BŁĘDY PODCZAS WPROWADZANIA DANYCH NUMERYCZNYCH LUB WYKONYWANIA OBLICZEŃ

1. Wprowadzanie danych całkowitych

Przykład 1.1.

W programie zadeklarowano zmienną a jako jednobajtową liczbę całkowitą bez znaku (zakres: 0..255).

Dane będą wprowadzane z klawiatury i dla potwierdzenia wyświetlane na ekranie:

Pascal	C	C++
`var a: byte;`	`unsigned char a;`	`unsigned char a;`

`read(a);`
`writeln('a = ',`
`↪a);`
	`scanf("%d", &a);`
	`printf("a = %d\n);`
		`cin >> a;`
		`cout << "a = " << a <<`
		`endl;`

A. Użytkownik, wprowadzając dane, poda liczbę przekraczającą zakres, np. 270.

Liczba 270 w zapisie binarnym ma postać 100001110, czyli do jej zapisania potrzeba 9 bitów.

Pascal, C — zadeklarowana zmienna jest ośmiobitowa, nadmiar bitów (starszych, z lewej strony) zostanie obcięty i w rezultacie, do dalszych działań, otrzymamy liczbę 14 (binarnie: 00001110, bez zer nieznaczących 1110). Zauważmy, że 270–256 = 14. Program nie sygnalizuje błędu tego typu.

C++ — ze strumienia danych zostanie odczytany tylko jeden znak, cyfra 2. W zmiennej a będzie przechowywana liczba 50 – kod znaku 2. Podczas wyświetlania wartości, na konsoli ponownie pojawi się cyfra 2. W taki sposób działają operatory pobierania danych ze strumienia wejściowego (obiekt cin i operator >>) i wstawiania danych do strumienia wyjściowego (obiekt cout i operator <<) dla zmiennych typu unsigned char. Ten typ służy przede wszystkim do przechowywania znaków (a właściwie kodów znaków). Należy dodać, że pozostałe dwa znaki pozostaną w strumieniu wejściowym, oczekując na następne operacje czytające dane. Może to zakłócić pracę dalszej części programu.

B. Użytkownik poda liczbę ujemną, np. –45.

Free Pascal — program przerwie działanie, na konsoli nie pojawi się żaden komunikat o błędzie.

Turbo Pascal, C — reprezentacja binarna liczby –45 (w kodzie U2 11010011) zinterpretowana jako liczba bez znaku ma wartość dziesiętną 211. Taką wartość przyjmie zmienna a.

C++ — ze strumienia danych zostanie odczytany jeden znak – (minus)... – dalszy komentarz wg punktu A.

C. Użytkownik poda liczbę zmiennoprzecinkową, np. 43.5 (z kropką, zgodnie ze sposobem zapisu liczb zmiennoprzecinkowych w omawianych językach programowania) lub 43,5 (zgodnie z przyzwyczajeniami i obowiązującym w Polsce sposobem zapisu liczb dziesiętnych).

Free Pascal — program przerwie działanie, na konsoli nie pojawi się żaden komunikat o błędzie.

Turbo Pascal — program zostanie przerwany z błędem: `Error 106: Invalid numeric format`.

C — zmienna a przyjmie wartość 43 (część całkowita liczby, mieszcząca się w zakresie typu) lub inną wartość zgodną z omówionymi wyżej przypadkami. W strumieniu pozostanie znak (kropka lub przecinek) i reszta cyfr. Może to zakłócić pracę dalszej części programu.

C++ — odczytany zostanie pierwszy znak, czyli '4' – dalszy komentarz wg punktu A.

D. Użytkownik wprowadzi tekst zamiast liczby, np. wpisze słowo dwanaście.

Free Pascal — program przerwie działanie, na konsoli nie pojawi się żaden komunikat o błędzie.

Turbo Pascal — program zostanie przerwany z błędem: `Error 106: Invalid numeric format`.

C — zmienna a zachowa swoją poprzednią wartość (0 jeśli po uruchomieniu programu zmienna nie była używana).

C++ — odczytany zostanie pierwszy znak, czyli 'd' – dalszy komentarz wg punktu A.

E. Użytkownik wprowadzi kilka liczb całkowitych, należących do zadeklarowanego typu, oddzielonych odstępami lub znakami tabulacji.

