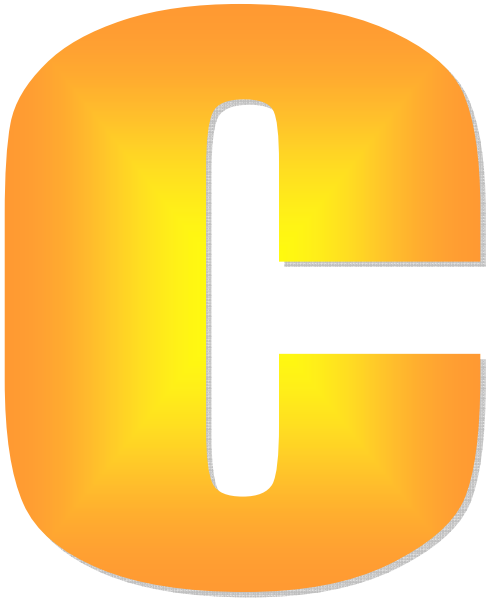
Andrzej Miszczyk

Wprowadzenie do programowania w języku C



Wersja dla uczniów i nauczycieli
Zespołu Szkół Elektrycznych w Kielcach

KIELCE 2007

Spis treści

Wstęp

Język C jest popularnym językiem programowania o szerokich zastosowaniach, pozwalającym stosować techniki programowania proceduralnego, strukturalnego oraz modularnego. Nie jest skomplikowany, a przy tym bardzo elastyczny.

Światowe standardy C (ISO i ANSI) są powszechnie stosowane przez twórców kompilatorów, toteż kod źródłowy programów w C może być łatwo przenoszony między różnymi komputerami. Programy napisane w C można kompilować także przy pomocy kompilatorów języka C++. Wynika to z faktu, że C++ powstał na bazie C i zachował z nim zgodność (poza małymi wyjątkami). Większość kompilatorów C++ posiada możliwość przełączenia w tryb standardowego C.

Omawiane poniżej elementy języka C stanowią niewielki wycinek jego możliwości, ale wystarczający do pisania większości algorytmów przetwarzających dane w postaci liczbowej i tekstowej.

Wszystkie przykładowe programy pracują w trybie tekstowym i mogą być uruchamiane w powłoce poleceń Windows lub dowolnej tekstowej powłoce systemów typu Unix. Pisząc je zachowano zgodność ze standardem ANSI C z 1990 roku, toteż czytelnik do ich uruchomienia może wykorzystywać nawet starsze kompilatory (np. Turbo C). Oczywiście wygodniejsze będzie użycie jakiegoś nowszego zintegrowanego środowiska programistycznego, na przykład w systemie MS Windows może to być darmowe środowisko Dev-C++, oparte o kompilator bazujący na GCC (GNU Compiler Collection).

Systemy operacyjne stosują różne standardy kodowania liter charakterystycznych dla języka polskiego. Toteż, aby zwiększyć czytelność i przenośność kodu źródłowego rezygnowano z używania ich w przykładach. Aby ułatwić omawianie elementów języka, ponumerowano (`1:, 2:, 3:,...`) wiersze kodu źródłowego, toteż należy pamiętać aby przy przepisywaniu kodu do pliku numerację wierszy pominąć.

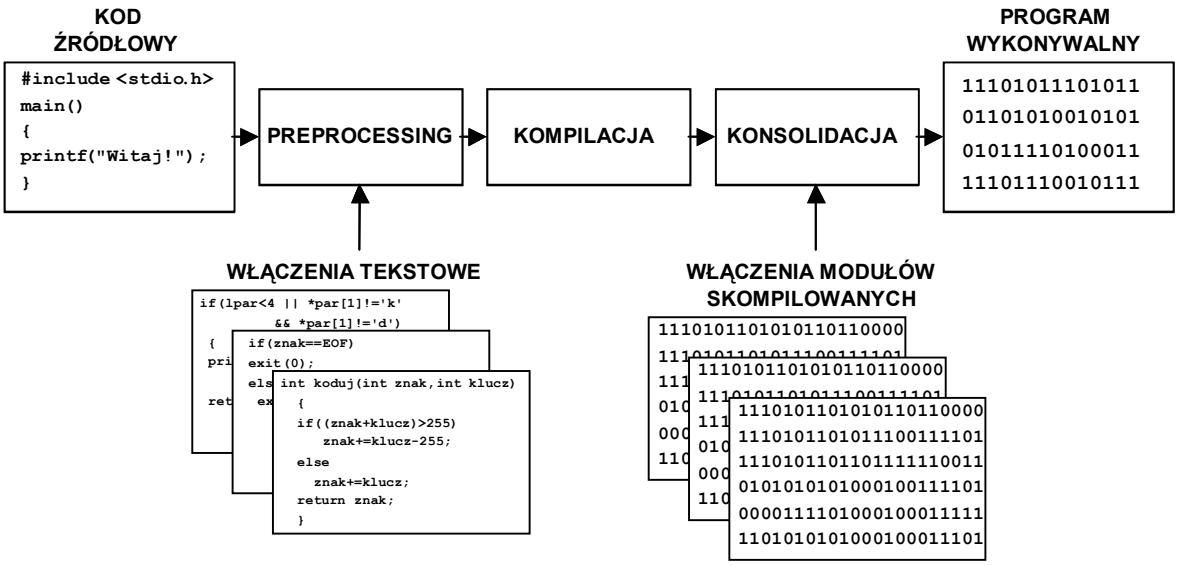
Proces tworzenia programu

Proces tworzenia programu w języku C można podzielić na następujące etapy:

1. Napisanie kodu źródłowego w dowolnym edytorze tekstu i zapisanie go w postaci niesformatowanego pliku tekstowego o nazwie zakończonej „**`.c`**”, na przykład **`pierwszy.c`**.
2. Kompilacja i konsolidacja pliku źródłowego przy pomocy narzędzi programistycznych języka C, dostępnych w systemie.

W fazie wstępnej kompilacji kodu źródłowego (ang. preprocessing), pracuje preprocesor języka wykonując: makrorozwinięcia, włączenia do programu innych plik źródłowych (tekstowych) oraz kompilację warunkową. Preprocesor interpretuje umieszczone w kodzie źródłowym dyrektywy poprzedzone znakiem #. Po zakończeniu pracy preprocesora uruchamiana jest właściwa kompilacja, której efektem jest powstanie programu w kodzie pośrednim (ang. object code), zwykle w pliku o nazwie kończącej się „**`.obj`**” lub „**`.o`**”. Następnie w fazie konsolidacji do programu skompilowanego dołączane są potrzebne moduły (skompilowane wcześniej), dołączane w ten sposób są na przykład moduły bibliotek standardowych. Wynikiem konsolidacji jest program wykonywalny, który można uruchomić z poziomu systemu operacyjnego komputera.

