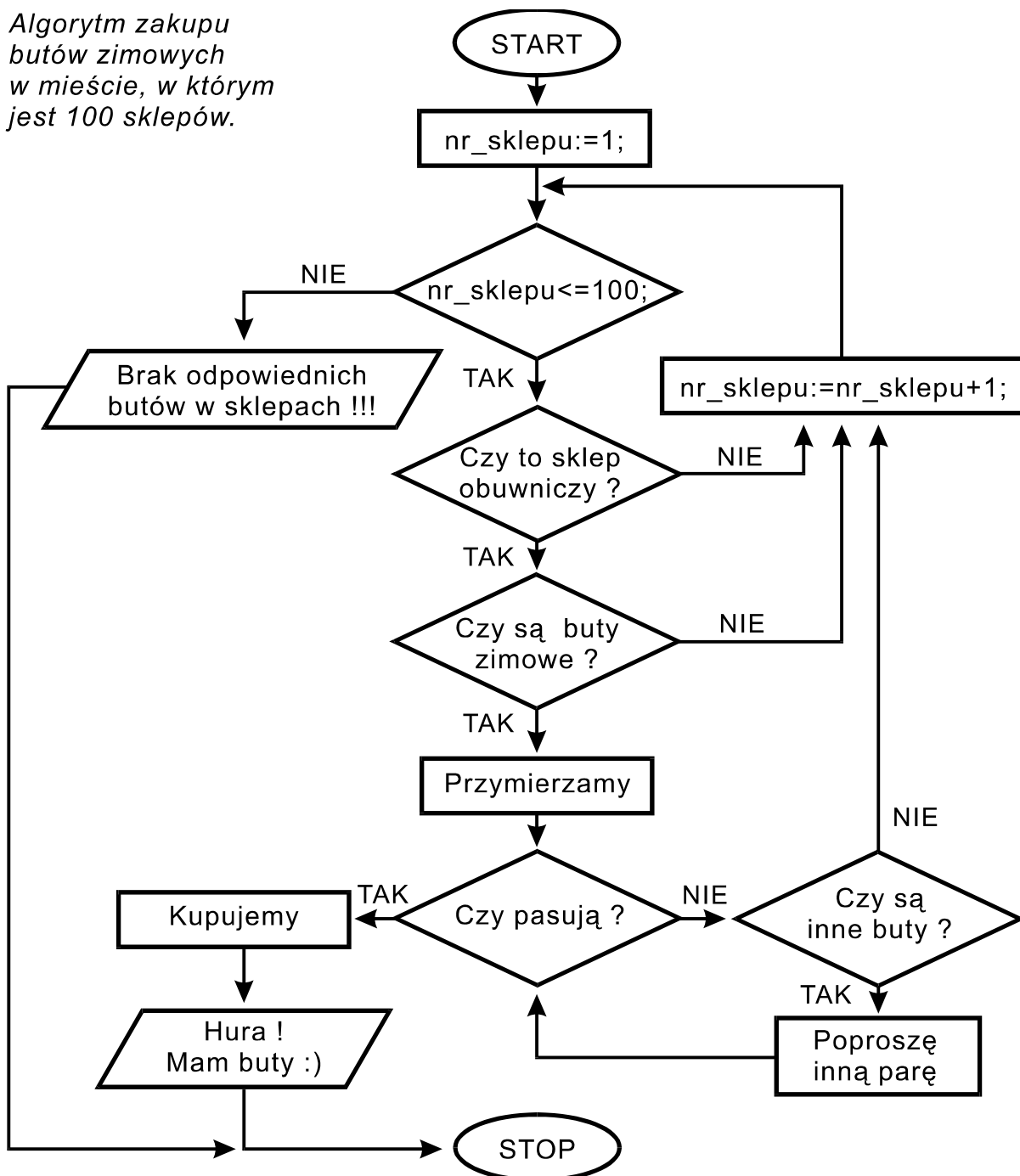
Ciągły rozwój języków programowania → Standardy C i C++

ROK	Język C	Język C++
1972	'udoskonalony' B	
1978	Kenigham & Ritchie C	
1979		C z klasami
1982→1985		C++
1989	ANSI C	C++ 2.0
1998		C++98
1999	<u>C99</u>	
2003		C++03
2007→2011	<u>C11</u>	<u>C++11</u>
2014		C++14
2017		C++17
2018	C18	
2020		C++20
2023	C23	C++23

# ZAPIS PROGRAMU ZA POMOCĄ SCHEMATÓW BLOKOWYCH



*Algorytm zakupu butów zimowych w mieście, w którym jest 100 sklepów.*



```
int main( ) { }
```

// najprostszy program w języku C/C++

---

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

// C/C++, z jawnym podaniem argumentów

---

```
#include < iostream >
int main()
{
    std::cout << "Czesc ! To ja, twój komputer" ;
    std::cin.get();
    return 0;
}
```

