podstawy i języki programowania

pjp

Podstawy programowania

Część druga

Pierwsze programy

Autor

Roman Simiński

Kontakt

roman.siminski@us.edu.pl

www.programowanie.siminskionline.pl

Niniejsze opracowanie zawiera skrót treści wykładu, lektura tych materiałów nie zastąpi uważnego w nim uczestnictwa.
Opracowanie to jest chronione prawem autorskim. Wykorzystywanie jakiegokolwiek fragmentu w celach innych niż nauka własna jest nielegalne.
Dystrybuowanie tego opracowania lub jakiejkolwiek jego części oraz wykorzystywanie zarobkowe bez zgody autora jest zabronione.

Problem do rozwiązania

Problem

Należy napisać program pozwalający na przeliczenie odległości podanej w kilometrach na mile amerykańskie.

Scenariusz działania programu:

```
Przeliczanie odleglosci wyrazonej w kilometrach na mile
Podaj odleglosc w kilometrach: 100
To w milach: 62.50
Nacisnij Enter by zakonczyc program...
```

Odległość w milach = 0.625 · odległość w kilometrach

Odległość w kilometrach = 1.6 · odległość w milach

Jak się za to zabrać? Załóżmy, że nic nie wiemy o programowaniu...

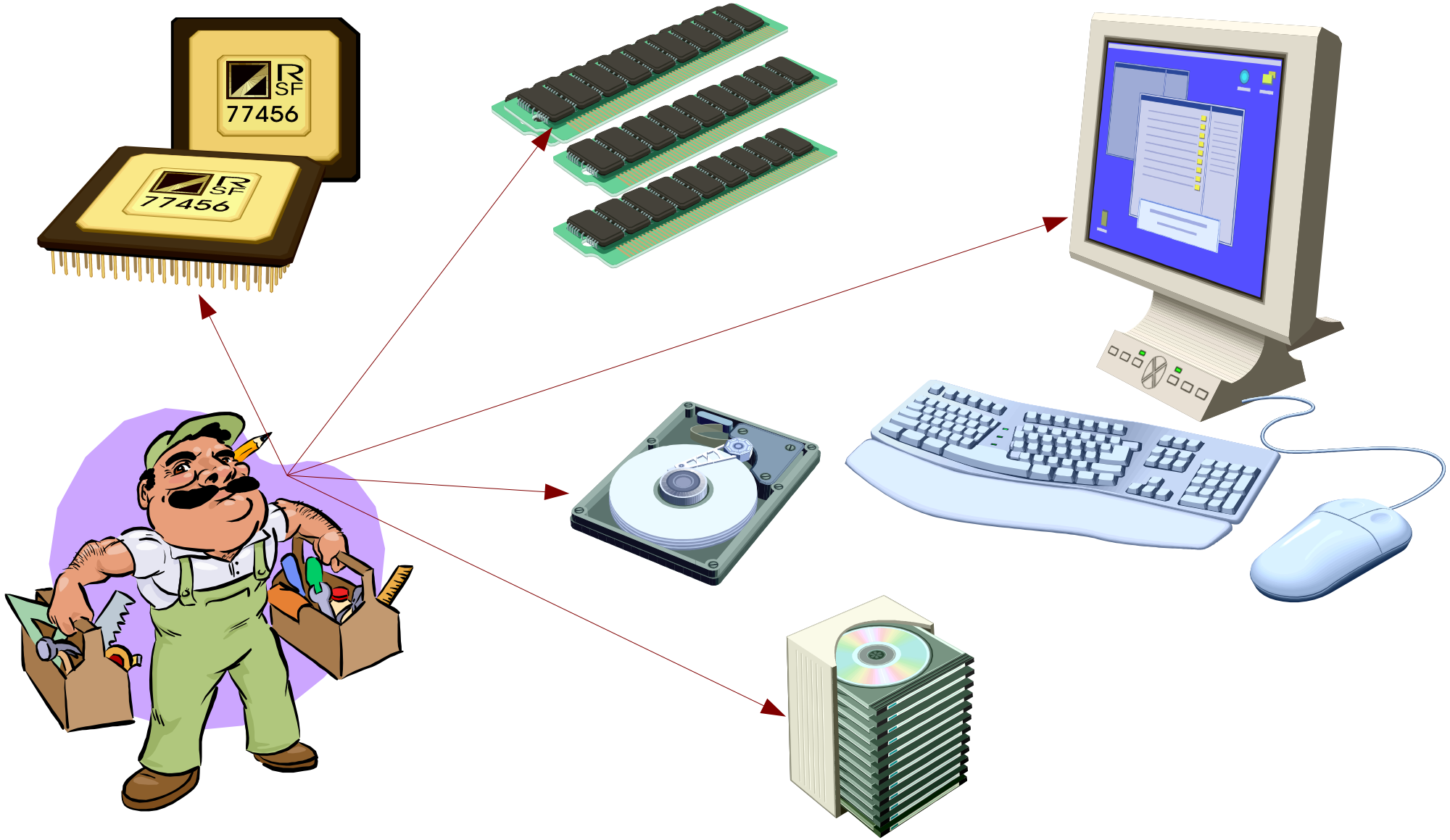
Copyright © Roman Simiński

Spojrzenie użytkownika — komputer to czarna skrzynka

Użytkownika interesuje CO potrafi zrobić komputer a nie JAK to robi… .

Copyright © Roman Simiński

Spojrzenie programisty — komputer od środka



Programista musi wiedzieć jak zmusić komputer, żeby zrobił to co trzeba.

Copyright © Roman Simiński

Algorytm, czyli JAK TO SIĘ ROBI!

Programista musi znać sposób rozwiązania problemu, który stanowi przedmiot programu. Ten sposób określa algorytm.

Algorytm określa *dane* oraz skończony ciąg *operacji*, jakie należy na tych danych wykonać, aby rozwiązać dowolny problem z określonej klasy.

Inaczej mówiąc, **algorytm** określa *dane* oraz *plan akcji*, jakie należy wykonać, aby osiągnąć założony cel.

Copyright © Roman Simiński

Cechy algorytmu

Algorytm ma być:

- *Poprawny* — dla każdego poprawnego zestawu informacji wejściowych prowadzić do poprawnych rezultatów.
- *Jednoznaczny* — każdorazowo, dla każdego poprawnego zestawu informacji wejściowych, prowadzić do tych samych rezultatów.
- *Szczegółowy* — aby *ktoś* albo *coś* wykonujący algorytm rozumiał dokładnie opisane operacje i potrafił je wykonać.
- *Uniwersalny* — pozwalał na rozwiązanie *dowolnego* problem z określonej klasy, a nie dotyczył pewnych przypadków.

Copyright © Roman Simiński

Metody zapisu algorytmów

Wykorzystuje się różne metody zapisu algorytmów:

- Język naturalny (potoczny) określający ciąg kroków algorytmu.
- Notacje graficzne, najpopularniejsze są *schematy blokowe*.
- Pseudokod — połączenie języka naturalnego i notacji matematycznej z elementami języków programowania.
- Zapis w postaci kodu programu w pewnym języku programowania.

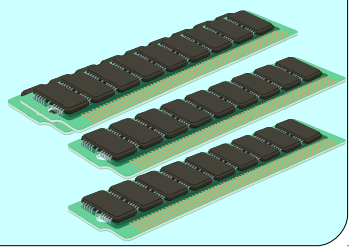
Copyright © Roman Simiński

Co to za „dane”, co to za „operacje”?

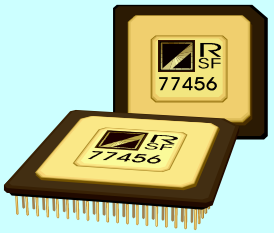
Dane, które program pamięta
w czasie swojego działania

Wykonanie programu
Obliczenia, sterowanie, nawroty,
decyzje itp., itd. ...