Podczas procesu tworzenia programu wykonywalnego mogą wystąpić różnego rodzaju błędy (ang. errors). W czasie kompilacji mogą wystąpić błędy syntaktyczne, związane z nieprzestrzeganiem reguł języka (ang. compiler errors). Natomiast przy konsolidacji błędy (ang. linker errors) mogą być związane z niekompletnością dołączanych modułów lub wręcz ich brakiem. Błędy mogą także objawić się dopiero po uruchomieniu programu, tzw. błędy wykonania (ang. runtime errors). Powstają one w wyniku niewłaściwej implementacji algorytmów lub problemów z dostępem do zasobów komputera. Do błędów tego typu należy na przykład: próba dzielenia przez zero, brak pamięci dla zmiennych dynamicznych, niedostępność dysku, itp.



Rysunek 4.1. Proces tworzenia programu.

Podczas kompilacji i konsolidacji informacje o błędach wyświetlane są na ekranie lub zapisywane są do plików tekstowych. Powstaje lista zawierająca dla każdego błędu jego krótki opis oraz orientacyjną lokalizację w kodzie źródłowym (numer wiersza).

Oprócz informacji o błędach kompilator może wypisywać ostrzeżenia (ang. warnings), wskazując miejsca w kodzie programu, które mogą być przyczyną błędów wykonania. Przykładowo, można spotkać następujące ostrzeżenia:

- użyto zmiennej bez przypisanej początkowej wartości,
- przypisana wartość zmiennej nigdy nie jest wykorzystywana w programie,
- użyto nie zalecanej (przestarzałej) funkcji bibliotecznej,
- dzielenie przez zero.

Przykład implementacji prostego programu

Program w języku C składa się z funkcji oraz zmiennych. Funkcje zawierają instrukcje, z których budowany jest algorytm działania, natomiast zmienne przechowują przetwarzane wartości. Poniżej podano kod źródłowy programu `p1.c` obliczającego pole powierzchni koła.

```
 1:    /* Program p1.c */
 2:    /* Obliczanie pola kola */
 3:
 4:    #include <stdio.h>
 5:
 6:    main()
 7:    {
 8:    float promien, pole;
 9:    promien = 5.25;
10:    pole = 3.1415926 * promien * promien;
11:    printf("Pole kola = %f ", pole);
12:    }
```

Budowa programu p1.c

Wiersze 1 i 2 to komentarze, które ignorowane są przez kompilator. Komentarz zaczyna się znakami `/*`, a kończy się znakami `*/`, może zajmować wiele wierszy kodu.

Wiersze puste lub zawierające znaki niewidoczne (odstępy, tabulacje, itp.) są traktowane przez kompilator podobnie jak komentarze, używa się ich w celu poprawienia czytelności kodu.

Wiersz 4 zawiera instrukcję dla preprocesora `#include`, która powoduje włączenie w miejscu jej wystąpienia zawartości pliku `stdio.h`. Plik ten jest nagłówkiem biblioteki standardowej, zawierającej funkcje wejścia-wyjścia.

Wiersz 6 zawiera definicję funkcji `main()`, od której rozpoczyna się wykonywanie programu, zatem każdy program powinien zawierać funkcję o nazwie „main". Wszystkie instrukcje wykonywane w ramach danej funkcji zgrupowane są w jeden blok nawiasami klamrowymi { i } (wiersz 7 i 12) i stanowią tzw. ciało funkcji.

Wiersze od 8 do 11 zawierają cztery instrukcje zawarte w funkcji main, wykonywane liniowo (jedna po drugiej).

```
float promien, pole;
```

Instrukcja deklaruje dwie zmienne rzeczywiste typu `float` (pojedyncza precyzja) o nazwach `promien` i `pole`.

```
promien = 5.25;
```

Instrukcja przypisania, nadająca zmiennej `promien` wartość `5.25`. Należy pamiętać, że separatorem dziesiętnym w języku C jest kropka „`.`". Znak „=" to operator przypisania, czyta się go „przypisz".

```
pole = 3.1415926 * promien * promien;
```

Instrukcja przypisania, nadająca zmiennej `pole` wartość wyrażenia obliczonego po prawej stronie operatora przypisania. Znak „ `*` " jest operatorem mnożenia.

```
printf("Pole kola = %f ", pole);
```

Instrukcja wywołania funkcji `printf()`, z dwoma argumentami. Pierwszy z nich to łańcuch tekstowy `"Pole kola = %f "`, drugi to zmienna `pole`. Funkcja `printf()` pochodzi ze standardowej biblioteki stdio i służy do wyprowadzania danych na ekran. Wypisze na ekranie tekst zawarty w pierwszym argumencie, zastępując sekwencję `%f` wartością zmiennej `pole`.

Uruchomienie programu

Sposób wykonania programu zależy od używanego systemu. Tutaj omówiono translację kodu źródłowego przy wykorzystaniu darmowego kompilatora gcc, pracującego z poziomu wiersza poleceń systemu.

W systemie operacyjnym Linux można zapisać kod źródłowy w pliku o nazwie zakończonej „`.c`", a potem przetłumaczyć go wywołując kompilator `gcc`. Poniżej umieszczono najprostszą formę wywołania kompilatora `gcc` dla programu zapisanego w pliku `p1.c`.

```
gcc p1.c
```

Jeżeli program nie zawiera błędów, zostanie skompilowany i powstanie program wykonywalny w pliku o nazwie `a.out`. Program `a.out` można wykonać podając jego nazwę jako polecenie w wierszu poleceń, efektem jego działania będzie wyświetlenie na ekranie napisu:

```
Pole kola = 86.590149
```

Można także otrzymać program wykonywalny o innej nazwie niż `a.out`, na przykład `polekola`. Wtedy polecenie wywołania kompilatora gcc wyglądałoby następująco:

```
gcc -o polekola p1.c
```

Kompilator gcc jest także dostępny w wersji dla MS Windows i można z niego korzystać w taki sam sposób jak w systemie Linux. Oczywiście w Windows programy wykonywalne będą posiadały rozszerzenie nazwy „`.exe`”.

Zestaw znaków języka C

Zestaw znaków języka C umożliwia tworzenie wszystkich elementów kodu źródłowego, składa się z następujących elementów:

- duże litery (od `A` do `Z`),
- małe litery (od `a` do `z`),
- cyfry (od `0` do `9`),
- znaki specjalne:
  `! * + \ " < # ( = | { > % ) ~ ; } / ^ - [ : , ? & _ ] ' .`
  i znak odstępu (spacja).

