Struktura programu

```
#include <...  .h>
int main (int argc, char **argv)
{
  // instrukcje
}
```

Słowa kluczowe

Słowa o predefiniowanym znaczeniu. Nie mogą być użyte jako identyfikatory (np. nazwy zmiennych).
Do słów kluczowych języka C++ należą m.in.:

int	**long**	**float**	**double**
if	**else**	**for**	**while**
void	**class**	**private**	**try**

Komentarze

Ignorowane przez kompilator.

```
// od łamane-łamane do końca linii
/* od łamane-gwiazdka ...
      ... do gwiazdka-łamane          */
```

Zmienne

Przy deklarowaniu zmiennych obowiązyje składnia:

```
typ deklarator1, deklarator2, ... ;
```

```
int a, b, c;
int p = 0, q, r = 1;
```

Podstawowe typy zmiennych prostych:

```
char, short, long, int, float, double
```

Literały

Napis reprezentujący wartość (np. liczbową).
Literał ma typ – jest to najprostszy typ, w którym można go zapisać:

```
1           // int
1.0         // float
```

Konwersje

Zmiana typu zmiennej lub literału. Konwersje standardowe (np. między typami *int* oraz *float*) wykonywane są automatycznie. Można też wymusić konwersję:

```
int a = 3.99;          // a=3 (nie zaokragla!)
float x = 1;           // x=1.0
x = a / 4;             // x=0     (3/4 = 0 r 3)
x = float(a) / 4;      // x=0.75  (3.0/4 = 0.75)
```

Operatory

zwiększania, przyrostkowe	++ --	lewe
zwiększania, przedrostkowe	++ --	prawe
dotyczące wskaźników	* & new delete	lewe
jednoargumentowe	! + -	prawe
arytmetyczne dwuargumentowe	* / %	lewe
	+ -	lewe
dotyczące strumieni - przesłania	<< >>	lewe
relacji	< <= > >=	lewe
równości	== !=	lewe
logiczne	&&	lewe
	||	lewe
przypisania (w tym rozszerzone)	= += *= itd.	prawe
łączenia	,	lewe

Kolejność operacji, wynikającą z hierarchii i wiązania, można zmienić używając nawiasów. Służą do tego wyłącznie nawiasy okrągłe: ( ).

Biblioteki C++ - użyteczne obiekty i funkcje

iostream.h

```
cout          - strumień wyjścia
cin           - strumień wejścia

cout << "Suma " << s << "\n";
cin >> a >> b;
```

math.h

```
M_PI          - liczba π
abs (n)       - moduł liczby całkowitej
fabs (x)      - moduł liczby rzeczywistej
sqrt (x)      - pierwiastek kwadratowy
log (x)       - logarytm naturalny
log10 (x)     - logarytm dziesiętny
exp (x)       - exponenta
sin (x)       - sinus; argument w radianach
cos (x)       - cosinus
pow (x, y)    - x  do potęgi y
```

math.h lub **stdlib.h**

```
randomize ()  // inicjalizacja
random (n)    // liczba losowa (0 .. n-1)
```

```
float r, m=-3.0, s=4.2, t=8.0;
r = fabs(m);                    // r = |m|
r = sqrt(t/M_PI);               // r = √(s/π)
r = sin(t*M_PI/180.0);          // r = sin(t°)
r = log(t)/log(2.0);            // r = log2(t)
r = pow (t, 3.0);               // r = t^3
r = pow (8.0, 1.0/3.0);         // r = 8^(1/3) = 2
```

Argumentem funkcji może być literał, zmienna lub wyrażenie.
Powyżej *n* oznacza argument całkowity, *x* i *y* argumenty rzeczywiste.

