Język C (#1)

Jednostki leksykalne języka C:
- **identyfikatory**
- **słowa kluczowe**
- **stałe**
- **napisy**
- **operatory**
- **separatory**

Odstępy (spacje), pozioma i pionowa tabulacja, znaki nowego wiersza i nowej strony są ignorowane (służą do rozdzielania jednostek leksykalnych).

Komentarze

postać klasyczna :

```
/* ........................................
   ........................................
   ........................................
   ........................................ */
```

- **mogą rozciągać się na wiele linii,**
- **nie mogą występować w napisach i stałych znakowych,**
- **NIE MOGĄ być zagnieżdżone (standard ANSI),**
- **większość kompilatorów akceptuje (opcjonalnie) zagnieżdżanie,**
- **wiele kompilatorów akceptuje również komentarze języka C++ o postaci:**
  `// ..... ↵`

Identyfikatory

- **ciąg liter (alfabetu angielskiego) i cyfr dziesiętnych**
- **pierwszym znakiem musi być litera (znak _ jest zaliczany do liter)**
- **wielkie i małe litery są rozróżniane**
- **mogą być dowolnej długości**
- **liczba początkowych znaków znaczących zależy od łączności, i tak:**
  - **dla wewnętrznej (jedna jednostka tłumaczenia) co najmniej 31**
  - **dla zewnętrznej co najmniej 6 (identyfikatory międzymodułowe)**

```
alfa Alfa AlfA ALFA    /* uwaga - różne zmienne ! */
Cena_Mleka
__zmienna_systemowa
```

Słowa kluczowe

Słowa kluczowe jezyka C są zastrzeżone – nie mogą być używane do innych celów.

Język ANSI C zawiera następujące słowa kluczowe:

`auto`	**klasa pamięci - zmienna lokalna**
`break`	**wyjście z pętli lub instrukcji wyboru**
`case`	**alternatywa w instrukcji wyboru**
`char`	**typ danych – typ znakowy**
`const`	**kwalifikator typu – wartość nie będzie modyfikowana**
`continue`	**instrukcja powrotu do początku pętli**
`default`	**domyślna alternatywa w instrukcji wyboru**
`do`	**instrukcja – część pętli *do-while***
`double`	**typ danych – typ rzeczywisty podwójnej dokładności**
`else`	**instrukcja – opcjonalna część instrukcji warunkowej *if***
`enum`	**deklaracja typu wyliczeniowego**
`extern`	**klasa pamięci – symbol zewnętrzny**
`float`	**typ danych – typ rzeczywisty pojedynczej precyzji**
`for`	**instrukcja – część pętli *for***
`goto`	**instrukcja skoku**
`if`	**instrukcja – element instrukcji warunkowej *if***
`int`	**typ danych – typ całkowity ze znakiem**

`long`	**modyfikator i/lub typ danych**
`register`	**klasa pamięci – zmienna rejestrowa**
`return`	**instrukcja powrotu z podprogramu**
`short`	**modyfikator i/lub typ danych**
`signed`	**modyfikator – zmienna ze znakiem**
`sizeof`	**operator rozmiaru**
`static`	**klasa pamięci – zmienna statyczna/symbol lokalny**
`struct`	**deklaracja struktury (rekordu)**
`switch`	**instrukcja wyboru**
`typedef`	**nazywanie typów**
`union`	**deklaracja unii (rekord z wariantami)**
`unsigned`	**modyfikator – zmienna bez znaku**
`void`	**typ danych**
`volatile`	**kwalifikator typu – zmienna ulotna**
`while`	**instrukcja – część pętli *while* i *do-while***

Słowa kluczowe nie należące do ANSI (rozszerzenia) to m. in.:

`asm`	**wstawki w języku wewnętrznym**

`near`
`far`
`huge`
`cdecl`
`pascal`
	modyfikatory

Stałe

➢ **stała całkowita**

typ liczby określany jest na podstawie:
- **wartości (pierwszy możliwy z int, unsigned int, unsigned long int)**
- **zapisu liczby (przyrostek u/U i l/L)**

Literał	Typ
5	**int**
5u	**unsigned int**
5U	**unsigned int**
5l	**long int**
5L	**long int**
5ul	**unsigned long int**

0555	**stała ósemkowa (rozpoczyna się od 0, nie zawiera cyfr 8 i 9)**
0x10FA	**stała szesnastkowa**
0XFFFF	**stała szesnastkowa**