Pascal, C — program poprawnie odczyta pierwszą liczbę, pozostałe liczby pozostaną w strumieniu danych i mogą być odczytane w dalszej części programu.

C++ — odczytany zostanie pierwszy znak z pierwszej liczby...

Uwaga. Pomimo, że typ `unsigned char` w C i C++ zawiera jednobajtowe liczby bez znaku, to jest to typ przeznaczony głównie do przechowywania znaków. Dlatego też taka jest interpretacja wczytywanych (zapisywanych) danych przy użyciu obiektu klasy iostream i operatora >> (<<).

W C taki efekt uzyskamy zastępując specyfikator `%d` lub `%u`) specyfikatorem `%c`, a w Pascalu do operacji na znakach używamy zmiennych typu char (nie będących liczbami).

Łatwiej będzie analizować omawiane przypadki jeśli zastosujemy pętlę, która umożliwi wielokrotne odczytywanie danych ze strumienia.

Pascal	C	C++

`repeat`
`  read(a);`
`  writeln('a = ',`
`↪a);`
`until a = 0;`
	`do {`
	`  scanf("%d",`
	`&a);`
	`  printf("a =`
	`%d\n",a);`
	`} while (a != 0);`
		`do {`
		`  cin >> a;`
		`  cout << "a = " <<`
		`a << endl;`
		`} while (a != '0');`

Ten kod pozwoli również zaobserwować wpływ pozostających w strumieniu znaków na dalszą pracę programu (odczytywanie danych przez kolejne instrukcje wejścia). Ponieważ pozostające w strumieniu znaki mogą mieć niekorzystny wpływ na dalszą pracę programu, to pożądane może być czyszczenie strumienia:

Pascal — dodajemy instrukcję `readln` bez parametrów, która usunie ze strumienia resztę znaków do końca linii lub stosujemy procedurę `readln(a)` zamiast procedury `read(a)`.

C — wywołujemy funkcję `fflush(stdin)`, która usunie resztę znaków ze standardowego strumienia.

C++ — wywołujemy funkcję `fflush(stdin)` i metodę `cin.clear()`, ustawiającą (zerującą) flagi strumienia wejściowego. Możliwe jest też użycie metody `cin.ignore(n)`, usuwającej `n` znaków ze strumienia. Jeśli podana liczba znaków jest zbyt mała, to problem nie zostanie rozwiązany. Również druga postać wywołania tej metody `cin.ignore(n, '\n')` usuwająca ze strumienia wszystkie znaki aż do wystąpienia znaku `'\n'` (koniec linii), ale nie więcej niż `n` znaków, nie zawsze okaże się skuteczna (np. jeśli użytkownik jednorazowo wpisze więcej niż n znaków).

Przykład 1.2.

W programie zadeklarowano zmienną a, jako jednobajtową liczbę całkowitą ze znakiem (zakres: –128..127).

Pascal	C	C++

`var a: byte;`	`unsigned char a;`
	lub
	`char a;`
		`unsigned char a;`
		lub
		`char a;`

Program testujący na podstawie kodu z przykładu 1.1.		

C++ — bez zmian, ze strumienia `cin` będą odczytywane pojedyncze znaki, wartość zmiennej `a` będzie równa kodowi znaku (postać binarna identyczna jak w poprzednim przypadku, wzrtość dziesiętna zgodna z interpretacją kodu U2).

Turbo Pascal — wprowadzana z klawiatury liczba całkowita nie może przekroczyć zakresu dla 32 bitowych liczb całkowitych bez znaku (typ longint: –2147483648.. 2147483647) — po przekroczeniu tego zakresu otrzymamy komunikat: `Error 106: Invalid numeric format` (w IDE).

C — nie ma ograniczeń na długość wprowadzanego ciągu cyfr. Z obserwacji działania programu wynika, że jeśli liczba jest zbyt długa, to pewne cyfry są pomijane (może Czytelnik pokusi się o wnikliwszą analizę i znajdzie odpowiednią zależność). Interpretacja liczby, po odrzuceniu zbędnych cyfr przebiega w sposób niżej opisany.