Istnieją także pewne kombinacje znaków tworzące tzw. sekwencje specjalne. Umożliwiają one zapis niewidocznych znaków w operacjach wyjścia, jak na przykład:

`\b` znak cofania,
`\n` znak nowego wiersza,
`\t` znak tabulacji.

Słowa kluczowe

Słowa kluczowe (zarezerwowane, zastrzeżone) stanowią zestaw słów o ustalonym w standardzie języka znaczeniu i nie mogą być inaczej używane. Nie wolno na przykład użyć słowa kluczowego jako nazwy zmiennej. Poniżej umieszczono zestaw słów kluczowych należących do standardu języka.

```
auto       double   int        struct
break      else     long       switch
case       enum     register   typedef
char       extern   return     union
const      float    short      unsigned
continue   for      signed     void
default    goto     sizeof     volatile
do         if       static     while
```

Część kompilatorów wprowadza dodatkowo własne słowa kluczowe, na przykład:

```
asm   ada   fortran   near   far   pascal   huge
```

Komentarze

Komentarze to fragmenty tekstu w kodzie źródłowym, które ignorowane są przez kompilator. Komentarz zaczyna się znakami `/*`, a kończy się znakami `*/`, może zajmować wiele wierszy kodu.

```
/* Taka linia jest komentarzem w jezyku C */
/* Komentarz
          moze
            zajmowac
                 kilka
                   wierszy */
```

Komentarze nie mogą być zagnieżdżone oraz nie mogą wystąpić w napisach i stałych znakowych. Zwykle komentarze opisują kod źródłowy, oprócz tego można je wykorzystać do wyłączania fragmentu programu z kompilacji.

Wiersze puste lub zawierające znaki niewidoczne (odstępy, tabulacje, itp.) traktowane są przez kompilator podobnie jak komentarze, wykorzystywane są w celu poprawienia czytelności kodu.

Podstawowe typy danych

Podstawowe typy danych języka C mogą przechowywać liczby całkowite i rzeczywiste, należą do nich: `int`, `char`, `float` oraz `double`.

Typ `char` reprezentuje liczbę całkowitą pozwalającą na zapis jednego znaku (np. litery alfabetu) w systemie, typowo zajmuje 1 bajt. Typ `int` wykorzystywany jest do zapisu liczb całkowitych. Typy `float` i `double` reprezentują liczby rzeczywiste, przy czym `double` zajmuje dwa razy więcej pamięci i pozwala na zapis danych z większą dokładnością (podwójna precyzja). W przypadku skomplikowanych obliczeń na liczbach rzeczywistych dane wynikowe zwykle zapisywane są jako `double`.

Nazwa typu	Opis
`char`	Jeden bajt, jeden znak w kodzie ASCII.
`int`	Typ całkowity.
`float`	Typ zmiennopozycyjny (liczba rzeczywista) pojedynczej precyzji.
`double`	Typ zmiennopozycyjny (liczba rzeczywista) podwójnej precyzji.

Zakres wartości, można zmieniać poprzedzając nazwy odpowiednich typów słowami kluczowymi: `short` (liczba krótka), `long` (liczba długa), `signed` (liczba ze znakiem), `unsigned` (liczba bez znaku). Fizyczny rozmiar i zakres wartości zależy od używanego kompilatora, poniżej podano przykładowe dane dla kompilatora „Borland C++ Builder 6”.

Nazwa typu	Rozmiar w bitach	Zakres wartości
`unsigned char`	8	$0 \le X \le 255$
`char`	8	$-128 \le X \le 127$
`short int`	16	$-32{,}768 \le X \le 32{,}767$
`unsigned int`	32	$0 \le X \le 4{,}294{,}967{,}295$
`int`	32	$-2{,}147{,}483{,}648 \le X \le 2{,}147{,}483{,}647$
`unsigned long`	32	$0 \le X \le 4{,}294{,}967{,}295$
`enum`	32	$-2{,}147{,}483{,}648 \le X \le 2{,}147{,}483{,}647$
`long`	32	$-2{,}147{,}483{,}648 \le X \le 2{,}147{,}483{,}647$
`float`	32	$1.18E\text{-}38 < \lvert X\rvert < 3.40E38$
`double`	64	$2.23E\text{-}308 < \lvert X\rvert < 1.79E308$
`long double`	80	$3.37E\text{-}4932 < \lvert X\rvert < 1.18E4932$

Do typów całkowitych należy także typ wyliczeniowy `enum`, pozwalający na przypisanie kolejnych wartość całkowitych odpowiadającym im napisom tekstowym.

Kolejnym typem jest `void`, który oznacza brak wartości. Jest on używany jako typ funkcji, która nie zwraca żadnej wartości oraz jako symbol braku parametrów przy deklaracji funkcji.

W języku C stosowany jest także typ wskaźnikowy, służący do reprezentacji adresów zmiennych w pamięci komputera.

Stałe

Stałe programowe możemy podzielić na dwie zasadnicze grupy:

1. numeryczne (całkowite i rzeczywiste),
2. tekstowe.

Standard języka C definiuje cztery rodzaje stałych programowych: stałe całkowite, stałe rzeczywiste, stałe znakowe oraz stałe napisowe.

Stałe całkowite

Jeżeli w kodzie programu pojawia się liczba bez kropki dziesiętnej traktowana jest jako stała całkowita typu `int`. W przypadku, gdy nie mieści się w zakresie `int`, traktowana jest jako stała typu `long` (inaczej `long int`). Można także wymusić typ stałej całkowitej dodając na jej końcu odpowiednie oznaczenie literowe, jak w poniższej tabeli.

Oznaczenie	Opis	Przykłady
brak	Stała typu `int`	`123 4567`
`l` lub **`L`**	Stała typu `long`	`123l 4567L`
`u` lub **`U`**	Stała typu `unsigned`	`123u 4567U`
`ul` lub **`UL`**	Stała typu `unsigned long`	`123ul 4567UL`

Stałe całkowite mogą być przedstawiane w trzech systemach liczbowych: dziesiętnym, ósemkowym i szesnastkowym.