➢ **stała znakowa**

ciąg złożony z jednego lub więcej znaków ujętych w apostrofy:
- **wartością stałej znakowej zawierającej tylko jeden znak jest numeryczna wartość tego znaku w zbiorze znaków maszyny wykonującej program,**
- **wartość stałej wieloznakowej zależy od implementacji**

\a	**znak alarmu (bell)**
\b	**znak cofania (backspace)**
\f	**znak nowej strony (form feed)**
\n	**znak nowego wiersza (line feed)**
\r	**znak powrotu karetki (carriage return)**
\t	**znak tabulacji poziomej (horizontal tab)**
\v	**znak tabulacji pionowej (vertical tab)**
****	**znak \ (backslash)**
\?	**znak zapytania (question mark)**
\'	**znak apostrofu (single quote)**
\"	**znak cudzysłowu (double quote)**
\ooo	**liczba ósemkowa (octal value) jedna, dwie lub trzy cyfry ósemkowe**
\xhh	**liczba szesnastkowa (hexadecimal value)**

Przykłady stałych znakowych:

```
'a'   '5'    '+'  '.'  'C' '\071'
'x41' '\101' 'A'     /* te same stałe, w PC - 'A' */
'x5F' '\n'   '\t' '\r' '\\' '\"'
'AB'              /* nieprzenośne, w TC/BC - 'A' */
```

W niektórych implementacjach dostępny jest również rozszerzony typ znakowy wchar_t (głównie ze względu na języki azjatyckie):
- **w starszych, 16-bitowych:**
  `typedef char wchar_t;`
- **w nowszych, 32-bitowych:**
  `typedef unsigned short wchar_t;`

➢ **stała zmiennopozycyjna**

- **stała zmiennopozycyjna składa się z:**
  - **części całkowitej,**
  - **kropki dziesiętnej,**
  - **części ułamkowej,**
  - **litery *e* lub *E*,**
  - **wykładnika potęgi (z opcjonalnym znakiem),**
  - **opcjonalnego przyrostka typu (*f*/*F* lub *l*/*L*).**
- **część całkowitą albo część ułamkową (ale nie obie) można pominąć,**
- **kropkę dziesiętną albo część wykładniczą z literą e (ale nie obie) można pominąć**
- **typ stałej określany jest na podstawie przyrostka:**
  - ***f* lub *F* oznacza *float***
  - ***l* lub *L* oznacza *long double***
  - **brak przyrostka oznacza *double***

```
4.5f 4.5F .5F 2.F 2.e1F .2e1f      /* float */
4.5l 4.5L .5L 2.L 2.e1l .2e1L      /* long double */
0.0  4.5  .5  2.  2.e1  .2e1       /* double */
```

➢ **wyliczenia**

```
enum kolory_kart {trefl, karo, kier, pik };
```

identyfikatory na liście wyliczników są deklarowane jako stałe i mogą wystąpić wszędzie tam gdzie mogą wystąpić stałe (za wyjątkiem *#if*)

Napisy

ciąg znaków ujętych w znaki cudzysłowu,

- **typ napisu - tablica znakowa (char [])**
- **klasa pamięci - static**
- **sąsiadujące napisy są łączone w jeden napis,**
- **po ewentualnych połączeniach na końcu dodawany jest wartownik '\0'**
- **identyczne napisy moga nie być rozróżniane (jeżeli są dwa lub więcej takie same napisy generowana jest jedna kopia)**
- **napisów nie wolno modyfikować,**

- **napisy nie zawierają znaków cudzysłowu ani znaków nowego wiersza, do ich wyrażenia można stosować te same sekwencje specjalne co do stałych znakowych**
- **napisy można również tworzyć z rozszerzonego typu znakowego wchar_t, nie został określony sposób łączenia napisów z normalnych i rozszerzonych typów znakowych**

Przykłady:

```
/* łączenie łańcuchów */
"I like programming in C language"
"I" " " "like progra" "mming in ""C language"

/* sekwencje specjalne */
"\n\aI\tlike\tprogramming\tin\tC\tlanguage\n"
"I like programming in \"C language\""
```

Typy podstawowe

zakresy dla danej implementacji w pliku limits.h

➢ **Typy liczb całkowitych (całkowitoliczbowe)**

`char`	**jeden bajt, jeden znak z lokalnego zbioru znaków**
`int`	**podstawowy typ całkowity, zazwyczaj naturalny rozmiar liczb całkowitych komputera**