Instrukcje

1. Instrukcja pusta

```
;          // średnik
```

2. Instrukcja grupująca (blok instrukcji)

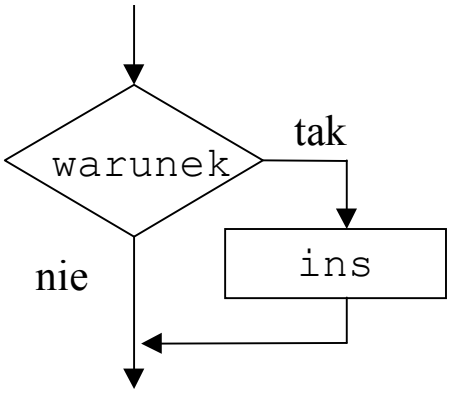
```
{
          // tutaj dowolna liczba instrukcji
}
```

3. Instrukcja wyrażeniowa
   Poprawne składniowo wyrażenie (kombinacja literałów zmiennych i operatorów), zakończone średnikiem:

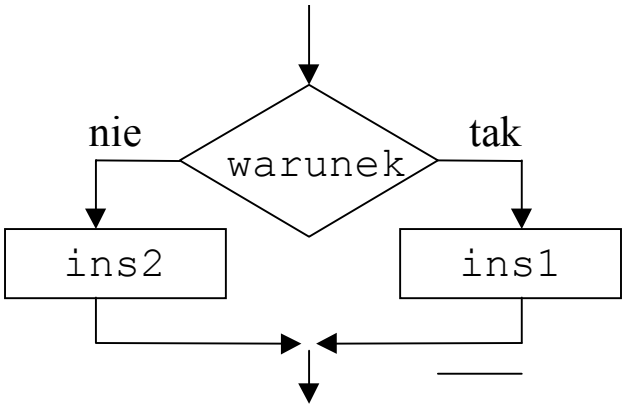
```
Wyrażenie;
```

4. Instrukcja warunkowa

```
if (warunek)
  instrukcja
```



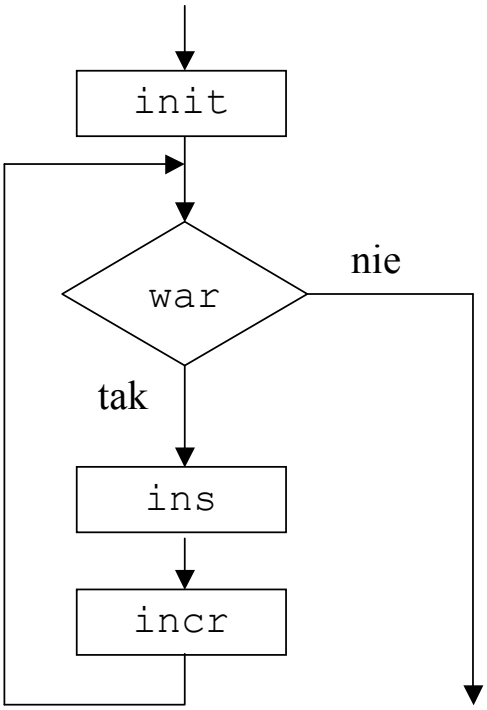

```
if (warunek)
  instrukcja_1
else
  instrukcja_2
```



Instrukcje (cd.)

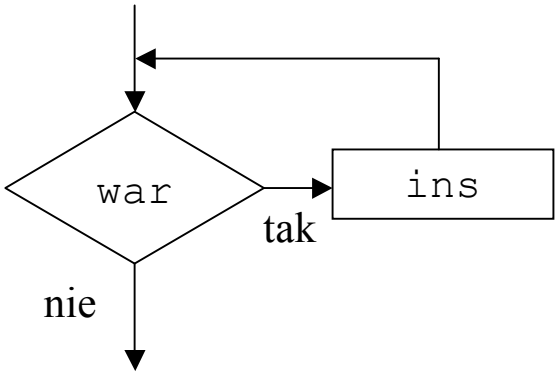
5. Instrukcje iteracyjne (pętle programowe)

```
for (inicjalizacja; warunek; inkrementacja)
  instrukcja
```