A. Użytkownik wprowadzi za dużą liczbę dodatnią, np. 264 lub 400.

Pascal, C — podane liczby mają (odpowiednio) reprezentację binarną *110010000* lub *100001000* (bez zer nieznaczących). Po obcięciu słowa do 8 bitów pozostanie *10010000* lub *00001000*, co w kodzie U2 odpowiada liczbie *–112* lub *8*.

B. Użytkownik poda za małą liczbę ujemną, np. –250 lub –270.

Wprowadzona wartość zostanie zinterpretowana jako 32 bitowa liczba bez znaku, a następnie "obcięta" do 8 bitów (rozmiar typu shortint). Zatem dla *–250* otrzymamy binarnie *11111111 11111111111111100000110*, po obcięciu *00000110*, czyli wartość zmiennej będzie równa *6*. Podobnie dla –270 dostaniemy *11111111111111111111111011110010*, po obcięciu *11110010*, czyli *–14* (kod U2).

Free Pascal — jw. – jedyną różnicą jest przerwanie działania programu w przypadku błędu, bez jakiegokolwiek komunikatu.

C — jak wspomniano wcześniej, nie ma ograniczeń na długość wprowadzanego ciągu cyfr. Interpreatacja liczby, po odrzuceniu zbędnych cyfr przebiega w sposób podobny jak w Pascalu.

Uwaga. W przypadku popełnienia innych błędów, reakcja program jest podobna do opisanej w przykładzie 1.1.

Przykład 1.3.

Deklarujemy zmienną a **jako zmienną całkowitą 16, 32 lub 64 bitową, ze znakiem lub bez znaku.**

W programie testującym zmienimy typ zmiennej a na odpowiedni typ i możemy prowadzić testowanie wprowadzanych wartości. Efekty będą dokładnie takie, jakie omówiono w przykładach 1.1. i 1.2. Zmienią się jedynie zakresy wartości, stosownie do wybranego typu danych. Przykłady realizowane w języku C++ (w wersji z obiektami cin i cout oraz operatorami << i >>) będą działały w sposób analogiczny jak w języku C (lub C++) z funkcjami scanf i printf. Obserwacje w tym zakresie Czytelnik może przeprowadzić samodzielnie.

Rozmiar	Pascal	C lub C++	Zakres wartości
Liczby całkowite bez znaku			
1B	byte	unsigned char	0..255 (0..2^8–1)
2B	word	unsigned short	0..65535 (0..2^{16}–1)

4B	longword*	unsigned int	0.. 4294967295 (0..2^{32}–1)
8B	----------	unsigned long long int	0.. 18446744073709551615 (0..2^{64}-1)
Liczby całkowite ze znakiem			
1B	shortint	signed char, char	-128..127 (–2^7..2^7)
2B	integer	signed short, short	–32768..32767 (–2^{15}..2^{15}–1)
4B	longint	signed int, int	–2147483648.. 2147483647 (–2^{31}..2^{31}–1)
8B	int64*	signed long long int, long long int	–9223372036854775808.. 9223372036854775807 (–2^{63}..2^{63}–1)
* tylko Free Pascal			

2. Obliczenia wykonywane na liczbach całkowitych

Zakładamy, że użytkownik wprowadzi dane zgodnie z zakresem zadeklarowanej zmiennej. Jednak podczas wykonywania obliczeń wynik przekroczy dopuszczalny zakres.

Możliwe są trzy sytuacje:

- wynik zostanie przypisany do zmiennej tego samego typu – otrzymana wartość nie będzie zgodna z prawdą (nadmiar bitów zostanie obcięty)

- wynik zostanie skierowany na konsolę, o ile nie zostanie przekroczony zakres maksymalnego dostępnego typu danych, to wyświetlona wartość będzie poprawna

- wynik przekroczy zakres maksymalnego dostępnego typu całkowitego i nadmiar bitów zostanie obcięty (niezależnie od dalszego postępowania z wynikiem).