Pamięć
podręczna
procesora

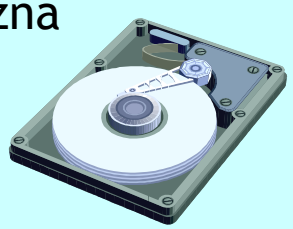


Procesor
i
spółka



Urządzenia
wejścia
i
wyjścia

Pamięć
zewnętrzna



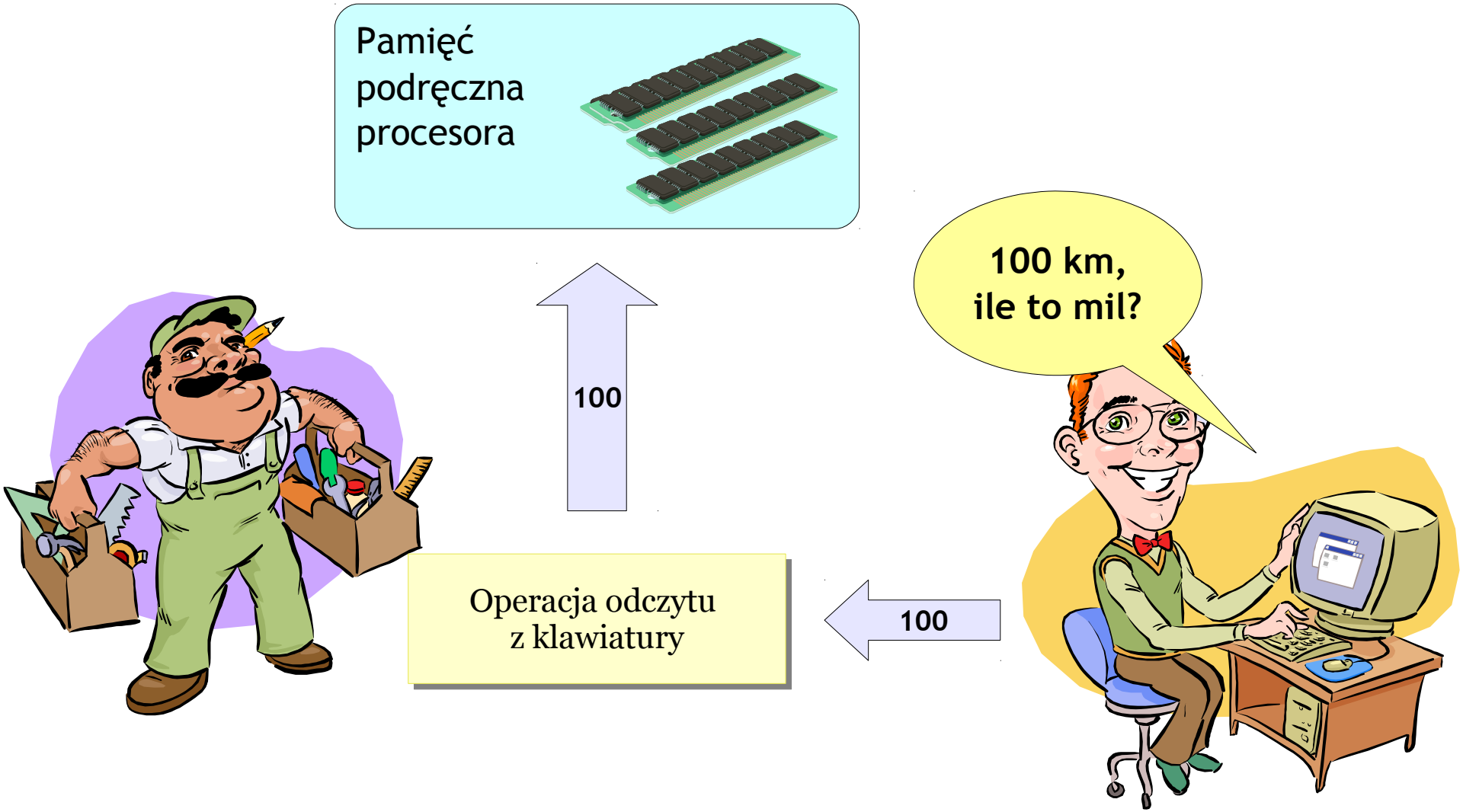
Dane, które mają być zapamiętane trwale

Operacje wejścia-wyjścia

Copyright © Roman Simiński

A co z przeliczaniem z kilometrów na mile?

Czy w tym problemie występują jakieś dane?



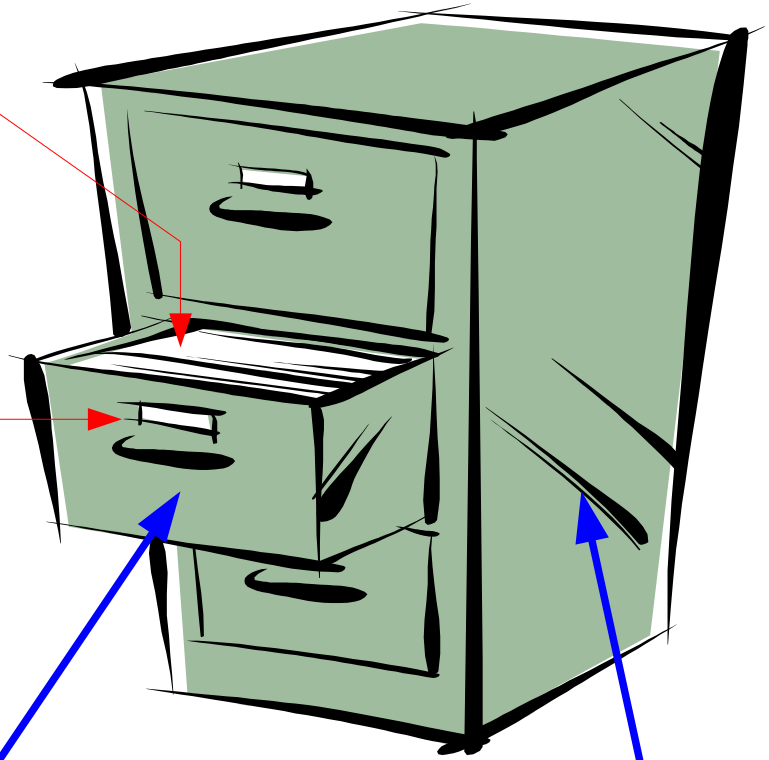
Copyright © Roman Simiński

Jak dane są przechowywane w pamięci operacyjnej?

Zmienna jest to element programu, rezydujący w pamięci operacyjnej, przeznaczony do przechowywania wartości.

Każda zmienna ma swój typ

Każda zmienna ma swoją nazwę



Zmienna

Pamięć operacyjna

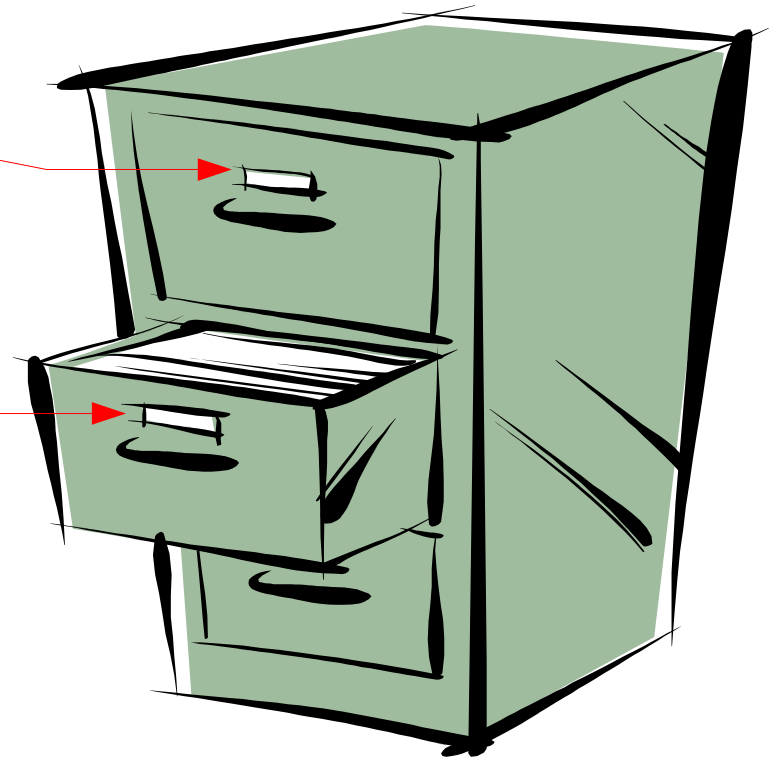
Copyright © Roman Simiński

Rola nazwy zmiennej

Nazwa zmiennej jednoznacznie *identyfikuje* ją w programie.
Posługując się nazwą zmiennej, odwołujemy się do zapisanej w niej wartości.