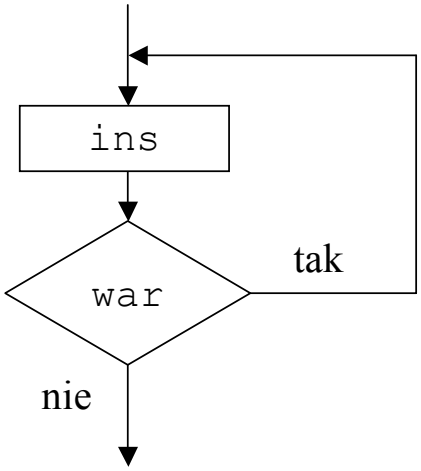

```
for (i=1; i<10; i++)
  cout << i;

for (x=1.5; x<3.0; x+=0.1)
  cout << x;

s = 1;
for (i=1; i<n; i++)
  s *= i;
```

```
while (warunek)
  instrukcja
```




```
while (!file.eof())
{
  file.getline(s, 100);
  cout << s << "\n";
}
```

```
do
  instrukcja
while (warunek);
```




```
do
{
  y = 0.5 * (x/y +y);
  d = fabs(y*y-x) / x;
}
  while (d>tolerance);
```

Wskaźniki

Wskaźnik to zmienna, która przechowuje adres innej zmiennej. Wskaźnik ma określony w deklaracji typ – inaczej nie można by było wykonywać operacji na zmiennych wskazywanych przez wskaźniki. Do operacji na wskaźnikach służą operatory pobrania adresu (&) oraz wyłuskania (*):

```
typ *identyfikator;

int *wsk;
int a, b;

wsk = &a;                    // wsk = adres (a)
*wsk = 7;                    // a = 7
wsk = &b;                    // wsk = adres (b)
*wsk = 7;                    // b = 7
```

Odnośniki

Odnośnik to zmienna, która identyfikuje inną zmienną. Wykonanie operacji na odnośniku ma taki sam skutek, jak na zmiennej identyfikowanej przez odnośnik.

```
typ &identyfikator = zmienna;

int a;
int &odn = a;

odn = 7                      // a = 7
cout << odn;                 // cout << a
```

Odnośniki spełniają ważną rolę w funkcjach, w tym w definicjach operatorów. Tylko dzięki odnośnikom można zapisać np.:

```
y = (x = 2);          // rezultatem x=2 jest x
(cout << x) << y;     // rez. cout<<x jest cout
```

Tablice

Tablica jest zmienną złożoną. Składa się z wielu elementów tego samego typu. Deklaracja tablicy ma postać:

```
typ identyfikator [rozmiar]

int  A[3];                  // A[0] .. A[2]
float Tab[100];             // Tab[0] .. Tab[99]
```

Dostęp do poszczególnych elementów tablicy umożliwia operator indeksowania. Domyślnie w C++ nie ma kontroli wartości indeksów.

```
int  A[3];
A[0] = 3;
cout << A[0];
A[3] = 7;                   // poza tablicą!
```

Do obsługi tablic zazwyczaj wykorzystuje się pętlę *for*.
Konstrukcja pętli ma typowo postać:

```
int  tab[rozm];             // "rozm" musi być
for (i=0; i<rozm; i++)      // wyr. stałym!
{
   .. tab[i] ..
}
```

Jeżeli rozmiar tablicy nie jest znany w czasie kompilacji, tablica musi być tworzona i niszczona dynamicznie (operatory *new* oraz *delete*):

```
float *wsk;
int rozm;

cin >> rozm;
wsk = new float [rozm];
for (i=0; i<rozm; i++)      //korzystamy z 'wsk'
  cout << wsk[i];           //jak z tablicy
delete []wsk;
```

Znaki

Reprezentacja znaków w C++:

```
char c1;
char c2 = 'A';
char c3 = '\n';
```

Kod ASCII:

	0x	1x	2x	3x	4x	5x	6x	7x	
x0	NULL			0	@	P	`	p	x0
x1			!	1	A	Q	a	q	x1
x2			"	2	B	R	b	r	x2
x3			#	3	C	S	c	s	x3
x4			**$**	4	D	T	d	t	x4
x5			%	5	E	U	e	u	x5
x6			&	6	F	V	f	v	x6
x7	BEL		'	7	G	W	g	w	x7
x8	BS		(	8	H	X	h	x	x8
x9	TAB		)	9	I	Y	I	y	x9
xA	LF		*	:	J	Z	j	z	xA
xB		ESC	+	;	K	[	k	{	xB
xC	FF		,	<	L	\	l	|	xC
xD	CR		-	=	M	]	m	}	xD
xE			.	>	N	^	n	~	xE
xF			/	?	O	_	o		xF

Dwoista natura zmiennych i stałych typu znakowego:

```
c1 = 'A';
c2 = 65;                    // 'A'

cout << 'd';                // d
cout << +'d';               // 100
cout << 'd' * 'd';          // 10000

c1 = 'A' + 32;              // c1 = 97
cout << c1;                 // 'a'
cout << +c1;                // 97
```

Łańcuchy

Reprezentacja łańcuchów w C++:

```
char str1[25];              // 24 znaki i '\0'
char str2[50] = "Hello"; // 'H'…'o' '\0'
char str3[] = "Hello";    // str3[6]

char *p_str1, *p_str2;
p_str1 = new char [roz];
p_str2 = str2;              // & str2[0]

cout << str2;               // char*
cout << p_str2;             // char*
cout << "Hello";            // char*

cout << &str2[0];           // "Hello" (char*)
cout << str2[0];            // 'H'     (char)
cout << "Hello"[1];         // 'e'
```

Przetwarzanie łańcuchów:

```
char s1[20] = "Hello", s2[20];

for (i=0; i<20; i++)          // kopiowanie
  s2[i] = s1[i];

char *p = s1;
char aA = 'a' - 'A';          // zamiana liter
while (*p)                    // małych na duże
  {
  if (*p>='a' && *p<='z')
    *p -= aA;
  p++;
  }
```

Funkcje do obsługi łańcuchów

Wybrane funkcje do obsługi łańcuchów (`string.h`)

strcat	strchr	**strcmp**	strcmpi	strcoll
strcpy	strcspn	**strdup**	stricmp	**strlen**
strlwr	strncat	strncmp	strncmpi	strncpy
strnset	strpbrk	strrchr	strrev	strset
strspn	strstr	strtok	**strupr**	strxfrm

Informacje dostępne w systemie pomocy [Ctrl]+[F1]:

Syntax

```
#include <string.h>
char *strcat(char *dest, const char *src);
```

Description

strcat appends a copy of *src* to the end of *dest*. The length of the resulting string is *strlen*(*dest*) + *strlen*(*src*).

Return Value

strcat returns a pointer to the concatenated strings.

Deklaracje wybranych funkcji:

```
size_t strlen (const char *s);
char *stpcpy (char *dest, const char *src);
char *strcat (char *dest, const char *src);
char *strdup (const char *s);
char *strlwr (char *s);
char *strupr (char *s);
int strcmp (const char *s1, const char *s2);

char s1[20], s2[20], s3[40];
strcpy (s3, s1);
strupr (s3);
strcat (s3, ", ");
strcat (s3, s2);
```

Funkcje

Deklarcja (prototyp) funkcji

```
typ_rezultatu nazwa (lista_argumentów);

float sqrt (float x);
float pow (float b, float e);
int rand (void);
void randomize (void);
```

Definicja funkcji

```
typ_rezultatu nazwa (lista_argumentów)
{
  // ciało funkcji
}

float suma (float a, float b)
{
      float c;
      c = a + b;
      return c;
}
```