A. Obliczmy pole powierzchni kwadratu o boku, którego długość jest liczbą całkowitą.

Test przeprowadzimy stosując program napisany w Free Pascalu (ze względu na dostęp do wielu typów danych całkowitych).

```
var a, pole: byte;
begin
  repeat
    readln(a);
    pole := a*a;
```

```
    writeln('P = ', pole);
    writeln('P = ', a*a);
  until a = 0;
  writeln('Koniec obliczeń...');
  readln;
end.
```

Zestawienie użytych typów danych, badane wartości danych wejściowych i uzyskane wyniki zestawiono w tabeli:

Typ zmiennych	Wartość zmiennej a	Wartość zmiennej pole		Wartość obliczona (a*a)	
shortint	9	81	poprawna	81	poprawna
	12	–112	błędna	144	poprawna
byte	14	196	poprawna	196	poprawna
	17	33	błędna	289	poprawna
integer	130	16900	poprawna	16900	poprawna
	190	–29463	błędna	36100	poprawna
word	254	64516	poprawna	64516	poprawna
	256	0	błędna	65536	poprawna
	300	24464	błędna	90000	poprawna
	65535	1	błędna	4294836225	poprawna
longint	46340	2147395600	poprawna	2147395600	poprawna
	46341	–2147479015	błędna	–2147479015	błędna
longword	46341	2147488281	poprawna	2147488281	poprawna
	65535	4294836225	poprawna	4294836225	poprawna
	65536	0	błędna	0	błędna
	70000	605032704	błędna	605032704	błędna
int64	65536	4294967296	poprawna	4294967296	poprawna
	70000	4900000000	poprawna	4900000000	poprawna
	10000000	100000000000000	poprawna	100000000000000	poprawna
	4294967296	0	błędna	0	błędna
	5000000000	6553255926290448384	błędna	6553255926290448384	błędna

Niektóre błędy łatwo dostrzec, np. znając tabliczkę mnożenia i zasady budowy systemu dziesiątkowego zauważymy, że 5000000000·5000000000 = 25000000000000000000, a nie jak obliczył komputer 6553255926290448384.

Podobne testy można wykonać w Turbo Pascalu w zakresie dostępnych typów danych oraz w C i C++. Wyniki będą podobne.

Przekroczenie zakresu wartości dostępnych dla danego typu liczb całkowitych może wystąpić również podczas dodawania lub odejmowania. Uzyskiwane efekty będą podobne.

B. Dzielenie całkowite i dzielenie zmiennoprzecinkowe.

W Pascalu mamy dwa operatory dzielenia. Dzielenie całkowite, operator `div` i operator `/` dający wynik w postaci liczby zmiennoprzecinkowej. Dla zmiennej a typu całkowitego możemy wykonać podstawienie `a := a div 3`, natomiast podstawienie `a := a/3` zostanie zasygnalizowane jako błędne już na etapie kompilacji (nie można zmiennej całkowitej przyporządkować wartości zmiennoprzecinkowej).

W C i C++ wynik dzielenia `a/b` jest typu całkowitego, gdy obie zmienne `a` i `b` są typu całkowitego (dzielenie całkowite). Natomiast jeśli conajmniej jedna ze zmiennych `a` i `b` jest typu zmiennoprzecinkowego, to iloraz jest też typu zmiennoprzecinkowego. Podobnie jest, gdy jedna lub obie wartości są stałymi.

W związku z powyższymi uwagami, wykonując dzielenie *1/3* otrzymamy wartość *0,3333333333* w Pascalu i *0* w C lub C++. Chcąc uzyskać przybliżenie dziesiętne ułamka *1/3* w C lub C++, musimy wyrażenie zapisać w postaci *1.0/3*, *1/0.3* lub *1.0/3.0*.

Wzór na obliczanie pola powierzchni trójkąta można zatem zapisać w następujący sposób:

Pascal	C, C++
`pole := 1/2*a*h;` lub `pole := 0.5*a*h;` lub `pole := a*h/2;`	`pole = 1.0/2*a*h;` lub `pole = 0.5*a*h;` lub `pole = a*h/2;`

3. Wprowadzanie danych zmiennoprzecinkowych

Dane możemy wprowadzać w postaci liczby całkowitej, liczby dziesiętnej (z kropką oddzielającą część całkowitą od części ułamkowej) i w postaci naukowej. W zależności od typu wczytywanej zmiennej, istotna może być liczba cyfr znaczących w wprowadzanej liczbie. O przekroczenie zakresu, podczas wprowadzania danych, zwykle nie musimy się troszczyć. Błędem może być podanie przecinka, zamiast używanej w językach programowania kropki dziesiętnej. Czytelnik może samodzielnie przygotować i przeprowadzić odpowiednie testy.