Każda zmienna ma swoją *nazwę*

Nazwy zmiennych powinny być *unikatowe* w obrębie danego fragmentu programu



Nazwa zmiennej:

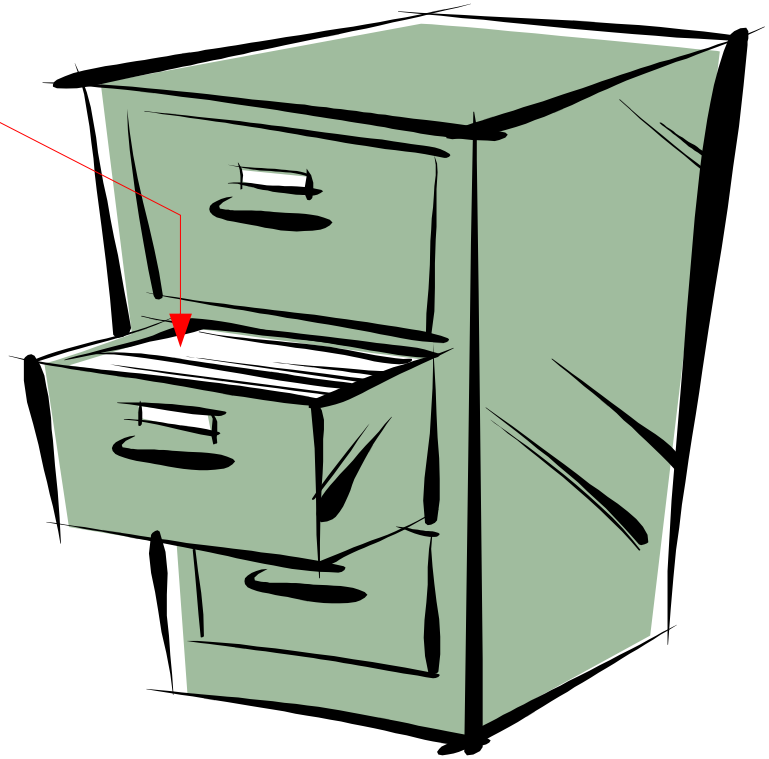
- To *słowo*, *skrót*, *akronim* lub ich *zbitek*, wymyślony przez programistę.
- Zwykle nazwa zmiennej może zawierać tylko *litery*, *cyfry* i *znaki podkreślenia*.

Copyright © Roman Simiński

Rola typu zmiennej

Typ zmiennej określa jakie wartości można do niej wstawiać.

Każda zmienna ma swój typ



Typ zmiennej określa dokładnie:

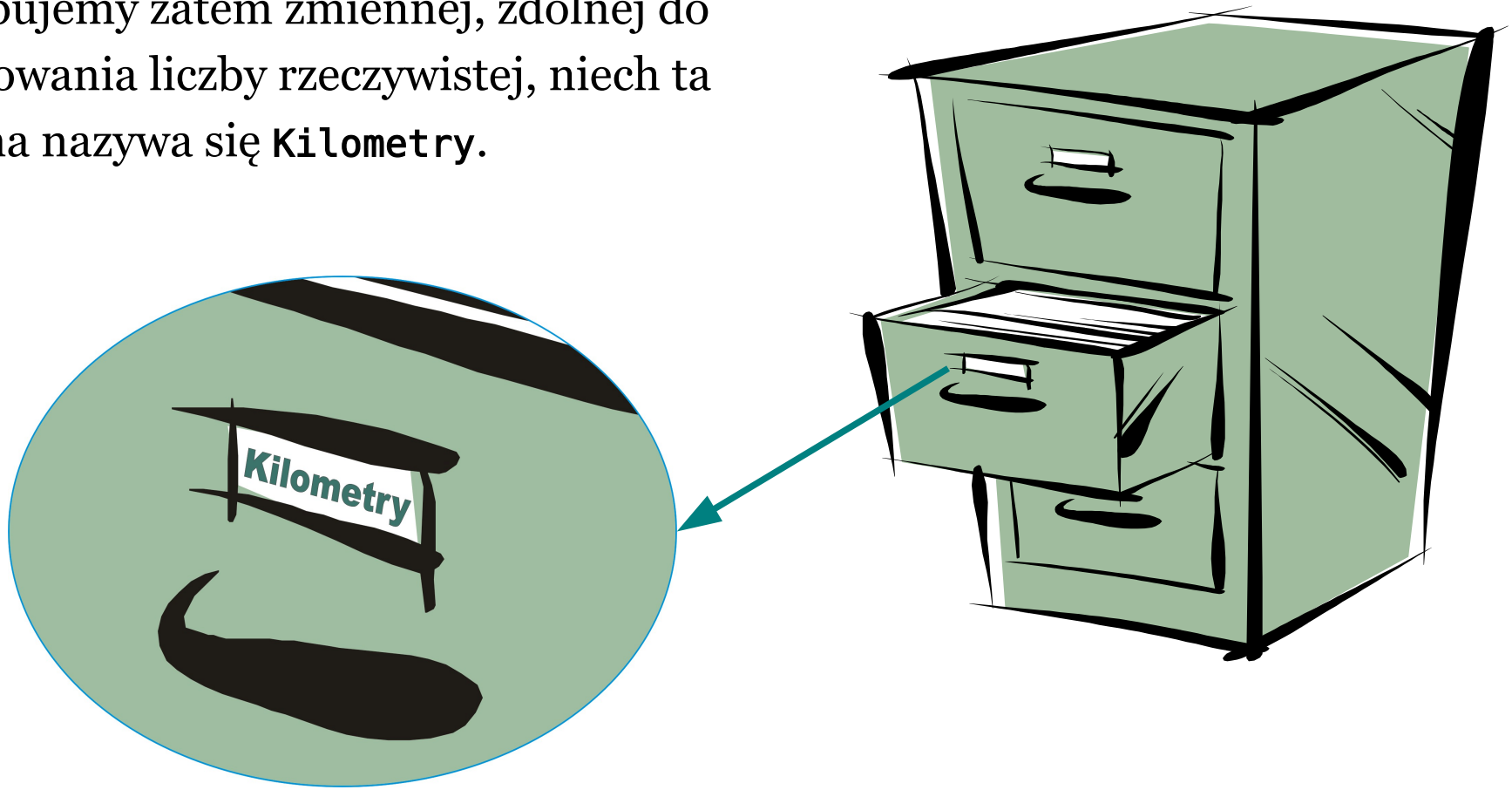
- Jakie *wartości* może przyjmować konkretna zmienna, i z *jakiego zakresu* mogą one pochodzić.
- Jakie *operacje* można wykonywać na zmiennej.
- Ile *bajtów* zmienna zajmuje w pamięci operacyjnej.

Copyright © Roman Simiński

Zmienne a przeliczenie odległości

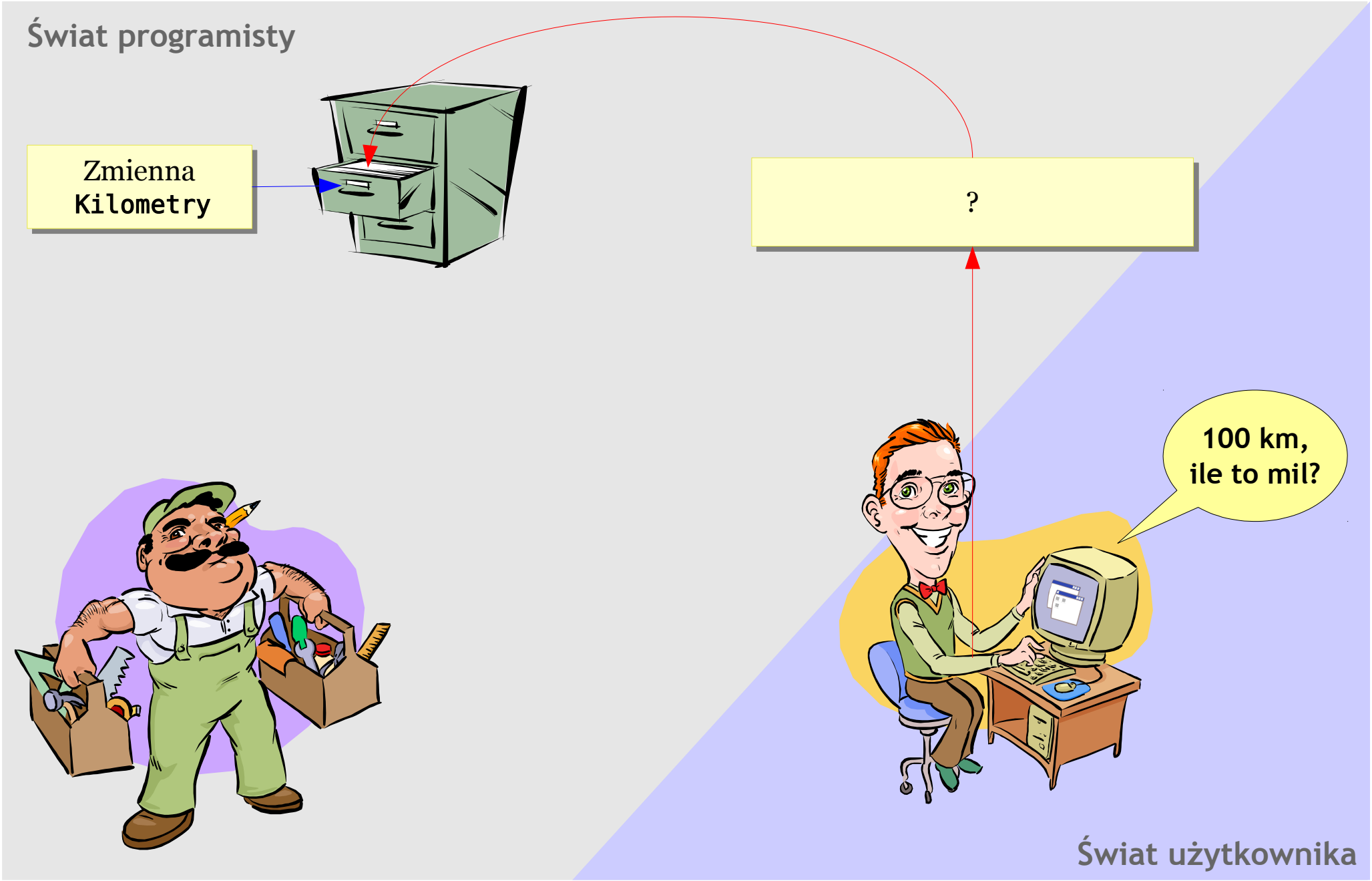
Potrzebujemy miejsca w pamięci operacyjnej, w którym przechowamy na czas działania programu odległość w kilometrach, podaną przez użytkownika.