Stałe dziesiętne
Uwaga! Jeżeli stała składa się z dwóch lub więcej znaków, pierwszym znakiem nie może być zero.
Przykłady**:**

```
0  10  123  -100  +213  3429
```

Stałe ósemkowe
Pierwszą cyfrą musi być zero.
Przykłady:

```
0  0234  -0435  +0657
```

Stałe szesnastkowe
Zaczynają się od znaków 0x lub 0X.
Przykłady:

```
0x 0x123 -0X4A  +0xff 0xFF
```

Stałe rzeczywiste (zmiennopozycyjne)

Stała numeryczna jest traktowana jako rzeczywista, gdy zawiera kropkę dziesiętną (np. `1.23`), wykładnik (np. `2e4`) lub oba te elementy (np. `2.23e-3`). Domyślnym typem stałej rzeczywistej jest `double`, chyba że dodano odpowiednie oznaczenie na końcu.

Oznaczenie	Opis	Przykłady
brak	Stała typu `double`	`123.23 5.2e-2`

Oznaczenie	Opis	Przykłady
f lub **F**	Stała typu `float`	`123.23f 12.3F`
l lub **L**	Stała typu `long double`	`123.2l 5678.24L`

Przykłady:

```
0.  2.  0.2  234.12  -86.537 +12.5  2.3e3  4.4e-5  34.33E12
```

Stałe znakowe

Stałą znakową tworzy pojedynczy znak ujęty w apostrofy (np. `'A'`) i reprezentuje wartość całkowitą odpowiadającą kodowi znaku w systemie. Zwykle wykorzystywany jest kod ASCII w formie podstawowej (128 znaków –kody od 0 do 127) lub rozszerzonej (256 znaków –kody od 0 do 255)

Stałe znakowe można wykorzystywać w obliczeniach na równi z liczbami całkowitymi, ale najczęściej wykorzystywane są do porównań z innymi znakami.

Przykłady stałych znakowych i ich wartości:

Stała	**Wartość**
`'a'`	97
`'A'`	65
`'+'`	43
`'9'`	57

Stałe znakowe mogą być reprezentowane w postaci sekwencji specjalnych, czyli zestawu kilku znaków oznaczających jeden znak. Można w ten sposób odwoływać się do znaków niewidocznych bez (np. znak nowego wiersza). Pierwszym znakiem sekwencji specjalnej jest ukośnik „ \ ”.

Sekwencje specjalne	Znaczenie
\a	alarm (bell, BEL)
\b	cofnięcie karetki (backspace, BS)
\e	znak ucieczki (escape, ESC)
\f	nowa strona (form feed, FF)
\n	nowa linia (new line, NL)
\r	powrót karetki (carriage return, CR)
\t	tabulacja pozioma (horizontal tab, HT)
\v	tabulacja pionowa (vertical tab, VT)
****	ukośnik (backslash)
\"	cudzysłów (double quote)
\'	apostrof (single quote)
\ooo	ooo = wartość ósemkowa kodu znaku
\xhh	hh = wartość szesnastkowa kodu znaku

Przykłady stałych znakowych:

```
'A'  '#'  'x'  ' '  '\n' '\"' '\113'  '\x4B'
```

Stałe napisowe

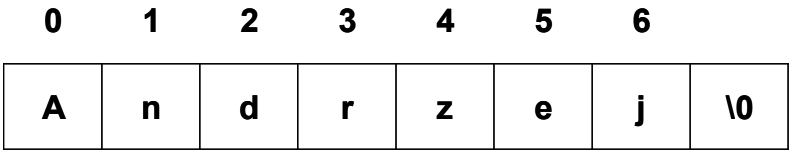
Stałą napisową tworzy ciąg zera lub więcej znaków ograniczony znakami cudzysłowu. Stałe napisowa nazywane są także: napisami, stałymi tekstowymi lub stałymi łańcuchowymi (łańcuchy znaków).

Przykłady stałych napisowych:

```
"Ala ma kota"  "2+2=4"  "linia1\n linia2\n linia3"
"Pole= "       ""       "A"       "Andrzej"
```

Jak widać na powyższych przykładach stałe napisowe mogą zawierać sekwencje specjalne oraz odstępy (spacje). Umieszczenie obok siebie dwóch znaków cudzysłowu, tworzy napis pusty ("").

Od strony technicznej stała napisowa jest tablicą znaków, której koniec wyznacza znak `'\0'` (null). Na przykład napis `"Andrzej"` składa się z siedmiu znaków, indeksowanych od 0 do 6 oraz znaku `'\0'`.



Stała napisowa składająca się z jednego znaku nie jest równoważna stałej znakowej, bowiem napis zakończony jest znakiem`'\0'`. Na przykład `'A'` jest różne od `"A"`, gdyż `"A"` składa się technicznie ze znaków `'A'` i `'\0'`.

Wyrażenia i podstawowe operatory

Najprostszym wyrażeniem może być element danych reprezentujący określoną wartość na przykład: zmienna, stała, znak, liczba. Bardziej rozbudowane wyrażenia można budować łącząc wymienione elementy danych przy pomocy operatorów. Podstawowymi operatorami języka C są:

- operatory przypisania,
- operatory arytmetyczne,
- operatory porównania,
- operatory logiczne,
- operatory wskazywania.

Poniższe tabele przedstawiają wybrane operatory języka C. Operatory w poszczególnych tabelach mają taki sam priorytet, tabele uporządkowano od najwyższego do najniższego priorytetu.

Operator	Opis
`( )`	nawiasy okrągłe
`[ ]`	nawiasy klamrowe
`->`	wskazywania
`.`	wskazywania (kropka)

Operator	Grupa	Opis
`!`	logiczne	negacja
`++`	unarne	inkrementacja
`--`	unarne	dekrementacja
`-`	unarne	minus jednoargumentowy
`*`	wskazywania	adresowanie pośrednie
`&`	wskazywania	operator adresu

Operator	Grupa	Opis
*	arytmetyczne	mnożenie
/	arytmetyczne	dzielenie
%	arytmetyczne	reszta z dzielenia

Operator	Grupa	Opis
+	arytmetyczne	dodawanie
-	arytmetyczne	odejmowanie

Operator	Grupa	Opis
<	porównania	mniejsze
<=	porównania	mniejsze lub równe
>	porównania	większe
>=	porównania	większe lub równe

Operator	Grupa	Opis
==	porównania	równe
!=	porównania	różne

Operator	Grupa	Opis
&&	logiczne	iloczyn logiczny (AND)

Operator	Grupa	Opis
||	logiczne	suma logiczna (OR)

Operator	Grupa	Opis
=	przypisania	przypisz

Przykłady wyrażeń:

```
x=3*(2+y)    liczba >= 0    ++licznik    pole= bok1 * bok2
```

Zmienne

Zmienna odnosi się do określonej wartości umieszczonej w pamięci komputera, charakteryzują ją następujące właściwości:

- nazwa (identyfikator),
- typ przechowywanych danych,
- wartość,
- adres w pamięci komputera.