Kwalifikatory

1. Kwalifikatory rozmiaru (różne zakresy liczb całkowitych):

- `long`
- `short`

•skróty

`short int` ⇔ `short`
`long int` ⇔ `long`

•wymagania

`short` i `int` conajmniej 16-bitowe,
`long` conajmniej 32-bitowy
`sizeof(short) ≤ sizeof(int) ≤ sizeof(long)`

2. Kwalifikatory znaku dowolnego typu całkowitego:

- `signed`
- `unsigned`

•skróty

`unsigned int` ⇔ `unsigned`
`signed int` ⇔ `signed`

➢ **Typy liczb zmiennoprzecinkowych**

`float`	**pojedyncza precyzja**
`double`	**podwójna precyzja**
`long double`	**rozszerzona precyzja (zastapił `long float`)**

Typy i przykładowe zakresy i rozmiary

Typ	Zakres (Watcom C 11.0/32b/PC)	Rozmiar [B]
`char`		
`char`	-128 ÷ 127 albo 0 ÷ 255	1
`signed char`	-128 ÷ 127	1
`unsigned char`	0 ÷ 255	1
`short int`		
`short int`	-32768 ÷ 32767	2
`unsigned short int`	0 ÷ 65535	2
`int`		
`int`	-2147483648 ÷ 2147483647	4
`unsigned int`	0 ÷ 4294967295	4
`long int`		
`long int`	-2147483648 ÷ 2147483647	4
`unsigned long int`	0 ÷ 4294967295	4
zmiennoprzecinkowe		
`float`	$1{,}2*10^{-38} \div 3{,}37 * 10^{38}$ (6-7 cyfr znaczących)	4
`double`	$2{,}3*10^{-308} \div 1{,}67 * 10^{308}$ (15-16 cyfr znaczących)	8
`long double`	$3{,}4*10^{-4932} \div 1{,}2 * 10^{4932}$ (19 cyfr znaczących)	10

Kolejność wykonywania konwersji:

- **jeśli jeden argument jest typu *long double*, to drugi jest poddawany konwersji do typu *long double***
- **jeśli jeden argument jest typu *double*, to deugi jest poddawany konwersji do typu *double***
- **jeśli jeden argument jest typu *float*, to deugi jest poddawany konwersji do typu *float***
- **jeśli jeden argument jest typu *unsigned long int*, to deugi jest poddawany konwersji do typu *unsigned long int***
- **jeśli jeden argument jest typu *long int*, to deugi jest poddawany konwersji do typu *long int***
- **jeśli jakikolwiek argument jest typu *char* albo *short int*, ze znakiem albo bez znaku, to jest (bez zmiany wartości danej) poddawany konwersji do typu *int*, a jeśli nie jest ty wykonalne zawsze, jest poddawany konwersji do typu *unsigned int***
- **jeśli jeden argument jest typu *unsigned int*, to drugi jest poddawany konwersji do typu *unsigned int***
- **jeśli okaże się, że żadna z wymienionych konwersji nie jest konieczna, to oba argumenty są typu *int*.**

Zmienne

deklaracje zmiennych:

```
typ ident [, ident]* ;

int i;
char a, b, c;
unsigned duza_wartosc;
double masa, gestosc;
```

definicje zmiennych (deklaracja + inicjalizacja),

```
typ ident = wartość [, ident = wartość]*;

int licznik=125, suma=0;
float dokladnosc=0.001, blad=0.1;
double moc=15e6, straty=1500;
```

Ważniejsze operatory

+	dodawanie
-	odejmowanie
*	mnożenie, operator dostępu pośredniego (wyłuskanie)
/	dzielenie, dzielenie całkowite
%	dzielenie modulo
++	inkrementacja (zwiększenie o 1)
--	dekrementacja (zmniejszenie o 1)
=	przypisanie
==	równe
!=	różne
<	mniejsze
<=	mniejsze lub równe
>	większe
>=	większe lub równe

```
int a, b, c;
a = 1;
b = 2;
c = a+b;
a = b = c = 0;        /* wielokrotne podstawienie */
a = b/c; /* część całkowita z dzielenia b przez c */
a = b%c           /* reszta z dzielenia b przez c */
a++;                            /* powiększ a o 1 */
++a;                            /* powiększ a o 1 */
a=a+1;                          /* powiększ a o 1 */
```