Potrzebujemy zatem zmiennej, zdolnej do przechowania liczby rzeczywistej, niech ta zmienna nazywa się `Kilometry`.

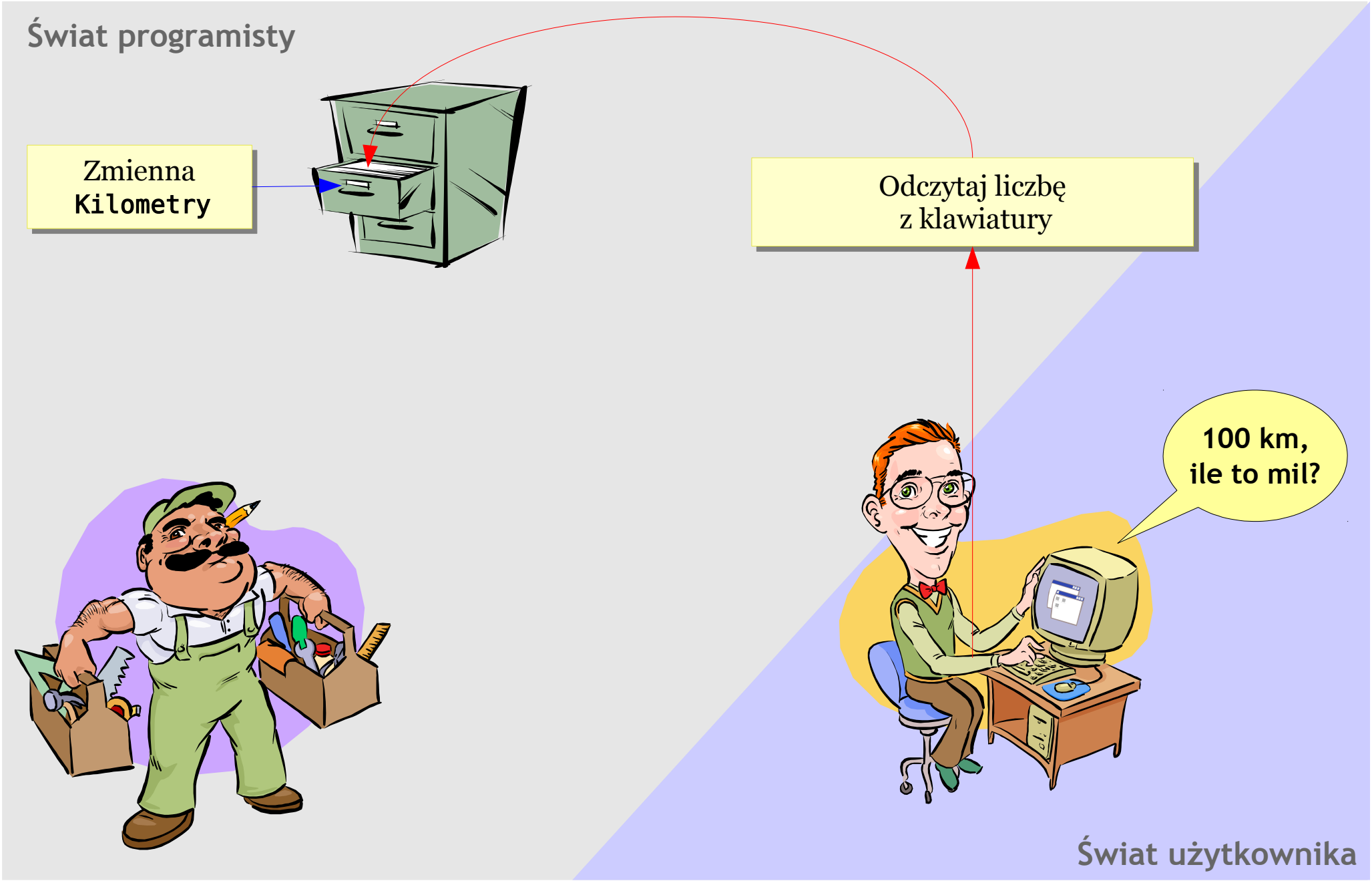


Copyright © Roman Simiński

Zmienne a przeliczenie odległości



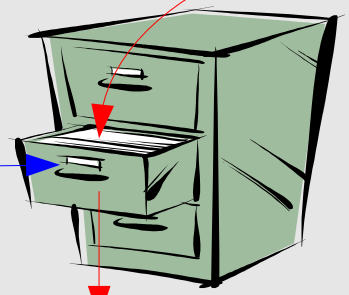
Zmienne a przeliczenie odległości



Zmienne a przeliczenie odległości

Świat programisty

Zmienna
`Kilometry`



Przelicz na mile:

???

Odczytaj liczbę
z klawiatury

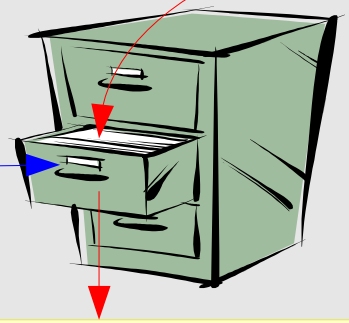
Świat użytkownika

Copyright © Roman Simiński

Zmienne a przeliczenie odległości

Świat programisty

Zmienna
`Kilometry`



Przelicz na mile:

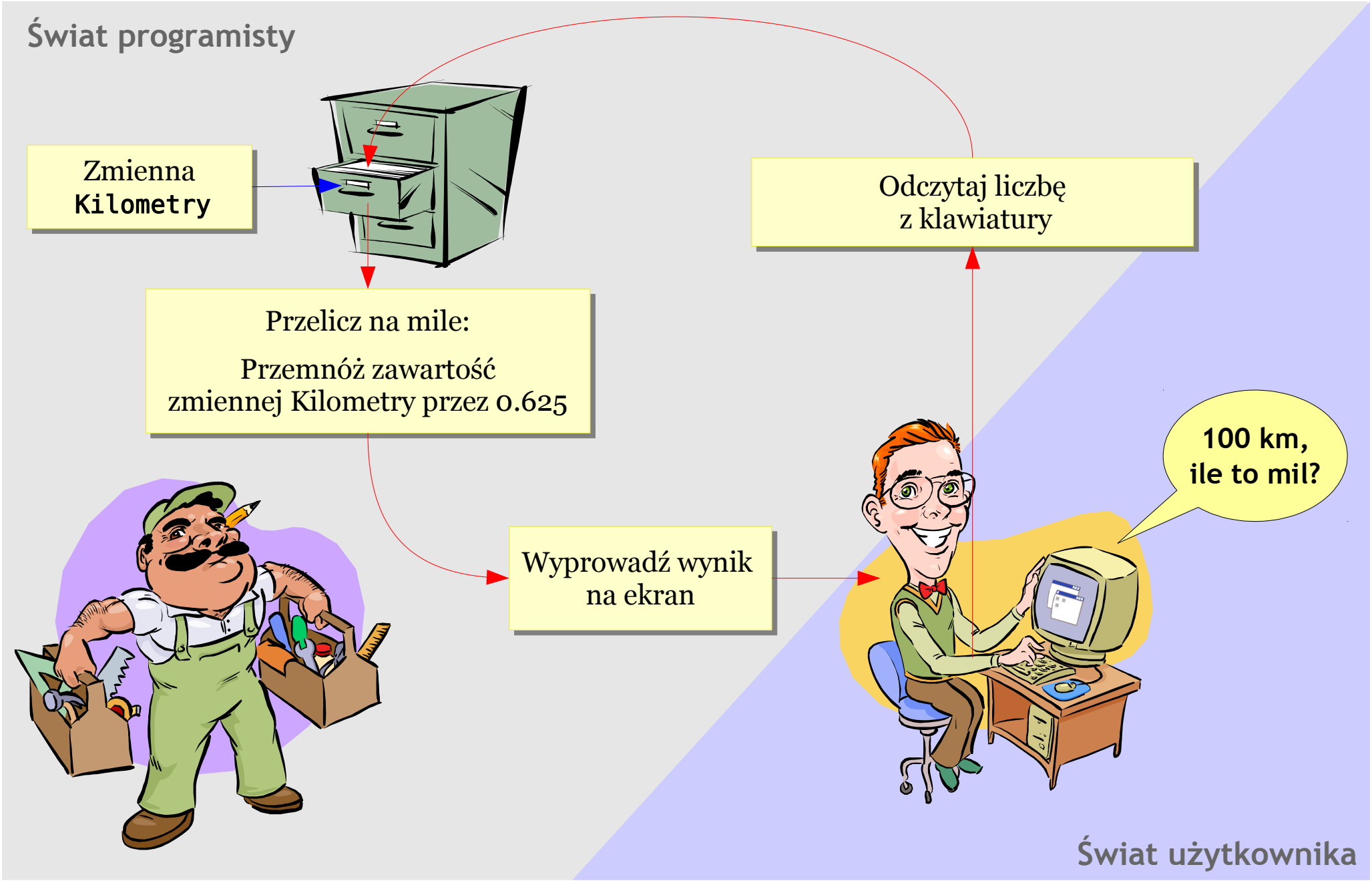
Przemnóż zawartość
zmiennej Kilometry przez 0.625