// C++, wypisanie tekstu na ekranie

// cout ← console output  
// cin ← console input

---

```
#include < iostream >
using namespace std;
```

// C++, proste obliczenia - iloczyn liczb

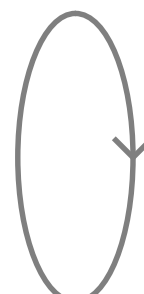
```
int main( )
{
    int   liczba_1, liczba_2 ;
    float wynik ;
    cout << endl << "To jest program obliczajacy iloczyn dwóch liczb " << endl ;
    cout << "Podaj pierwsza liczbe X = " ;
    cin >> liczba_1 ;
    cout << "Podaj druga liczbe      Y = " ;
    cin >> liczba_2 ;
    wynik = liczba_1 * liczba_2 ;
    cout << endl << "Wynik obliczenia X * Y = " << wynik << endl ;
    return 0;
}
```

---

```
#include < iostream >
using namespace std;
```

// C++ , cykliczne wykonywanie programu

```
int main( )
{
    char znak;
    do
    {
        .....
        instrukcje programu
        .....
        cout << endl << "Czy chcesz zakonczyc program ( T/N ) ?" ;
        cin >> znak ;
    }
    while( znak != ' t ' );
    cout << endl << "Koniec programu " ;
    return 0;
}
```



## Proceduralna i obiektowa komunikacja z użytkownikiem

```
/* proceduralnie: C / C++ */
#include <stdio.h>
int main(void)
{
    printf("Dzien ");
    printf("dobry!\n");
    getchar();
    return 0;
}
```

```
// obiektowo: C++
#include <iostream>
int main(void)
{
    std::cout << "Dzien ";
    std::cout << "dobry" << endl ;
    std::cin.get();
    return 0;
}
```

**#include** ← dyrektywa dołączenia tekstu zawartego w pliku

**stdio.h** ← (**StandardInputOutput**) plik definicji funkcji Wej/Wyj

**iostream** ← (**InputOutputStream**) plik definicji strumieni obiektowych

**main** ← zastrzeżona nazwa głównej funkcji programu

**void** ← typ danej "pustej"

**\n** ← przejście do nowego wiersza

**\t** ← znak tabulacji

**\"** ← znak cudzysłowu

**\\** ← jeden znak \

**endl** ← manipulator przejścia do nowej linii

*// 2 przykład → proceduralnie, C*

```
#include <stdio.h>

int x,y,s;
int main( )
{
    printf ("Podaj x = ");
    scanf ( "%d", &x );
    printf ("Podaj y = ");
    scanf ( "%d", &y );
    s = x+y;
    printf("Suma x+y = %d\n", s);

    fflush(stdin);
    getchar();
    return 0;
}
```

*// 2 przykład → obiektowo, C++*

```
#include <iostream>
using namespace std;

int x,y,s;
int main( )
{
    cout << "Podaj x = " ;
    cin >> x ;
    cout <<"Podaj y = " ;
    cin >> y ;
    s = x+y;
    cout<< "Suma x+y="<< s <<endl;

    cin.ignore( 100, '\n' );
    cin.get();
    return 0;
}
```

Definiowanie zmiennych → ustalenie nazwy, typu, rezerwacja pamięci

```
nazwa_typu nazwa_zmiennej ;
nazwa_typu zmienna_1, zmienna_2, zmienna_3 ;
```

Podstawowe typy: (dla aplikacji 32-bitowych)

Nazwa typu	Zawartość	Przedział wartości	Zajęta pamięć
<b>char</b>	znak	-128 ÷ 127	1 bajt
<b>int</b>	liczba całkowita	-2147mln ÷ 2147mln	4 bajty
<b>float</b>	liczba rzeczywista	$10^{-38} \div 10^{38}$ (7cyfr)	4 bajty
<b>double</b>	liczba rzeczywista	$10^{-308} \div 10^{308}$ (15 cyfr)	8 bajtów

Modyfikatory typu:

<b>signed</b>	→	ze znakiem ( $\pm$ ),	<b>int</b>	<b>char</b>	–
<b>unsigned</b>	→	bez znaku,	<b>int</b>	<b>char</b>	–
<b>short</b>	→	krótka (mniejsza),	<b>int</b>	–	–
<b>long</b>	→	długa (większa)	<b>int</b>	–	<b>double</b>

np. **unsigned long int dlugaLiczbaBezZnaku ;**

Wartości domyślne:

<b>long</b>	=	<b>long int</b>
<b>int</b>	=	<b>signed int</b>
<b>char</b>	=	<b>signed char</b>