Nazwa zmiennej to ciąg znaków reprezentujący jej aktualną wartość w kodzie programu. Przy nadawaniu nazw zmiennym należy kierować się następującymi zasadami:

- Nazwy tworzymy z liter i cyfr.
- Rozróżniane są duże i małe litery.
- Znak podkreślenia „_” traktowany jest jak litera.
- Pierwszym znakiem musi być litera.
- Nazwa nie może zawierać odstępu.
- Nazwa nie może zawierać znaków charakterystycznych dla języka polskiego.
- Nazwa nie może być słowem kluczowym języka C.

Przykłady poprawnych nazw zmiennych:

```
rata            Rata      wspolczynnik1   LICZBY
pole_trapezu    x12       _b5             NumerPracownika
```

Deklaracja zmiennej

Przed użyciem zmiennej należy ją zadeklarować. Deklaracja składa się z podania typu zmiennej oraz podania jednej lub wielu nazw zmiennych (oddzielonych przecinkami) na przykład

```
int a;
int numer;
float bok1, bok2, pole;
char litera;
```

W języku C zmienna po deklaracji nie posiada przypisanej standardowej wartości, przechowuje wartość przypadkową. Aby przy deklaracji nadać zmiennej wartość początkową używając operatora przypisania „=” np.

```
int a=10;
int numer=1200;
float bok1=1, bok2=12.5, pole=10;
char litera='A';
```

Jeżeli zamierza się raz przypisaną wartość zmiennej zablokować przed zmianami w trakcie działania programu, należy jej deklarację poprzedzić słowem kluczowym `const`.

```
const float pi= 3.14159265;
```

Zmienne lokalne i globalne

Jeżeli zmienną zadeklarowano wewnątrz dowolnej funkcji (np. main()), to zmienną taką nazywa się lokalną, zaś zasięg widoczności nazwy tej zmiennej ograniczony jest blokiem instrukcji, w którym wystąpiła. Zmienna globalna musi być zadeklarowana na zewnątrz wszystkich funkcji i jest ona widoczna w całym programie.

Jeżeli w danym miejscu programu wystąpią dwie zmienne o takiej samej nazwie: jedna globalna oraz druga lokalna, to pod tą nazwą będzie dostępna zmienna lokalna. Ilustruje to poniższy przykład, w którym zmienna globalna `a` zostaje przesłonięta przez zmienną lokalną `a`.

```
/* Program p8.c */
#include<stdio.h>

int a=10; /* zmienna globalna */

int main()
{
  int b=20; /* zmienna lokalna */
  printf(" a + b = %d\n", a+b);
    {
    int a=30; /* zmienna lokalna */
    printf(" a + b = %d\n", a+b);
    }
return 0;
}
```

Instrukcje

Instrukcjami nazywane są fragmenty kodu źródłowego, które realizują jakąś operację w trakcie działania programu. Technicznie instrukcja języka C to ciąg znaków, zgodny z regułami języka, zakończony średnikiem „`;`”. Umieszczenie w programie pojedynczego średnika tworzy tzw. instrukcję pustą, która nic nie wykonuje, lecz może zwiększyć kod wynikowy programu. Poniżej pokazano kod źródłowy, który zawiera trzy instrukcje puste.

```
main()
{
; ; ;
}
```

Instrukcja przypisania

Operatora przypisania można użyć do zmiany wartości istniejącej zmiennej, np. w celu przypisania jej wyniku obliczanego wyrażenia arytmetycznego.

```
float x=3.33,a=2, b=1, c=4;
double y;
y = a*x*x + b*x + c;
```

Blok instrukcji (instrukcja złożona)

Blok instrukcji składa się z ciągu instrukcji i deklaracji umieszczonych między nawiasami klamrowymi.

```
{ ciąg instrukcji i deklaracji }
```

Stosując bloki instrukcji grupuje się instrukcje tworzące funkcjonalną całość, może to być treść funkcji lub złożona instrukcja sterująca (warunkowa lub pętla). Bloki instrukcji mogą być w sobie wielokrotnie zagnieżdżane.

```
{
/* Pierwszy blok */
int x=10;
x=x-y;
if(x>0)
  {
  /* Drugi blok */
  printf("x > 0");
  x=y;
  }
printf("%d\n", x);
}
```

Instrukcja decyzyjna if-else

Instrukcja `if-else` stosowana jest przy podejmowaniu decyzji w programie. Najprostsza jej postać wygląda następująco:

```
if(wyrażenie)
    instrukcja1 (lub blok instrukcji)
```

Wykonanie powyższej instrukcji zaczyna się od obliczenia wartości wyrażenia umieszczonego w nawiasach, następnie w przypadku, gdy wyrażenie jest prawdziwe wykonaniu instrukcja 1. Należy zaznaczyć, że w języku C wartość wyrażenia równa zero traktowana jest jako fałszywa, a każda inna jako prawdziwa.

```
if(x > 0)      /* jeżeli zmienna x jest większa od zera */
   x=x-1;      /*    to zmniejsz x o jeden              */
```

Instrukcję można rozbudować o blok `else` realizowany w przypadku nieprawdziwości wyrażenia, czyli wykonana zostanie wtedy instrukcja 2.

```
if(wyrażenie)
    instrukcja1 (lub blok instrukcji)
else
    instrukcja2 (lub blok instrukcji)
```

Po bloku `if` można umieścić dowolną ilość bloków `else-if`, dających możliwość realizacji decyzji wielowariantowych. Realizacja takiej instrukcji polega na kolejnym obliczeniu wartości wyrażeń i wykonaniu instrukcji odpowiadającej pierwszemu prawdziwemu wyrażeniu. W przypadku, gdy żadne z wyrażeń nie jest prawdziwe wykonywany jest blok `else`.

```
if(wyrażenie1)
    instrukcja1 (lub blok instrukcji)
else if(wyrażenie2)
    instrukcja2 (lub blok instrukcji)
else if(wyrażenie3)
    instrukcja3 (lub blok instrukcji)
else
    instrukcja4 (lub blok instrukcji)
 instrukcja
```

Poniższy program ilustruje trzywariantową postać instrukcji `if-else`.

```
/* Program p4.c                  */
/* Program sprawdza znak liczby  */
/* wczytanej z klawiatury        */
#include <stdio.h>

main()
{
  int liczba;
  printf("Wprowadz liczbe: ");
  scanf("%d",&liczba); /* Wczytanie liczby */

  if( liczba > 0 )
    printf("Liczba jest dodatnia! \n");
  else if( liczba < 0 )
    printf("Liczba jest ujemna! \n");
  else
     printf("Liczba jest rowna zero! \n");

  printf("Koniec programu \n");
}
```

Pętla while

Instrukcje pętli pozwalają na cykliczne wykonywanie instrukcji. Formalna postać pętli `while` wygląda następująco:

```
while(wyrażenie)
    blok instrukcji
```

Blok instrukcji wykonywany jest tak długo dopóki wartość wyrażenia jest prawdziwa. Wyrażenie obliczane jest na początku każdego cyklu pętli. Blok instrukcji musi zawierać instrukcje zmieniające wartość wyrażenia, dające możliwość zakończenia pętli.