Odczytaj liczbę
z klawiatury

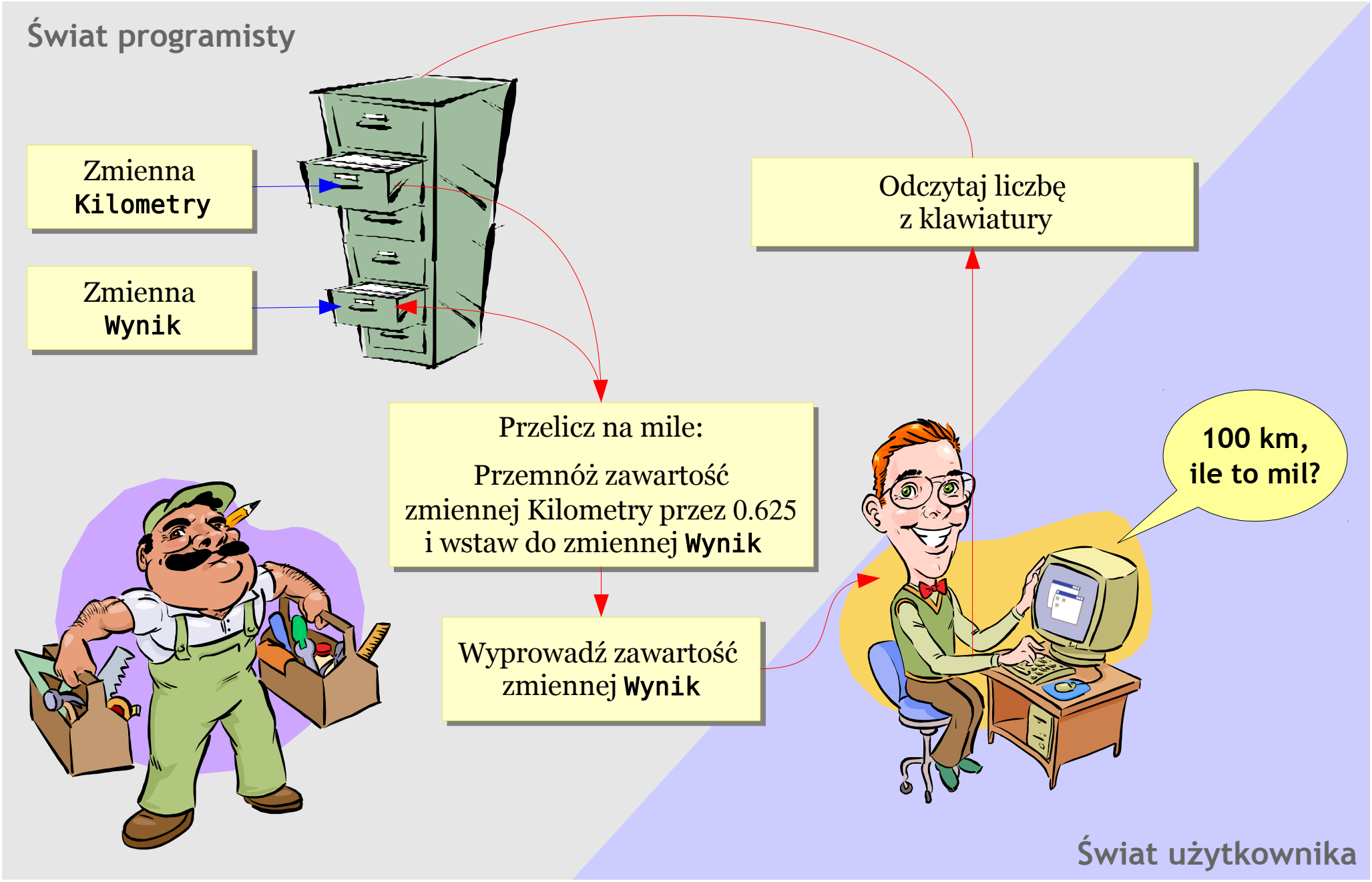
Świat użytkownika

Copyright © Roman Simiński

Zmienne a przeliczenie odległości



Potrzebna jest jeszcze jedna zmienna do przechowania wyniku obliczeń



Copyright © Roman Simiński

Zmienne a przeliczenie odległości

Świat programisty

Zmienna `Kilometry`

Zmienna `Wynik`

Odczytaj liczbę z klawiatury

Czy to już jest algorytm?

Tak! Tylko algorytmy zwykle się zapisuje inaczej... .

100 km, ile to mil?

Wyprowadź zawartość zmiennej `Wynik`

Świat użytkownika

Copyright © Roman Simiński

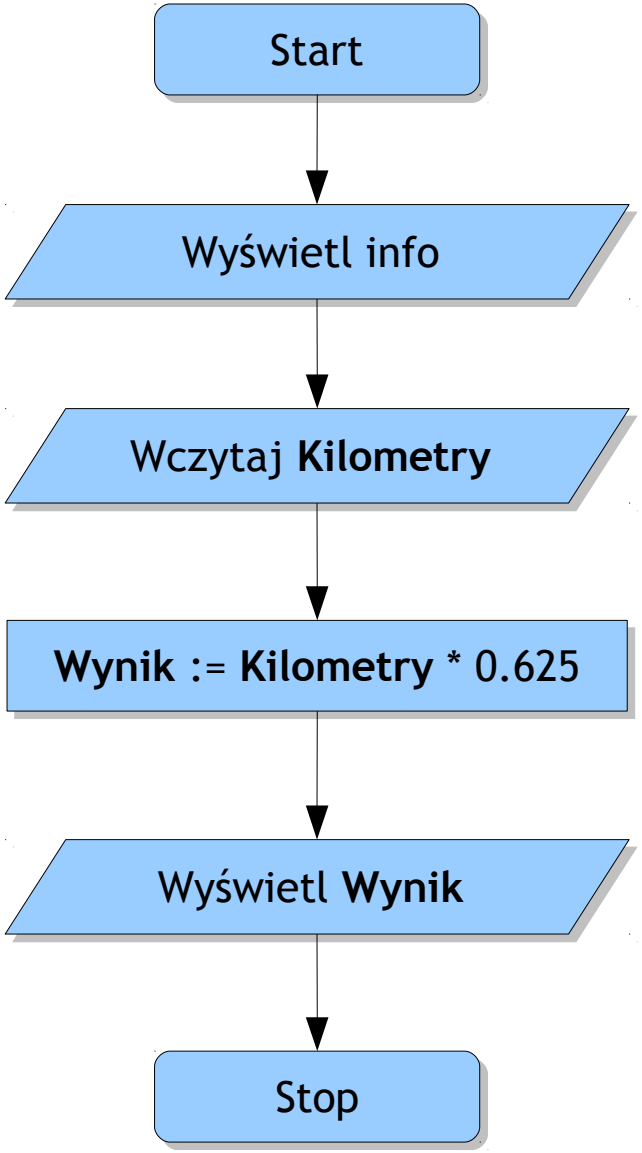
Algorytm przeliczenia odległości — opis słowny

Algorytm

- Niech **`Kilometry`** i **`Wynik`** będą zmiennymi typu liczba rzeczywista.
- Wyświetl informacje o programie.
- Wyświetl tekst zachęty do wprowadzenia liczby kilometrów.
- Odczytaj liczbę z klawiatury, zapisz w zmiennej **`Kilometry`**.
- Przelicz na mile: **`Wynik`** = zawartość zmiennej **`Kilometry`** pomnożonej przez 0.625.
- Wyprowadź **`Wynik`** na ekran.