4. Obliczenia na liczbach zmiennoprzecinkowych

Testy wykonamy w programie obliczającym pole kwadratu dla zmiennych typu `real` w Pascalu `double` w C i C++.

Pascal	C	C++

`var a, pole: real;`
`begin`
`  repeat`
`  readln(a);`
`  pole := a*a;`
`  writeln(pole);`
`  until a = 0;`
`end.`
	`#include<stdio.h>`
	`int main()`
	`{`
	`  double a, pole;`
	`  do {`
	`  scanf("%lf",`
	`&a);`
	`  fflush(stdin);`
	`  pole = a*a;`
	`  printf("%lf\n",`
	`pole);`
	`  } while (a != 0);`
	`  return 0;`
	`}`
		`#include<iostream>`
		`using namespace std;`
		`int main()`
		`{`
		`  double a, pole;`
		`  do {`
		`  cin >> a;`
		`  cin.clear();`
		`  fflush(stdin);`
		`  pole = a*a;`
		`  cout << pole <<`
		`endl;`
		`  } while (a != 0);`
		`  return 0;`
		`}`

Zwróćmy uwagę na używanie procedury `readln` w Pascalu, funkcji `fflush(stdin)` w C oraz funkcji `fflush(stdin)` i metody `cin.clear()` — ma to na celu usuwanie ze strumienia wejściowego znaków pozostałych po wczytaniu liczby. Rola tych funkcji była wcześniej omówiona.

A. Turbo Pascal — typ real, liczby (dodatnie) w zakresie od 2.8·10–39 do 1.7·1038, precyzja 11-12 cyfr.

Wynik wyświetlany jest w notacji naukowej, z precyzją ustaloną przez twórców kompilatora.

Bok kwadratu a	Pole powierzchni (a*a)	Uwagi
5.6	3.1360000000E+01	Dane i wynik mieszczą się w zakresie typu zmiennej.
0.0037	1.3690000000E–05	
1e–6 (lub 0.000001)	1.0000000000E–12	
2.4e-22	0.0000000000E+00	
3.937e15	1.5499969000E+33	
1.3e19	1.6900000000E+38	Wynik zbliża się do górnej granicy zakresu typu real.
1.5e20	błąd	`Error 205: Floating point overflow.`(IDE) `Runtime error 205 at ...` (konsola) Wynik przekroczył dopuszczalny zakres.
1.8e38	błąd	`Error 106: Ivalid numeric format.` (IDE) `Runtime error 106 at ...` (konsola) Dane przekroczyły dopuszczalny zakres.

B. Free Pascal — typ real (lub double), liczby (dodatnie) w zakresie od 5.0·10–324 do 1.7·10308, precyzja 15-16 cyfr.

Test można przeprowadzić wg schematu zastosowanego wyżej. Przekroczenie zakresu przez wprowadzane dane lub wynik obliczeń sygnalizowany jest komunikatem: `Runtime error 205 at...` w konsoli (lub `Program ... exided with exit code = 205` w środowisku IDE). Dla wartości `a` nie większej niż *1.3e154*, obliczone pole nie przekroczy wartości *1.69e308* i mieści się w zakresie typu.

C. C — typ double, liczby (dodatnie) w zakresie od 5.0·10–324 do 1.7·10308, precyzja 15-16 cyfr.

Test przebiega podobnie. Wyniki wyświetlane są w postaci stałoprzecinkowej z dokładnością 6 cyfr po przecinku. Jesli liczba ma więcej niż 16 cyfr w części całkowitej, to pozostałe cyfry są zerami. Program bez problemu przyjmuje wartości spoza zakresu i wykonuje obliczenia. Jeśli dane lub wynik przekroczą zakres typu `double`, to wyświetlany jest wynik w postaci `1.#INFOO` (nadmiar). Możemy wyniki wyświetlać w postaci naukowej, jeśli użyjemy specyfikatora formatu `%e` (lub `%E`) czyli w kodzie wstawimy wiersz: `printf("%e\n", pole);`

D. C++ — typ double, liczby (dodatnie) w zakresie od 5.0·10–324 do 1.7·10308, precyzja 15-16 cyfr.

Test przebiega tak samo jak w przypadku języka C. Jedyną różnicą jest automatyczny sposób dobierania formatu wyniku przez operator << wstawiający dane do strumienia wyjściowego (obiekt `cout`). Jest to zachowanie domyślne, które można zmienić w sposób omówiony w dodatku D.