Instrukcje pętli

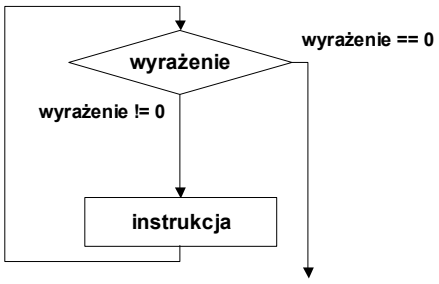
➢ pętla *while*

```
while(wyrażenie)
  instrukcja;
```

albo:

```
while(wyrażenie)
  {
    instrukcja1;
    instrukcja2;
    instrukcjan;
  }
```

- warunek jest testowany przed wejściem do pętli
- ciało pętli może nie zostać ani razu wykonane



przykład

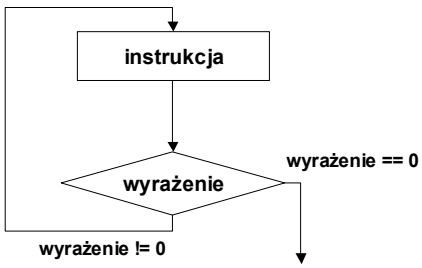
```
int n=10;
while(n>0)
{
  printf("%d\t",n);
  n--;
}
```

wyjście:

```
10   9    8    7    6    5    4    3    2    1
```


➢ pętla do-while

- warunek jest testowany <u>po</u> wykonaniu ciała pętli
- ciało pętli musi być choć raz wykonane



przykład:

```
int n=10;
do
{
  printf("%d\t",n);
  n--;
} while(n>0);
```

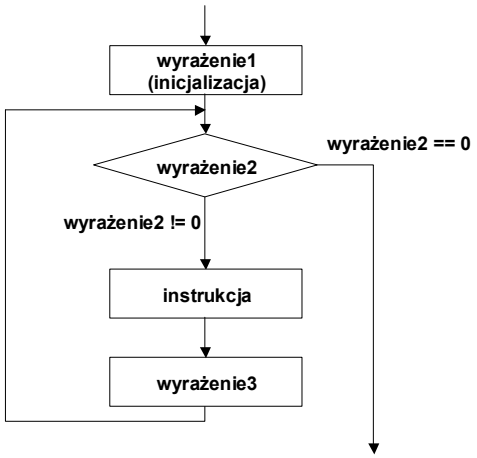
wyjście:

```
10   9    8    7    6    5    4    3    2    1
```


➢ pętla for

```
for(wyrażenie1;wyrażenie2;wyrażenie3)instrukcja;
```

- dowolne (wszystkie) wyrażenia można pominąć,
- *wyrażenie1* jest częścią inicjującą pętli, jest obliczane tylko raz; jeśli jest wyrażeniem złożonym poszczególne wyrażenia składowe rozdziela się przecinkami,
- *wyrażenie2* to warunek kontynuacji pętli, pominięcie go jest równoważne z zastąpieniem stałą różną od zera (warunek zawsze prawdziwy),
- *wyrażenie3* jest obliczane po każdym przebiegu pętli i określa zmianę stanu pętli; jeśli jest wyrażeniem złożonym poszczególne części rozdziela się przecinkami.



przykład:

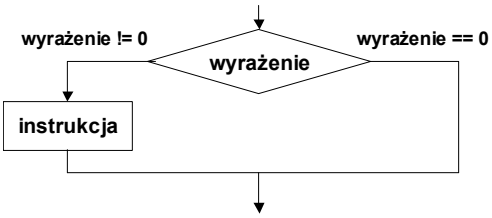
```
int i, j;
for(i=1,j=2;
    j<1000;
    i++,j*=i)printf("[%d,%d]\t",i,j);
  return 0;
}
```

wyjście:

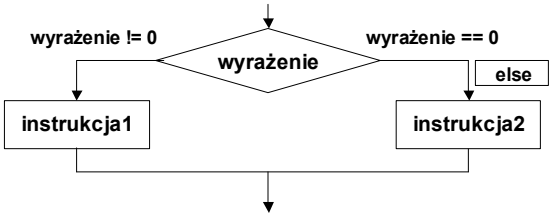
```
[1,2]   [2,4]   [3,12]   [4,48]   [5,240]
```

- **instrukcja warunkowa**

```
if(wyrażenie)instrukcja;
```




```
if(wyrażenie)instrukcja;
else instrukcja;
```




```
if((n % 2)==0)printf("parzysta");
else printf("nieparzysta");
```