Copyright © Roman Simiński

Algorytm przeliczenia odległości — schemat blokowy



Copyright © Roman Simiński

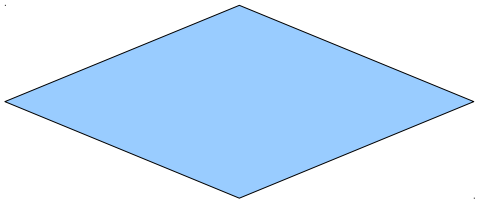
Podstawowe elementy schematów blokowych

Bloki początku i końca algorytmu:

Blok operacji wejścia/wyjścia

Blok wykonawczy

Blok decyzyjny



Copyright © Roman Simiński

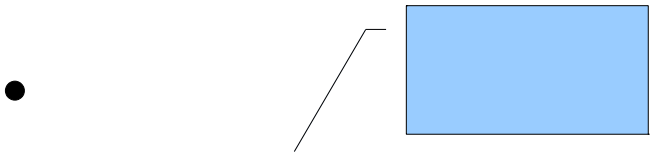
Podstawowe elementy schematów blokowych

Blok operacji wykonywanej poza rozważanym algorytmem

Łączniki stronnicowe

Łącznik międzystronnicowy

Punkt łączący i komentarz objaśniający



Copyright © Roman Simiński

Zanim napiszemy pierwszy program...

- Programy pisane w języku C/C++ są dzielone na bloki zwane *funkcjami*.
- Funkcje w programowaniu przypominają nieco funkcje matematyczne.
- Definicja funkcji określa jej *nazwę*, *argumenty* oraz *ciało*.

$$g(x) = 2x + 5$$

Copyright © Roman Simiński

Funkcja w matematyce

- Programy pisane w języku C/C++ są dzielone na bloki zwane *funkcjami*.
- Funkcje w programowaniu przypominają nieco funkcje matematyczne.
- Definicja funkcji określa jej *nazwę*, *argumenty* oraz *ciało*.

$$g(x) = 2x + 5$$

Nazwa funkcji

Argument

Ciało funkcji

Copyright © Roman Simiński

Definicja funkcji a jej wywołanie

- Zdefiniowana funkcja, to *coś*, co potencjalnie może *zadziałać*.
- Aby funkcja zadziałała, należy jej *użyć*, inaczej mówiąc *wywołać*.

$$g(x) = 2x + 5$$

$$z = \boxed{g(2)} + 5$$

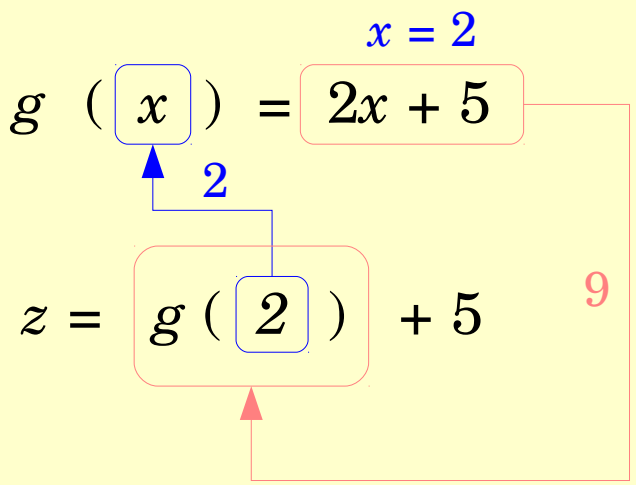
Wywołanie funkcji,
jej rezultatem jest liczba

- Wywołanie funkcji powoduje *wykonanie jej ciała* oraz *udostępnienie* w miejscu wywołania rezultatu funkcji.
- *Rezultat* funkcji w matematyce ma zwykle wartość *liczbową*.

Copyright © Roman Simiński

Przekazywanie argumentów

- W trakcie wywołania funkcji argument, wstępujący w wywołaniu wstawiany jest do argumentu funkcji.



Copyright © Roman Simiński

Zapamiętajmy...

Funkcja to coś, co posiada swoją *nazwę*.

Używając tej nazwy, można funkcję *wywołać*. Wywołanie funkcji powoduje wykonanie jej *ciała*.

W matematyce to zwykle wyznaczenie pewnej wartości liczbowej, która jest *rezultatem funkcji*.

Ciało funkcji określa *ciąg operacji*, które należy wykonać aby wyznaczyć rezultat funkcji.

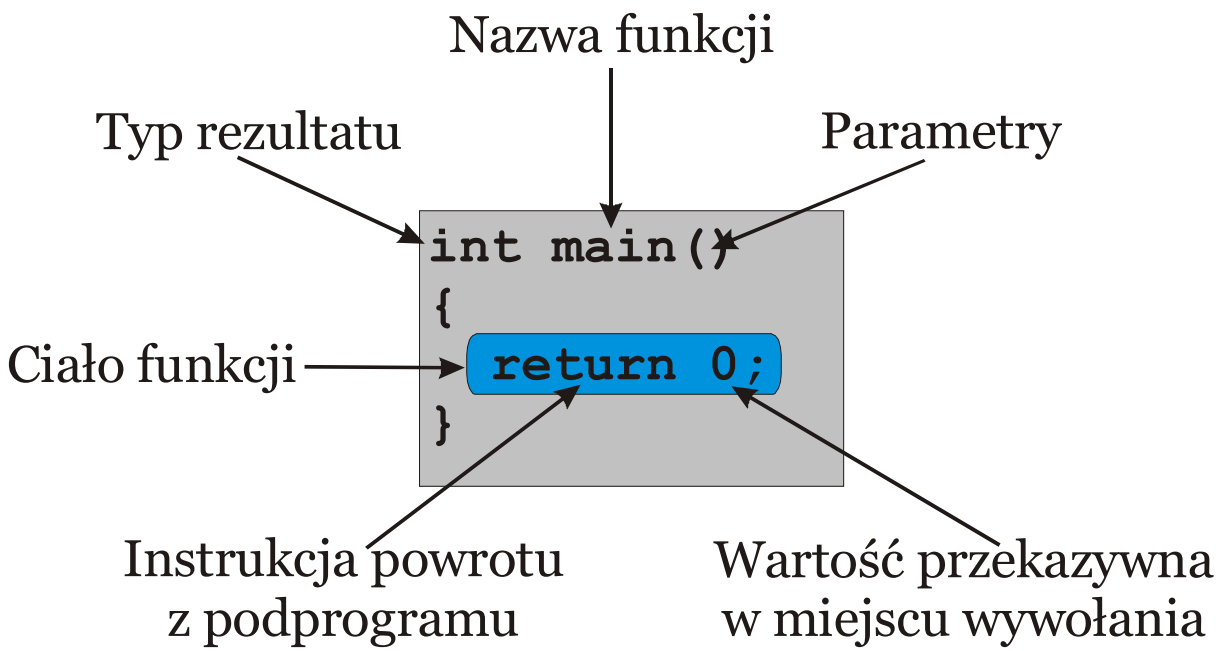
Rezultat funkcji jest udostępniany w *miejscu wywołania*, „komuś”, kto tę funkcję wywołał.

Copyright © Roman Simiński

Od algorytmu do programu — program, który robi nic

```
int main()
{
  return 0;
}
```

`int` (*integer*) to podstawowy typ całkowitoliczbowy



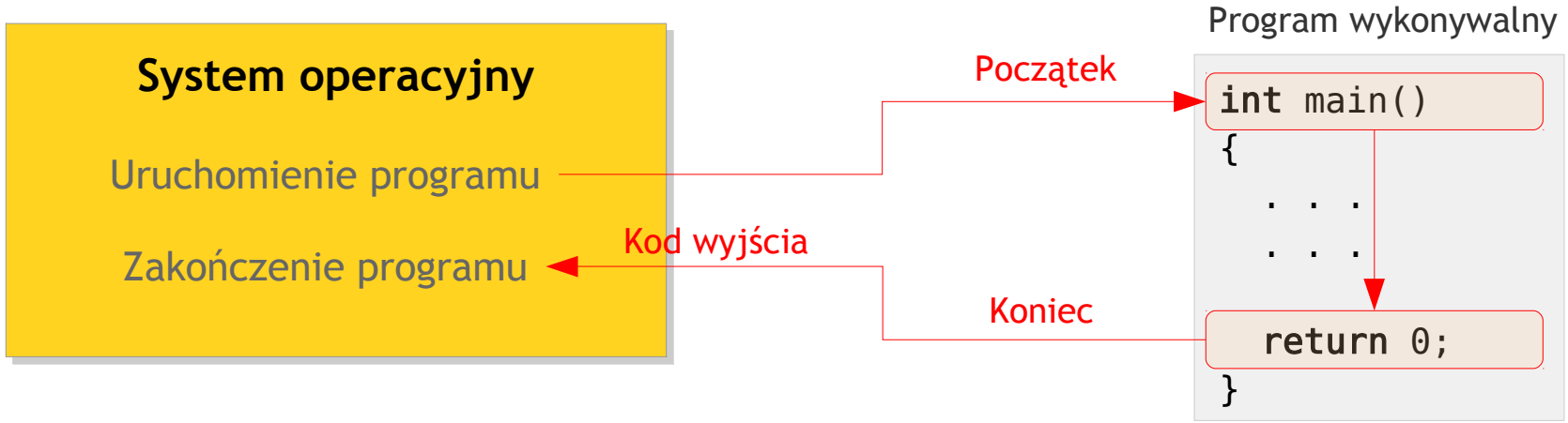
Każdy program C/C++musi posiadać funkcję o nazwie **`main`**, stanowiącą tzw. *punkt wejściowy* programu.