Poniższy program zamienia długość podaną w metrach na stopy, w zakresie od 0 do 10.

```
/* Program p5.c  */
#include <stdio.h>
main()
{
  printf("METRY \t STOPY\n");
  float stopy;
  int metry=1;
  while( metry <= 10 )
      {
      stopy = metry * 3.281;
      printf("%d \t %f \n", metry, stopy);
      metry = metry +1;
      }
  printf("Koniec programu \n");
}
```

Pętla for

Pętla `for` jest drugą postacią pętli, różniącą się od pętli `while` sposobem zapisu.

```
for(wyrażenie1; wyrażenie2; wyrażenie3)
    blok instrukcji
```

Wyrażenia w nawiasach sterują wykonaniem pętli:

wyrażenie 1	-określa wartości początkowe,
wyrażenie 2	-ustala warunek kontynuacji pętli.
wyrażenie 3	-zawiera instrukcje zmieniające wartości zmiennych sterujących pętlą.

Jeżeli zamiast pętli `while` we wcześniejszym programie użyto by pętli `for`, wyglądałby następująco:

```
/* Program p6.c  */
#include <stdio.h>
main()
{
  printf("METRY \t STOPY\n");
  float stopy;
  int metry;
  for( metry=1; metry <= 10; metry=metry+1)
      {
      stopy = metry * 3.281;
      printf("%d \t %f \n", metry, stopy);
      }
  printf("Koniec programu \n");
}
```

Pętla do-while

Blok instrukcji pętli `do-while` będzie wykonywany tak długo, dopóki wyrażenie warunkowe będzie prawdziwe. Z uwagi na to, że sprawdzenie warunku następuje po wykonaniu bloku instrukcji, pętla ta wykona się przynajmniej raz.

```
do
   blok instrukcji
while(wyrażenie)
```

Instrukcja zaniechania break

Instrukcja zaniechania daje możliwość przerwania wykonywania pętli w dowolnej fazie i kontynuację pracy programu od instrukcji następującej po pętli.

```
break;
```

Instrukcja kontynuacji continue

Instrukcja kontynuacji powoduje przerwanie wykonywania aktualnego kroku pętli i wykonanie następnego kroku pętli od początku.

```
continue;
```

Instrukcja wyboru switch

Instrukcja `switch` wykorzystywana jest do obsługi wielowariantowych sytuacji decyzyjnych. Może zastąpić rozbudowaną instrukcję `else-if`, gdy wartość wyrażenia porównywana jest z szeregiem wartości stałych.

```
switch(wyrażenie)
   {
   case wyrażenie_stałe1 : instrukcje
                           break;   /*opcjonalne*/
   case wyrażenie_stałe2 : instrukcje
                           break; /*opcjonalne */
   case wyrażenie_stałe3 : instrukcje
                           break; /*opcjonalne */
   default : instrukcje
   }
```

Jeżeli wartość wyrażenia równa jest jednemu z wyrażeń stałych, to wykonywana jest odpowiadająca mu lista instrukcji. Umieszczenie na końcu danej listy instrukcji `break`, powoduje przerwanie sprawdzania następnych pozycji wyboru. Instrukcje umieszczone po słowie `default` wykonywane są w przypadku przejścia przez wszystkie sekcje `case`. Poniżej umieszczono program, w którym wykorzystano instrukcję switch do porównania liczby wczytanej z klawiatury do kilku wartości całkowitych.

```
/* Program p7.c */
#include <stdio.h>
main()
{
  int liczba;
  printf("Wprowadz liczbe z zakresu od 0 do 3 : ");
```

```
  scanf("%d",&liczba);
  switch (liczba)
    {
    case 0 :
           printf("Wczytano 0");
           break;
    case 1 :
           printf("Wczytano 1");
           break;
    case 2 :
           printf("Wczytano 2");
           break;
    case 3 :
           printf("Wczytano 3");
           break;
    default:
           printf("Wczytano liczbe z poza zakresu");
           break;
    }
  printf("\nKoniec programu \n");
}
```

Instrukcja powrotu return

Instrukcja powrotu powoduje zakończenie wykonywania funkcji. Instrukcja `return` może być zapisana w dwóch postaciach:

```
return;

return wyrażenie;
```

Postać pierwsza powoduje opuszczenie funkcji bez wyprowadzania na zewnątrz żadnych wartości, natomiast postać druga zwraca wartość równą wartości wyrażenia. Instrukcja powrotu może być wykorzystana do zwrócenia wartości do systemu przy zakończeniu programu. Należy wtedy nazwę funkcji `main()` poprzedzić typem zwracanej wartości (np. `int`) i w miejscu zakończenia programu wpisać instrukcję powrotu. Przyjmuje się, że wartość zero zwracana przez program oznacza poprawne zakończenie jego działania.

```
int main()
    {
    /* instrukcje */
    /* instrukcje */
    /* instrukcje */

    return 0;
    }
```

Standardowe wejście i wyjście programu

Przy tworzeniu programu często istnieje konieczność komunikacji z otoczeniem zewnętrznym. Użytkownik komunikuje się z programem poprzez sprzęt działający pod kontrolą systemu operacyjnego. Dla komputerów osobistych standardowym urządzeniami wykorzystywanymi do komunikacji z użytkownikiem jest klawiatura (standardowe wejście) i ekran monitora (standardowe wyjście).

W raz z kompilatorem języka C dostarczany jest szereg bibliotek standardowych. Wśród nich znajduje się biblioteka „stdio”, zawierająca funkcje wejścia i wyjścia. Aby mieć do nich dostęp należy

na początku programu (przed funkcją main()) dopisać dyrektywę preprocesora dołączającą nagłówek biblioteki stdio.