Uwaga. Przekroczenie zakresu może również wystąpić podczas wykonywania innych działań. Reakcja programu będzie podobna.

5. Dzielenie przez 0

Dzielenie przez 0 jest w matematyce operacją niewykonywalną. Przetestujmy jak zachowa się w tej sytuacji program komputerowy. Zbadamy przypadek dzielenia całkowitego i dzielenia zmiennoprzecinkowego.

Pascal	C	C++

`var a, b, c:`
`↪integer;`
`{...}`
`a := 5;`
`b = 0;`
`c := a div b;`
`writeln(c);`
	`int a, b, c;`
	`a = 5;`
	`b = 0;`
	`c = a/b;`
	`printf("%d\n", c);`
		`int a, b, c;`
		`a = 5;`
		`b = 0;`
		`c = a/b;`
		`cout << c << endl;`

```
var a, b, c: real;
{...}
a := 5;
b := 0;
c := a/b;
writeln(c);
```

```
double a, b, c;
a = 5;
b = 0;
c = a/b;
printf("%lf\n", c);
```

```
double a, b, c;
a = 5;
b = 0;
c = a/b;
cout << c << endl;
```

Turbo Pascal — próba wykonania dzielenia przez *0* w obu przypadkach, zakończy się komunikatem błędu o numerze *200* (`Runtime error 200 at...`). Jeśli `a` jest typu całkowitego i w kodzie programu umieścimy wyrażenie postaci `a div 0`, to program nie skompiluje się (`Error 62: Division by zero`). Podobnie nie można skompilować wyrażenia `a/0` (`Error 26: Type mismatch.`) Natomiast dla zmiennej `a` typu zmiennoprzecinkowego, wyrażenie `a/0` kompiluje się, a błąd zostanie wykryty podczas działania programu. Próba użycia operatora div do dzielenia liczb zmiennoprzecinkowych zakończy się na etapie kompilacji (`Error 41: Operand types do not match operator.`)

Free Pascal — test wypadnie podobnie, inaczej będą sformułowane kody błędów.

C, C++ — próba wykonania dzielenia całkowitego przez *0* zakończy się przerwaniem programu. Dzielenie zmiennoprzecinkowe zostanie wykonane, a wynikiem będzie symbol: `1.#INF` (*infinity*, czyli nieskończoność ∞). Program zawierający wyrażenie typu `a/0` skompiluje się, przyczym kompilator wygeneruje odpowiednie ostrzeżenia. Błędy zostaną zasygnalizowane, w wyżej opisany sposób, podczas wykonywania programu.

6. Błędne argumenty innych funkcji matematycznych

Ograniczymy się do omówienia jednego przykładu, podejmiemy próbę obliczania pierwiastka kwadratowego z liczby ujemnej.

Pascal	C	C++

`begin`
`writeln(sqrt(-1));`
`end;`
	`#include<stdio.h>`
	`#include<math.h>`
	`int main()`
	`{`
	`  printf("%lf\n",`
	`sqrt(-1));`
	`  return 0;`
	`}`
		`#include<iostream>`
		`#include<cmath>`
		`using namespace std;`
		`int main()`
		`{`
		`  cout << sqrt(-1) <<`
		`endl;`
		`  return 0;`
		`}`

Turbo Pascal — program skompiluje się, po uruchomieniu zostanie przerwany z błędem `Runtime error 207 at...` (w IDE: `Error 207: Invalid floating point operation.`)

Free Pascal — obliczenia zostaną wykonane, wynik zostanie wyświetlony w postaci: `Nan` (not a number, czyli wynik nie jest liczbą).

C, C++ — obliczenia zostaną wykonane, wynik przyjmie postać `-1.#IND` (lub inaczej NaN — ta pisownia symbolu jest częście używana).