Od niej się rozpoczyna wykonanie programu napisanego w języku C/C++.

Copyright © Roman Simiński

Od algorytmu do programu — main a system operacyjny

- Funkcja `main` stanowi punkt programu, od którego zaczyna się jego wywołanie.
- *Upraszczając*, można powiedzieć, że funkcja `main` jest wywoływana przez system operacyjny (nie zawsze jednak tak jest).
- Wartość, będąca rezultatem funkcji *main* jest przekazywana systemowi operacyjnemu, stanowiąc *kod wyjścia* programu.
- Kod wyjścia programu może być wykorzystywany w *skryptach systemu operacyjnego* oraz jako środek *komunikacji pomiędzy procesami*.



Od algorytmu do programu — rezultat funkcji raz jeszcze

Zamiast rezultatu w postaci bezwzględnych wartości można wykorzystać symbole:

- `EXIT_SUCCESS` — oznacza bezbłędne zakończenie programu.
- `EXIT_FAILURE` —zakończenie z informacją o błędzie.

Stosowanie tych symboli jest zalecane przez standard *POSIX*. *POSIX* standaryzuje m.in. interfejs programistyczny, dotyczy głównie systemów klasy *UNIX*.

Niestety, program zapisany tak:

```
int main()
{
  return EXIT_SUCCESS;
}
```

Nie skompiluje się prawidłowo:

```
Message
In function `int main()':
`EXIT_SUCCESS' undeclared (first use this function)
(Each undeclared identifier is reported only once for each function it appears in.)
[Build Error] [main.o] Error 1
```

Od algorytmu do programu — włączanie plików nagłówkowych

Aby symbole:

- **EXIT_SUCCESS**
- **EXIT_FAILURE**

były dostępne w programie, należy włączyć do programu odpowiedni *plik nagłówkowy*.

Pliki nagłówkowe zawierają „wizytówki” elementów zapisanych w bibliotekach języka C/C++.

Dyrektywa **#include** powoduje włączenie do kodu źródłowego programu w języku C/C++ definicji i deklaracji niezbędnych dla kompilacji.

```
#include <iostream>

int main()
{
  return EXIT_SUCCESS;
}
```

Dyrektywa #include powoduje włączenie zawartości pliku *iostream*, zawierającego definicje niezbędne do obsługi standardowego wejścia-wyjścia w języku C++, również definicje symboli *EXIT_SUCCESS* i *EXIT_FAILURE*

Copyright © Roman Simiński

Od algorytmu do programu — przestrzeń nazw std jako domyślna

```
#include <iostream>
using namespace std;

int main()
{
  return EXIT_SUCCESS;
}
```

W języku C++ istnieją *przestrzenie nazw*. Pozwalają one na powtarzanie tych samych nazw w różnych przestrzeniach. Ta linia programu mówi, że będziemy korzystali ze standardowej przestrzeni nazw.

- Określenie domyślnej przestrzeni nazw (tutaj *std*) pozwoli na skrócenie zapisu pewnych odwołań — nie trzeba będzie podawać pełnych nazw, np.:

```
cout    zamiast    std::cout
cin     zamiast    std::cin
endl    zamiast    std::endl
```

Copyright © Roman Simiński

Od algorytmu do programu — wyprowadzanie komunikatów

```
#include <iostream>
using namespace std;

int main()
{
  cout << "Przeliczanie odleglosci wyrazonej w kilometrach na mile" << endl;

  return EXIT_SUCCESS;
}
```

Wyprowadzenie komunikatu do standardowego, znakowego strumienia wyjściowego programu reprezentowanego przez obiekt (*cout*), z przejściem do nowego wiersza (*endl*)

- **`cout`** — obiekt reprezentujący standardowy strumień wyjściowy programu.
- Wyprowadzanie danych odbywa się z wykorzystaniem operatora **<<**, zwanego *wstawiaczem* (ang. *inserter*). Dane są wyprowadzane zgodnie z ich typem, istnieje możliwość formatowania postaci wyjściowej.
- Manipulator **`endl`** powodują opróżnienie bufora strumienia **`cout`**, wstawia znak nowej linii przed opróżnieniem bufora.

Copyright © Roman Simiński

Od algorytmu do programu — literały łańcuchowe

```
#include <iostream>
using namespace std;

int main()
{
  cout << "Przeliczanie odleglosci wyrazonej w kilometrach na mile" << endl;

  return EXIT_SUCCESS;
}
```

Literały będące łańcuchami znaków zapisywane są w C++ pomiędzy znakami cudzysłowu ”, znaki te są ogranicznikiem i nie wchodzą w skład takiego literału.

```
Przeliczanie odleglosci wyrazonej w kilometrach na mile
_
```

Kursor

- Jeżeli wewnątrz łańcucha znakowego ma wystąpić znak cudzysłwu, należy go wpisać z poprzedzającym znakiem \ (*backslash*):

```
”Hans Kloss zwany \”J23\””
```

Copyright © Roman Simiński

Od algorytmu do programu — definiujemy zmienne

```
#include <iostream>
using namespace std;

int main()
{
  float kilometry, wynik;

  cout << "Przeliczanie odleglosci wyrazonej w kilometrach na mile" << endl;

  return EXIT_SUC
}
```

Zmienne trzeba zadeklarować — tzn. poinformować kompilator o ich nazwie i typie. Zmienne w C mogą być deklarowane na początku każdego bloku określonego nawiasami { }, w C++ mogą być deklarowane w dowolnym miejscu dozwolonym syntaktyką języka.

- *float* to typ zmiennopozycyjny, służący do reprezentowania liczb rzeczywistych pojedynczej precyzji.
- *double* to typ zmiennopozycyjny, służący do reprezentowania liczb rzeczywistych podwójnej precyzji precyzji.
- *int* to typ całkowitoliczbowy, służący do reprezentowania liczb całkowitych ze znakiem, o zakresie zależnym od implementacji.

Copyright © Roman Simiński

Od algorytmu do programu — zmienne

```
#include <iostream>
using namespace std;

int main()
{
  float kilometry, wynik;

  cout << "Przeliczanie odleglosci wyrazonej w kilometrach na mile" << endl;
  cout << "Podaj odleglosc w kilometrach: " << flush;

  return EXIT_SUCCESS;
}
```

Manipulator *flush* powoduje opróżnienie bufora strumienia cout.

Wyprowadzenie napisu do strumienia wyjściowego

```
Przeliczanie odleglosci wyrazonej w kilometrach na mile
Podaj odleglosc w kilometrach: _
```

Od algorytmu do programu — zmienne

```
#include <iostream>
using namespace std;

int main()
{
  float kilometry, wynik;

  cout << "Przeliczanie odleglosci wyrazonej w kilometrach na mile" << endl;
  cout << "Podaj odleglosc w kilometrach: " << flush;

  cin >> kilometry;

  return EXIT_SUCCESS;
}
```

Odczytanie znaków z klawiatury — do naciśnięcia klawisza *Enter* — oraz próba konwersji do postaci liczby rzeczywistej a następnie wstawienie tej liczby do zmiennej *kilometry*.

- `cin` — globalny obiekt *cin* umożliwia wczytywanie danych z wykorzystaniem operatora >> zwanego *wydobywaczem* (ang. *extractor*), który przeprowadza konieczne konwersje w trakcie pobierania danych.
- Operator >> pomija „białe” znaki w strumieniu wejściowym.