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

Funkcja printf()

Funkcja printf() należy do funkcji wyjścia, służy do wypisywania danych w postaci tekstu na standardowym wyjściu (np. ekranie komputera). Funkcja ta może mieć jeden lub więcej argumentów. Pierwszym argumentem jest napis, który może zawierać sekwencje specjalne np. instrukcja

```
printf(" Programowanie jest proste, \n szczegolnie w jezyku C.");
```

wypisze na ekranie dwa wiersze tekstu

```
Programowanie jest proste!
Szczegolnie w jezyku C!
```

Przy pomocy printf() można wypisywać wartości zmiennych lub wyrażeń. Pierwszy argument może zawierać tzw. specyfikacje przekształcenia zaczynające się od znaku „%", w miejsce których na ekranie wklejane są wartości kolejnych argumentów, sformatowanych według wzorca umieszczonego po znaku „%". Ilustruje to poniższy program, w którym użyto dwóch specyfikatorów przekształceń:

- %d -dla zmiennych całkowitych (int),
- %f -dla zmiennych zmiennoprzecinkowych (float).

```
/* Program p2.c */
#include <stdio.h>
main()
{
  int a=6,b=12,c=2;
  float srednia;
  srednia = (a+b+c)/3.0;

  /* Wypisanie trzech zmiennych typu int */
  printf("Liczba a = %d \nLiczba b = %d \nLiczba c = %d \n",a,b,c);

  /* Wypisanie zmiennej typu float */
  printf("Srednia = %f \n",srednia);

  /* Wypisanie zmiennej typu float z dokladnoscia
     do trzech liczb po przecinku  */
  printf("Srednia = %.3f \n",srednia);

  /*Zatrzymanie programu do nacisniecia klawisza ENTER */
  getchar();
}
```

Program wypisze na ekranie

```
Liczba a = 6
Liczba b = 12
Liczba c = 2
Srednia = 6.666667
Srednia = 6.667
```

Najważniejsze znaki przekształceń przedstawiono w poniższej tabeli.

Znak	Opis
d	Argument typu `int` zostanie wypisany jako liczba dziesiętna.
o	Argument typu `int` zostanie wypisany jako liczba ósemkowa bez znaku.
x	Argument typu `int` zostanie wypisany jako liczba szesnastkowa.
u	Argument typu `int` zostanie wypisany jako liczba dziesiętna bez znaku.
c	Argument typu `char` zostanie wypisany jako znak.
s	Wypisany zostanie napis wskazany przez argument typu `char*` (wskaźnik do napisu).
e	Argument typu `float` lub `double` będzie wypisany w postaci [-]m.nnnnnnE[±]xx
f	jak wyżej lecz w postaci [-]mmm.nnnnnn

Funkcja scanf()

Funkcja scanf() jest funkcją wejścia. Umożliwia wprowadzanie danych ze standardowego wejścia, którym może być np. klawiatura. Pierwszy argument scanf() to napis określający format wprowadzanych danych, podobnie jak w funkcji printf(). Kolejne argumenty to adresy zmiennych, do których funkcja ma zapisywać dane, dlatego też nazwy zmiennych należy poprzedzić operatorem adresu „&" (ang. ampersand). Nie należy używać operatora adresu w przypadku zmiennych wskaźnikowych, wskazujących zmienne docelowe. Użycie funkcji scanf() ilustruje poniższy kod źródłowy.

```
/* Program p3.c */
#include <stdio.h>
main()
{
float liczba1, liczba2;

printf("Podaj 1 liczbe: ");
/* Wprowadzanie jednej wartosci */
scanf("%f",&liczba1);

printf("Podaj 2 liczbe: ");
/* Wprowadzanie jednej wartosci */
scanf("%f",&liczba2);

printf(" %f + %f = %f\n", liczba1, liczba2, liczba1+liczba2);


printf("Podaj dwie liczby: ");
/* Wprowadzanie dwoch wartosci oddzielonych znakiem odstepu */
scanf("%f %f",&liczba1,&liczba2);

printf(" %f * %f = %f\n", liczba1, liczba2, liczba1*liczba2);
}
```

Program dla przykładowych danych wypisze na ekranie

```
Podaj 1 liczbe: 11
Podaj 2 liczbe: 12.25
 11.000000 + 12.250000 = 23.250000
Podaj dwie liczby: 10 12.5
 10.000000 * 12.500000 = 125.000000
```

Podstawowe znaki przekształceń funkcji scanf()

Znak	Opis
d i	liczba całkowita dziesiętna (argument `int*`).
o	liczba całkowita w postaci ósemkowej (argument `int*`).
x	liczba całkowita w postaci szesnastkowej (argument `int*`)
c	znak (argument `char*`)
s	tekst (argument `char*` -tablica znaków)
f	liczba zmiennopozycyjna (argument `float*`)

Funkcje

Przy pomocy funkcji można podzielić złożone zadania na mniejsze fragmenty. Raz napisana funkcja może być wywoływana wielokrotnie przez podanie jej nazwy wraz z argumentami wejściowymi. Pozwala to na ukrycie operacji realizowanych wewnątrz funkcji, zwiększając w ten czytelność kodu źródłowego. Formalnie deklaracja funkcji wygląda następująco

typ_danych nazwa_funkcji (***lista parametrów formalnych***)

Typ danych określa, jaki typ wartości zwraca funkcja, natomiast lista parametrów formalnych zawiera deklaracje zmiennych wejściowych oddzielonych przecinkami. Jeżeli funkcja nie zwraca żadnych wartości, typ danych określa się jako `void`. Podobnie należy uczynić przy braku parametrów wejściowych, wpisując `void` w miejsce listy parametrów. Po deklaracji funkcji umieszcza się blok instrukcji wykonywanych przez funkcję.

Aby nazwa funkcji widoczna była wewnątrz całego programu należy umieścić przed wszystkimi funkcjami umieścić jej prototyp, składający się z deklaracji funkcji zakończonej średnikiem. W prototypie nazwy parametrów formalnych można pominąć, pozostawiając jedynie ich typy, Ilustruje to poniższy przykład.

```
#include <stdio.h>

/* Prototyp funkcji */
double ObwodKola(float promien);

int main(void)
{
float r;
double obwod;
printf(" Podaj promien kola: ");
scanf("%f",&r);
obwod = ObwodKola(r);
printf("\n Obwod kola o promieniu %.3f wynosi %.3f\n\n ",r, obwod);
system("PAUSE");
return 0;
}

/* Definicja funkcji */
double ObwodKola(float promien)
{
return 2* 3.14159265 *promien;
}
```

Tablice

Tablica jest strukturą danych składającą się z elementów jednego typu. Każdy element można wskazać podając nazwę tablicy oraz indeksy określające pozycję w tablicy. Tablice mogą być jednowymiarowe lub wielowymiarowe. Formalna deklaracja tablicy wygląda następująco

```
typ_danych  nazwa_tablicy [ lista rozmiarów ] = { lista wartości };
```