Wywołanie funkcji

```
nazwa_funkcji (lista_argumentów_aktualnych)

x = sqrt (4);
x = sqrt (3*y +8);
x = sqrt (y * power (4, sqrt (Y)));

x = rand ();          // f. bezargumentowe:
randomize ();         // () są niezbędne
```

Instrukcja *return*
- funkcje mające rezultat:

```
typ_rezultatu nazwa (lista_argumentów)
{
      // ciało funkcji
      return wyrażenie;
}

int sum (int a, int b)
{
      return a+b;
}

int abs (int n)
{
      if (n<0)
           return -n;
      else
           return n;
}
```

- funkcje nie mające rezultatu (bezrezultatowe):

```
typ_rezultatu nazwa (lista_argumentów)
{
      // ciało funkcji
      return;
}

void welcome (void)
{
      cout << "Welcome \n";
      // return domyślne - można pominąć
}
```

Funkcja printf

Wyjście formatowane: funkcje printf, sprintf, fprintf

```
#include <stdio.h>
printf ("format", wyrażenie, wyrażenie, ...);
```

W łańcuchu *format*, każdemu *wyrażeniu* odpowiada sekwencja:

```
% [fl] [szer] [.prec] [mod] typ
```

```
typ:      d,i  int
          u    unsigned int
          f    float – 123.456
          e    float – 1.23456e+2
          g    e lub f, zależnie od wartości
          c    char
          s    char *

fl:       +    wymusza wydruk znaku
          -    wyrównanie do lewej strony
               np. %+i, %-f
szer:     n    min. szerokość wydruku
          0n   zera zamiast spacji
               np. %6i, %05i
.prec:    n    ilość znaków (cyfr)
               – znaczenie zależy od typu:
               d,i – n cyfr (np. 3 → 015)
               e,f – n cyfr po przec. (1.500)
               g   - n cyfr znaczących (15.0)
               s   - n znaków
               np. %8.3f, %.5g, %+04.1f
mod:      l    przed d,i → long
               przed e,f,g → double
               np. %li, %+10.5le

printf ("średnia = %.2f", sr);
printf ("Nazw.: %20s \nImie: %20s", n, i);
printf ("%10i %10i %+12.3f", a, b, x);
```

Funkcja scanf

Wejście formatowane: funkcje scanf, sscanf, fscanf.

```
#include <stdio.h>
int scanf ("format", adres, adres, ...);
```

W łańcuchu *format*, każdemu *adresowi* odpowiada sekwencja:

```
% [fl] [szer] [mod] typ
```

```
typ:      d,   int
          D    long
          u    unsigned int
          U    unsigned long
          e,f,
          g    float
          c    char
          s    char *
          []   char * ([abc], [a-c], [^a-c])

fl:       *    wymusza odczytanie następnego
               pola bez zapisania
               np. %*i
szer:     n    max. liczba znaków do odczyania
               np. %10s
mod:      l    przed d,i → long
               przed e,f,g → double
          h    przed d,i → short
               np. %le, %hi
```

```
scanf ("%d , %e", &num, &val); // "3, 1.2"
scanf ("%*d , %e", &val);      // "3, 1.2"
scanf ("%*[a-z] = %e", &val);  // "num = 1.2"
scanf ("%*[^=] = %e", &val);   // "num = 1.2"
scanf ("%*[^0-9] %e", &val);   // "num = 1.2"
```

Pliki

Mechanizmy obsługi plików:

```
#include <io.h>
int read (int handle, void *buf, int len);

#include <stdio.h>
FILE *fopen (char *n, char *m);

#include <fstream.h>
ofstream f1 (char *n);
```

