Język programowania C++

(*wykł. dr Marek Piasecki*)

Literatura:

- **dowolny podręcznik do języka C++** (na laboratoriach → Borland C++ 3.1)
- **Robert Lafore** "Programowanie w języku C przy użyciu Turbo C++"
- **Jerzy Grębosz** "Symfonia C++"
- **Andrzej Zalewski** "Programowanie w językach C i C++ z wykorzystaniem pakietu Borland 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++
   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, parametry.

7. Funkcje operujące na pamięci: biblioteka <mem.h>
   Łańcuchy znaków. Funkcje łańcuchowe <string.h>

8. **Typ strukturalny** – definicja, deklaracja i inicjacja 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.

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 celem. 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.

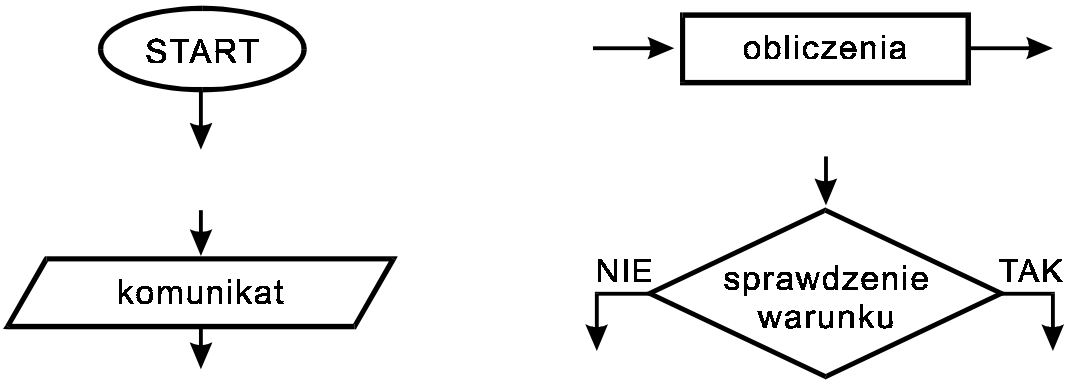
Proces tworzenia (kodowania?) programu:

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

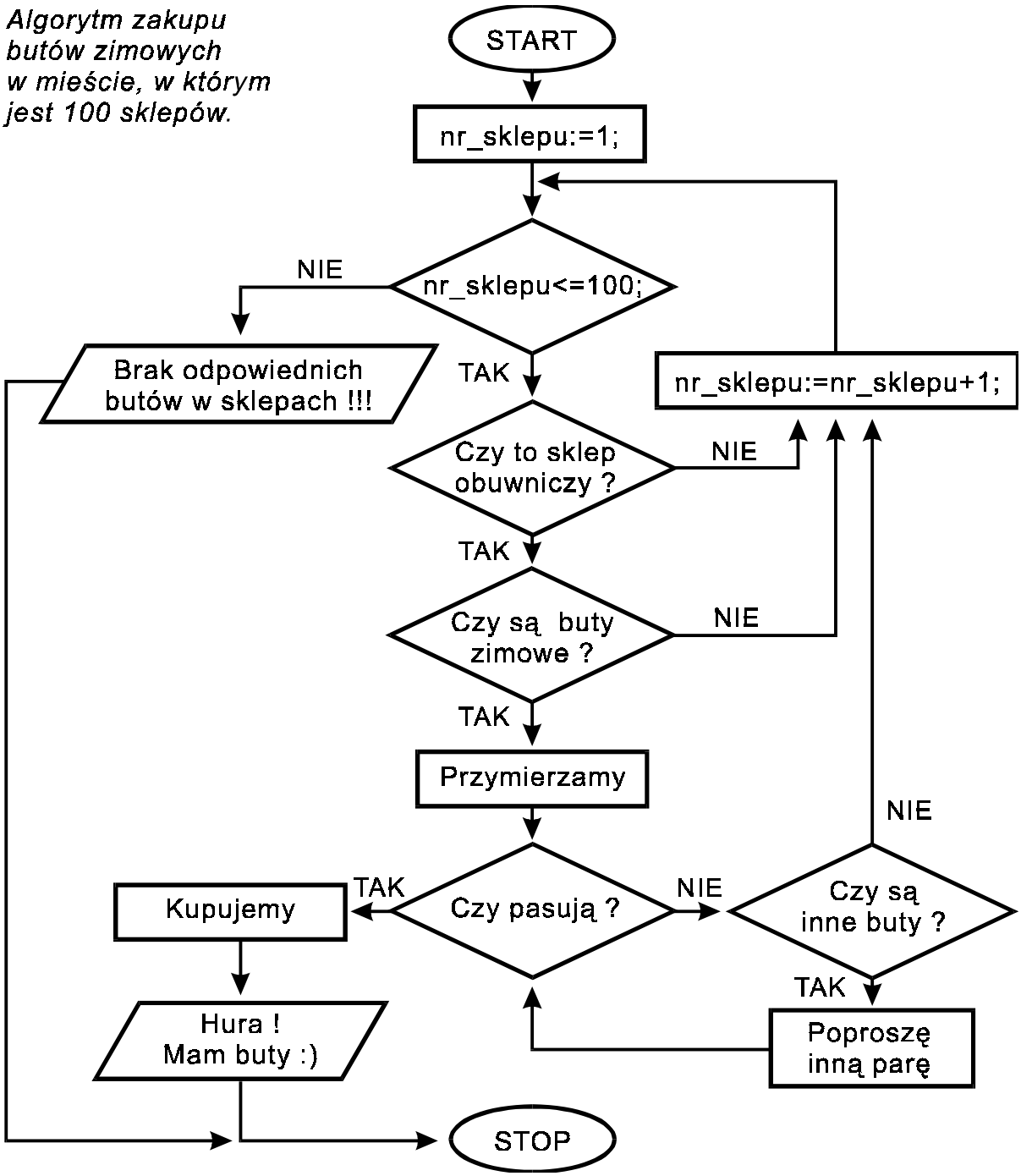
Język C++ jest rozszerzeniem **języka C :**