Lista wartości służy do wpisania wartości początkowych elementów i jest opcjonalna. Deklaracja jednowymiarowych tablic wygląda następująco

```
int liczby[20];
char nota[50];
int x[4] = { 4,8,1,12 };
```

Zadeklarowana powyżej tablica x składa się z czterech elementów indeksowanych od 0 do 3.

x[0]	x[1]	x[2]	x[3]
4	8	1	12

Deklaracja tablicy dwuwymiarowej i trzywymiarowej wygląda następująco

```
float dane[2][4];
int y[3][3][3];
```

Zmianę zawartości elementów tablicy można realizować przy pomocy instrukcji przypisania np.

```
x[0]=4;
x[1]=8;
x[2]=x[1]-7;
x[3]=x[0]+x[1];
```

W poniższym programie pokazano obsługę tablic z wykorzystaniem pętli `for`. W programie zastosowano instrukcję inkrementacji z operatorem ++.
Instrukcja

```
licznik++;
```

lub

```
++licznik;
```

jest równoważna instrukcji

```
licznik = licznik +1;
```

Analogicznie do dekrementacji zmiennej można użyć operatora `--` np.

```
licznik--;
```

Program wypełnia tablice wartościami, a następnie wyświetla ich zawartość na ekranie.

```
#include <stdio.h>
int main(void)
{
/* Deklaracje tablic */
```

```
 int liczby1[5]={1,2,3,4,5};
 int liczby2[2][3]={ {11,12,13},
                     {21,22,23} };
 int liczby3[2][3][4]={0};

/* deklaracje zmiennych do indeksowania tablic */
int i,j,k,licznik=0;
/* Wypelnianie tablicy trzywymiarowej
   kolejnymi wartosciami              */
 for(i=0;i<2;i++)
   for(j=0;j<3;j++)
     for(k=0;k<4;k++)
       liczby3[i][j][k]=++licznik;

/* Wydruk zawartosci tablic na ekranie*/

printf("Tablica liczby1\n");
 for(i=0;i<5;i++)
    printf("Element Nr %d = %d\n",i,liczby1[i]);

 printf("Tablica liczby2\n");
 for(i=0;i<2;i++)
   for(j=0;j<3;j++)
    printf("Element Nr %d %d = %d\n",i,j,liczby2[i][j]);

 printf("Tablica liczby3\n");
 for(i=0;i<2;i++)
   for(j=0;j<3;j++)
     for(k=0;k<4;k++)
       printf("Element Nr %d %d %d = %d\n",i,j,k,liczby3[i][j][k]);

  return 0;
}
```

Struktury

Struktura jako element języka C jest obiektem złożonym z jednej lub kilku zmiennych, dla wygody zgrupowanych pod jedną nazwą. Formalna definicja struktury wygląda następująco:

```
struct nazwa struktury
   {
    zbiór elementów (składowych) struktury

   } lista tworzonych zmiennych ;
```

W poniższym programie utworzono strukturę `punkt` reprezentującą punkt na płaszczyźnie. Przy deklaracji struktury utworzono dwie zmienne strukturalne `p1` i `p2`. Przy deklaracji zmiennych poza definicją struktury należy deklarację poprzedzić słowem `struct`.

Zmiana zawartości zmiennych strukturalnych odbywa się przez wskazanie nazwy zmiennej i nazwy składowej oddzielonej kropką, jak w poniższym programie.

```
#include <stdio.h>

int main(void)
{
  struct punkt
     {
```

```
      int x;
      int y;
      }p1,p2; /*deklaracja zmiennych p1 i p2 */

    /*deklaracja zmiennych p3 i p4 */
    struct punkt p3;
    struct punkt p4={10,20};

  /* Operacje na zmiennych strukturalnych */
    p1=p4; /* przekopiowanie całej zawartości zmiennej p4 do p1 */
    p2.x=50; /* przypisanie wartości 50 składowej x zmiennej p2 */
    p2.y=100; /* przypisanie wartości 100 składowej y zmiennej p2 */
    p3.x=p3.y=0;

  /* Wydruk zawartości zmiennych strukturalnych */
    printf("p1(%d,%d)\n",p1.x,p1.y);
    printf("p2(%d,%d)\n",p2.x,p2.y);
    printf("p3(%d,%d)\n",p3.x,p3.y);
    printf("p4(%d,%d)\n",p4.x,p4.y);
    return 0;
  }
```

Wskaźniki

Wskaźnik w podstawowym znaczeniu jest zmienną, której wartością jest adres innej zmiennej. Do operacji na wskaźnikach wykorzystuje się dwa operatory:

& -operator adresu,
* -operator adresowania pośredniego, określany także operatorem wyłuskania lub dereferencji.

Formalna postać deklaracji zmiennej wskaźnikowej wygląda następująco:

***typ danych** * **nazwa zmiennej wskaźnikowej** ;*

Na przykład deklaracja

```
double * px;
```

określa, że zmienna `px` jest zmienną wskaźnikową, która może wskazywać na zmienną typu `double`. Aby tak zadeklarowana zmienna wskazywała na konkretną zmienną, należy przypisać jej adres tej zmiennej, wykorzystując operator &.

```
double liczba = 12.5;
double * px;
px = &liczba;
```

Odwołanie się do wartości zmiennej na którą wskazuje zmienna poprzedzając wskaźnik operatorem adresowania pośredniego.

```
*px = 100;
```

Powyższa instrukcja spowoduje przypisanie zmiennej, na którą wskazuje `px` wartości 100. Operacje na wskaźnikach pokazano w poniższym programie.

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

```
{
  double x=3.14;
  double *px=&x;
  double *px2;
  px2=px;
  printf("ZMIENNA       ROZMIAR\n");
  printf("double   x;   %i\n",sizeof(x));
  printf("double *px;   %i\n\n",sizeof(px));
  printf("WYRAZENIE   WARTOSC\n");
  printf("x           %f\n",x);
  printf("*px         %f\n",*px);
  printf("*px2        %f\n",*px2);
  printf("&x          %p\n",&x);
  printf("px          %p\n",px);
  printf("px2         %p\n",px2);
  printf("&px         %p\n",&px);
  printf("&px2        %p\n",&px2);
  return 0;
}
```

Efektem działania programu będzie wypisanie na ekranie następującego tekstu.

```
ZMIENNA       ROZMIAR
double   x;   8
double *px;   4

WYRAZENIE   WARTOSC
x           3.140000
*px         3.140000
*px2        3.140000
&x          0022FF68
px          0022FF68
px2         0022FF68
&px         0022FF64
&px2        0022FF60
```

Graficzną ilustrację powiązań między zmiennymi w programie przedstawiono na poniższym rysunku.