Typ	Wielkość w bitach	Zakres
signed char	8	-128 ÷ 127
unsigned char	8	0 ÷ 255
short int	16	-32 768 ÷ 32 767
unsigned short int	16	0 ÷ 65 535
signed int	32	-2 147 483 648 ÷ 2 147 483 647
unsigned int	32	0 ÷ 4 294 967 295
signed long int	32	-2 147 483 648 ÷ 2 147 483 647
unsigned long int	32	0 ÷ 4 294 967 295
float	32	$1.2 * 10^{-38} \div 3.4 * 10^{+38}$
double	64	$2.2 * 10^{-308} \div 1.8 * 10^{+308}$
long double	80	$3.4 * (10^{*-4932}) \div 1.2 * (10^{*+4932})$
bool	8	true (prawda), false (fałsz)



## OPERATORY

operatory arytmetyczne:

+	dodawanie
-	odejmowanie
*	mnożenie
/	dzielenie
%	reszta z dzielenia

operatory przypisania:

=	zwykłe przypisanie	x = 2;
+=	przypisanie sumy	x+=2; → x = x + 2;
-=	przypisanie różnicy	x-=2; → x = x - 2;
*=	przypisanie iloczynu	x*=2; → x = x * 2;
/=	przypisanie ilorazu	x /=2; → x = x / 2;
%=	przypisanie reszty	x%=2; → x = x % 2;

operatory inkrementacji i dekrementacji:

**zmienna++** – inkrementacja zmiennej po wyliczeniu wyrażenia  
**++zmienna** – inkrementacja zmiennej przed wyliczeniem wyrażenia  
**zmienna--** – dekrementacja zmiennej po wyliczeniu wyrażenia  
**--zmienna** – dekrementacja zmiennej przed wyliczeniem wyrażenia

np. `int x, y = 1;`

`x = ++y ; /* rezultat: x=2, y=2 */`      `x = y ++ ; /* rezultat: x=1, y=2 */`

operatory relacyjne:

==	równe
!=	różne
<	mniejsze
>	większe
<=	mniejsze lub równe
>=	większe lub równe

operatory logiczne:

&&	koniunkcja (AND)
	alternatywa (OR)
!	negacja (NOT)

bitowe operatory logiczne:

&	bitowa koniunkcja (AND)
	bitowa alternatywa (OR)
^	bitowa różnica symetryczna (XOR)
<<	przesunięcie bitów w lewo
>>	przesunięcie bitów w prawo
~	negacja bitów

## Priorytety operatorów w języku C:

Operator	Opis	Przykład
( )	wywołanie funkcji	sin()
[ ]	element tablicy	tab[10]
.	element struktury	osoba.nazwisko
->	wskazanie elementu struktury	wsk_osoby->nazwisko
!	negacja logiczna	if( !(x > max) ) kontynuuj;
~	negacja bitowa	~(001101) ≡ (110010)
-	zmiana znaku (negacja)	x = 10 * (- y)
++	inkrementacja (zwiększenie o 1)	x+++y ≡ (x++) + y
--	dekrementacja (zmniejszenie o 1)	--y ≠ --y ≡ -( - y)
&	operator referencji (adres elementu)	wsk_x = &x
*	operator dereferencji	*wsk_x = 10
(type)	zmiana typu ( <i>typecast</i> )	(double) 10 ≡ 10.0
sizeof	rozmiar zmiennej lub typu (w bajtach)	sizeof( int ) ≡ 2
*	mnożenie	
/	dzielenie	
%	operacja modulo (reszta z dzielenia)	if( x%2 == 0 ) parzyste;
+	dodawanie	
-	odejmowanie	
<<	przesunięcie bitowe w lewo	1 << 2 ≡ (0001) << 2 ≡ (0100)
>>	przesunięcie bitowe w prawo	x = 4 >> 1 ≡ x = 2
<	mniejszy niż	if( liczba < max ) max = liczba;
<=	mniejszy lub równy	
>	wiekszy niż	
>=	wiekszy lub równy	
==	równy	
!=	nie równy (różny od)	
&	iloczyn bitowy	
^	suma bitowa modulo (różnica symetryczna)	
	suma bitowa	
&&	iloczyn logiczny	
	suma logiczna	
?:	wyrażenie warunkowe	
=	przypisanie	
*= /= %= +=	przypisania arytmetyczne	
-- <<= >>=		
&= ^=  =		
,	operator przecinka	

**Przykład:**    `int x=1, y=2, z=3, wynik=4 ;`

wynik \*= `--++x*x--+-y--%++z;`    **(???)**

wynik \*= `--(++x) * (x--) +- (y--) % (++z);`

wynik \*= `(--(++x)) * (x--) + (--(y--)) % (++z);`

wynik \*= `((--(++x))*(x--)) + ((--(y--))%(++z));`    // x=1, y=1, z=4, wynik=-24