- 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

ZAPIS PROGRAMU ZA POMOCĄ SCHEMATÓW BLOKOWYCH



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



```
void main( )        ..............................     // najprostszy program w języku C / C++
{ }
```

```
#include < iostream.h >        ................     // wypisanie tekstu na ekranie  (C++)
void main( void )
{
  cout << ”Czesc !  To ja, twój komputer” ;
}
```

```
#include < iostream.h >        ............     // proste obliczenia - iloczyn liczb (C++)
void 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 ;
}
```

```
#include < iostream.h >        ..................     // cykliczne wykonywanie programu
void main( )
{
   char  znak;
   do
     { . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
         instrukcje programu
       cout << endl << ”Czy chcesz zakonczyc program  ( T/N )  ? ” ;
       cin >> znak ;
     } . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
   while(  znak != ' t ' ) ;
   cout << endl << ”Koniec programu ” ;
}
```

Proceduralna i obiektowa komunikacja z użytkownikiem

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

```c
// obiektowo:   C++
#include <iostream.h>
void main(void)
{
  cout << "Dzien " ;
  cout << "dobry" << endl ;
}
```

#include ← dyrektywa dołączenia tekstu zawartego w pliku

stdio.h ← (**St**andar**dI**nput**O**utput) plik definicji funkcji Wej/Wyj

iostream.h ← (**I**nput**O**utput**Stream**) plik definicji strumieni obiektowych

main ← zastrzeżona nazwa głównej funkcji programu

void ← typ danej "pustej"

\n ← przejscie do nowego wiersza

\t ← znak tabulacji

\" ← znak cudzysłowu

**** ← jeden znak \

endl ← manipulator przejścia do nowej linii

```c
// 2 przyklad → proceduralnie
#include <stdio.h>
#include <conio.h>
int x,y,s;
void main(void)
{
  clrscr();
  printf ("Podaj x = ");
  scanf ( "%d" , &x );
  printf ("Podaj y = ");
  scanf ( "%d" , &y );
  s = x+y;
  printf("Suma x+y = %d\n", s );
  getch();
}
```

```c
// 2 przyklad → obiektowo
#include <iostream.h>
#include <conio.h>
int x,y,s;
void main(void)
{
  clrscr();
  cout << "Podaj x = " ;
  cin >> x ;
  cout <<"Podaj y = " ;
  cin >> y ;
  s = x+y;
  cout << "Suma x+y=" << s << '\n' ;
  getch();
}
```

Definiowanie zmiennych → ustalenie nazwy, typu, rezerwacja pamięci

nazwa_typu *nazwa_zmiennej* **;**
nazwa_typu *zmienna_1, zmienna_2, zmienna_3* ;

Podstawowe typy:

Nazwa typu	Zawartość	Przedział wartości	Zajęt. pamięć
char	znak	-128 ÷ 127	1 bajt
int	liczba całkowita	-32768 ÷ 32767	2 bajty
long	liczba całkowita	-2147mln ÷ 2147mln	4 bajty
float	liczba rzeczyw.	10^{-38} ÷ 10^{38} (7cyfr)	4 bajty
double	liczba rzeczyw.	10^{-308} ÷ 10^{308} (15 cyfr)	8 bajtów

Modyfikatory typu:

signed	→	ze znakiem (±),	**int**	**char**	–
unsigned	→	bez znaku,	**int**	**char**	–
short	→	krótka (mniejsza),	**int**	–	–
long	→	długa (większa)	**int**	–	**double**

np. **unsigned long int** *dluga_liczba_bez_znaku* ;

Wartości domyślne:

long	=	**long int**
int	=	**signed int**
char	=	**signed char**

Type	Length	Range
unsigned char	8 bits	0 ÷ 255
char	8 bits	-128 ÷ 127
enum	16 bits	-32,768 ÷ 32,767
unsigned int	16 bits	0 ÷ 65,535
short int	16 bits	-32,768 ÷ 32,767
int	16 bits	-32,768 ÷ 32,767
unsigned long	32 bits	0 ÷ 4,294,967,295
long	32 bits	-2,147,483,648 ÷ 2,147,483,647
float	32 bits	3.4 * (10**-38) ÷ 3.4 * (10**+38)
double	64 bits	1.7 * (10**-308) ÷ 1.7 * (10**+308)
long double	80 bits	3.4 * (10**-4932) ÷ 1.1 * (10**+4932)

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 elemenu 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 (zmiejszenie o 1)	– – y ≠ – – y ≡ – (– y)
&	operator referencji (adres elementu)	wsk_x = &x
∗	operator dereferencji	∗wsk_x = 10
(type)	zmiana typu (*typecast*)	(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)
>>	przesuniecie bitowe w prawo	x = 4 >>1 ≡ x = 2
<	mniejszy niż	if(liczba < max) max = liczba;
<=	mniejszy lub równy	
>	wiekszy niż	
>=	wiekszy lub równy	
==	równy	
!=	nierówny (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 ;  //rezultat: x=1, y=1, z=4, wynik=–24

wynik ∗= –++x∗x––+–y––%++z;  (???)

wynik ∗= – (++x) ∗ (x––) +– (y––) % (++z);
wynik ∗= (–(++x)) ∗ (x––) + (–(y––)) % (++z);
wynik ∗= ((–(++x))∗(x––)) + ((–(y––))%(++z));
```