Strumienie plikowe i ich metody:

```
ofstream f;
ofstream f (char *path [, int mode]);
ifstream f;
ifstream f (char *path);

void open (char *path [, int mode]);
void close (void);
int eof (void);
int fail (void);
void getline (char *ln, int maxl, char dlm);

ofstream f ("C:\\Plik.txt");
f << "Wynik: " << w;
f.close ();

ofstream fo;
fo.open ("C:\\Plik.txt", ios::app);
fo << "Wynik: " << w;
fo.close ();

ifstream fi ("C:\\Plik.txt");
fi.getline (ln, 100);
fi.close ();
```

Pliki

Obsługa plików przez zmienne plikowe

```
#include <stdio.h>

FILE *fopen (char *fn, char *mode);
int feof (FILE *f);
int fclose (FILE *f);
int ferror (FILE *f);
int fseek (FILE *f, long offset, int whnc);
long ftell (FILE *f);
void clearerr (FILE *f);
void rewind (FILE *f);

int fprintf (FILE *f, char *format,
             wyr_1, wyr_2, ...);
int fscanf (FILE *f, char *format,
             adr_1, adr_2, ...);

size_t fwrite (void *ptr,
               size_t size, size_t n,
               FILE *f);
size_t fread  (void *ptr,
               size_t size, size_t n,
               FILE *f);

// tryb tekstowy
f = fopen ("Plik.txt", "wt");
fprintf (f, "wynik: %+10.4f\n", w);
fclose (f);
// tryb binarny
f = fopen ("Plik.txt", "rb");
fseek (f, 0, SEEK_END)
len = ftell (f);
fseek (f, 0, SEEK_SET);
fread (buf, sizeof(char), len, f);
fclose (f);
```

fopen

```
FILE *fopen (char *fn, char *mode);
```

Otwiera plik.
fn –nazwa pliku, *mode* – tryb: r/w/a (odczyt /zapis /dopisywanie) oraz b/t (binarny/tekstowy) – np. "rt". Zwraca wskaźnik pliku lub *NULL*.

feof

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

Wykrywa koniec pliku (EOF)
Zwraca wartość niezerową, jeżeli wykryje koniec pliku
lub 0 w przeciwnym wypadku

fclose

```
int fclose (FILE *f);
```

Zamyka plik.
Zwraca 0 w przpadku powodzenia lub *EOF* w przypadku błędu.

ferror

```
int ferror (FILE *stream);
```

Wykrywa błąd opearacji na plikach.
Zwraca wartość niezerową, jeżeli wykryje błąd. Znacznik błędu pozostawia ustawiony.

clearerr

```
void clearerr (FILE *stream);
```

Kasuje znacznik błędu.
Jeżeli znacznik ten jest ustawiony, każda operacja na pliku zwraca kod błędu, aż do wywołania *clearerr* lub *rewind*.

fprintf

```
int fprintf (FILE *f, char *format,
             arg_1, arg_2, ...);
```

Wykonuje formatowany zapis do pliku.
Zasady formatowania są identyczne, jak dla *printf*. Zwraca liczbę bajtów zapisanych do pliku lub stałą *EOF* w przypadku błędu.

fscanf

```
int fscanf (FILE *f, char *format,
             adr_1, adr_2, ...);
```

Wykonuje formatowany odczyt z pliku.
Zasady formatowania są identyczne, jak dla *scanf*. Zwraca liczbę pól odczytanych lub stałą *EOF* przy próbie czytania na końcu pliku.

fwrite

```
size_t fwrite (void *ptr, size_t size,
               size_t n, FILE *f);
```

Zapisuje dane do pliku.
Zapisuje *n* elementów o rozmiarze *size* bajtów, poczynając od adresu *ptr*. Zwraca liczbę elementów (nie bajtów) faktycznie zapisanych.

fread

```
size_t fread  (void *ptr, size_t size,
               size_t n, FILE *f);
```

Odczytuje dane z pliku.
Odczytuje z pliku *n* elementów o rozmiarze *size* bajtów i zapisuje dane poczynając od adresu *ptr*. Zwraca liczbę elementów (nie bajtów) faktycznie odczytanych.