Copyright © Roman Simiński

Od algorytmu do programu — zmienne

```
#include <iostream>
using namespace std;

int main()
{
  float kilometry, wynik;

  cout << "Przeliczanie odleglosci wyrazonej w kilometrach na mile" << endl;
  cout << "Podaj odleglosc w kilometrach: " << flush;

  cin >> kilometry;

  wynik = kilometry * 0.625;

  cout << "To w milach: ";
  cout << wynik;

  return EXIT_SUCCESS;
}
```

Wyznacz odległość w milach, wstaw rezultat do zmiennej *wynik* (instrukcja przypisania)

Wyprowadź komunikat, dołącz zawartość zmiennej *wynik*

- Instrukcja przypisania powoduje nadanie zmiennej występującej po lewej stronie symbolu operatora przypisania = wartości wyrażenia, umieszczonego po stronie prawej. Operator przypisania oznaczany jest również symbolem ← lub :=

Copyright © Roman Simiński

Od algorytmu do programu — oczekiwanie na zakończenie programu

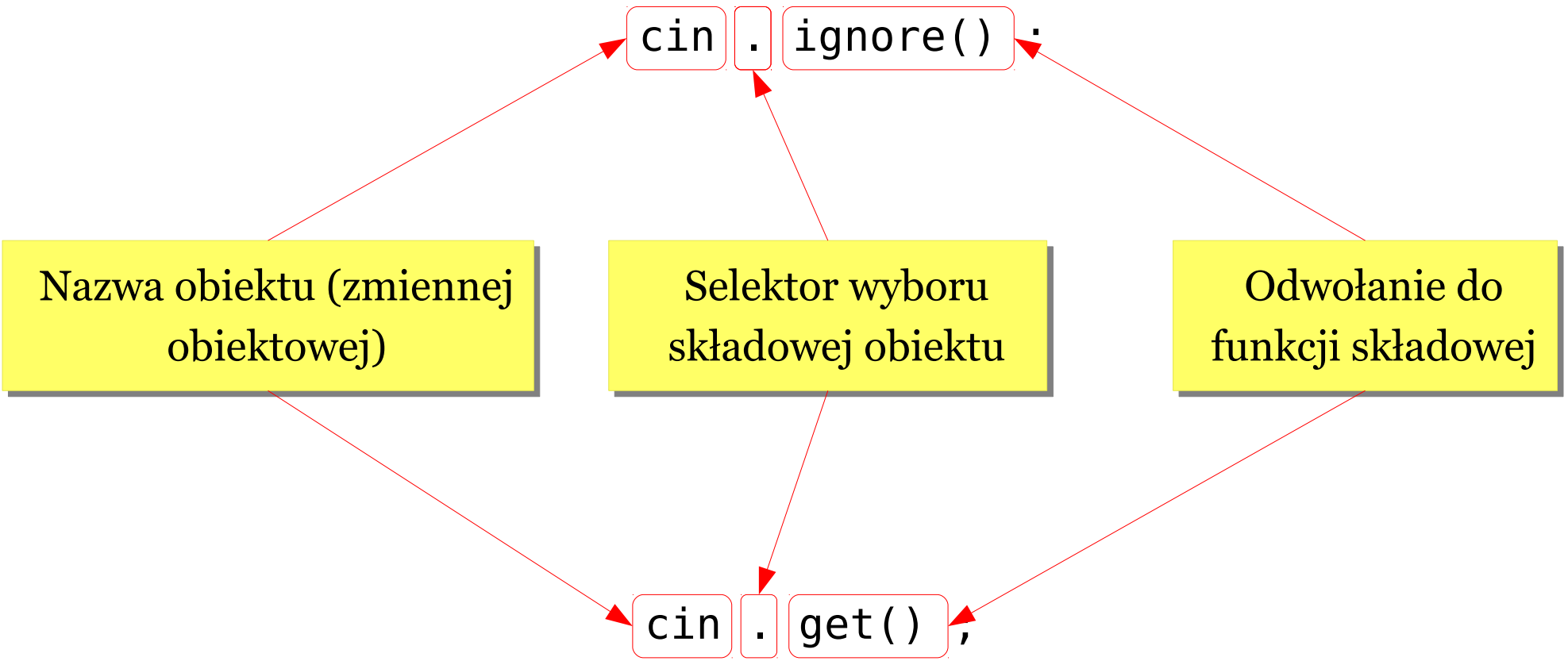
```
#include <iostream>
using namespace std;

int main()
{
  float kilometry, wynik;

  cout << "Przeliczanie odleglosci wyrazonej w kilometrach na mile" << endl;
  cout << "Podaj odleglosc w kilometrach: " << flush;

  cin >> kilometry;

  wynik = kilometry * 0.625;

  cout << "To w milach: ";
  cout << wynik;

  cout << endl << "Nacisnij Enter by zakonczyc program...";

  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

Informacja dla użytkownika

cin.ignore() pomija znak *Enter* pozostały w buforze po wprowadzeniu liczby kilometrów. *cin.get*() pobiera znak ze strumienia wejściowego.

Copyright © Roman Simiński

Od algorytmu do programu — cin to obiekt, ale o tym kiedy indziej



Copyright © Roman Simiński

Od algorytmu do programu — zmienna *wynik* jest niepotrzebna

```
. . .
cin >> kilometry;

wynik = kilometry * 0.625;

cout << "To w milach: ";
cout << wynik;

cout << endl << "Nacisnij Enter by zakonczyc program..." << endl;
cin.ignore();
cin.get();
. . .
```

```
. . .
cin >> kilometry;

cout << "To w milach: ";
cout << kilometry * 0.625;

cout << endl << "Nacisnij Enter by zakonczyc program..." << endl;
cin.ignore();
cin.get();
. . .
```

Copyright © Roman Simiński

Od algorytmu do programu — inne wyprowadzanie wyników

```
cout << "To w milach: "
cout << kilometry * 0.625;
```

```
cout << "To w milach: " << kilometry * 0.625;
```

Inna możliwość sformatowania wyświetlenia wyników:

```
cout << "Odleglosc " << kilometry << " km to w milach " << kilometry * 0.625 << endl;
```

```
Odleglosc 100 km to w milach 62.5
```

Copyright © Roman Simiński

Od algorytmu do programu — wersja końcowa

```
#include <iostream>
using namespace std;

int main()
{
  float kilometry;

  cout << "Przeliczanie odleglosci wyrazonej w kilometrach na mile" << endl;
  cout << "Podaj odleglosc w kilometrach: ";

  cin >> kilometry;

  cout << "To w milach: " << kilometry * 0.625 << endl;

  cout << "Nacisnij Enter by zakonczyc program...";

  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

Copyright © Roman Simiński

Suplement I — nazwy zmiennych

- Programista może pewne elementy programu nazywać. Właściwy dobór nazw wprowadza dodatkowe informacje o nazywanym obiekcie i zwiększa czytelność programu. Nazwy muszą być identyfikatorami.
- ***Identyfikator*** to ciąg liter, cyfr i znaków podkreślenia rozpoczynający się od litery, przy czym znak podkreślenia traktowany jest jako litera.
- Polskie znaki we większości języków programowania nie są traktowane jako litery.
- Podane reguły znajdują zastosowanie we większości popularnych języków programowania.

Poprawne identyfikatory	Niepoprawne identyfikatory
`J23, PI, Delta, x, _y, Linia, Dlugosc`	`J 23, 2PI, 007, liczba linii, długość`

Copyright © Roman Simiński

Suplement I — nazwy zmiennych, cd. ...

- Nie ma normatywnych zaleceń odnośnie konwencji pisania identyfikatorów.
- Tradycyjnie jednak, w programach pisanych w języku C nazwy zmiennych i funkcji pisze się małymi literami, czasem ze znakiem podkreślanie w identyfikatorach będących zlepkami.
- W języku C++ znacznie częściej wykorzystuje się małe i duże litery.
- Programista ma zatem dowolność w dobieraniu identyfikatorów, warto jednak przyjąć jedną z popularnych i sprawdzonych konwencji.
- W C/C++ duże i małe litery są rozróżniane, brane pod uwagę są 31-32 znaki identyfikatora.

```
liczbalinii    liczba_linii   liczbaLinii    LiczbaLinii    Liczba_Linii
```

~~LICZBALINII~~

Copyright © Roman Simiński

Suplement II — słowa kluczowe

Słowa kluczowe to identyfikatory zastrzeżone i nie mogą być inaczej stosowane niż określa to standard języka.

Słowa kluczowe winny być pisane tak jak je podano, a więc wyłącznie z wykorzystaniem małych liter. Słowa kluczowe wg. normy ANSI C89:

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

Do języka C++ zostały przeniesione słowa kluczowe istniejące w języku C, dodano nowe:

asm	dynamic_cast	namespace	reinterpret_cast	try
bool	explicit	new	static_cast	typeid
catch	false	operator	template	typename
class	friend	private	this	using
const_cast	inline	public	throw	virtual
delete	mutable	protected	true	wchar_t

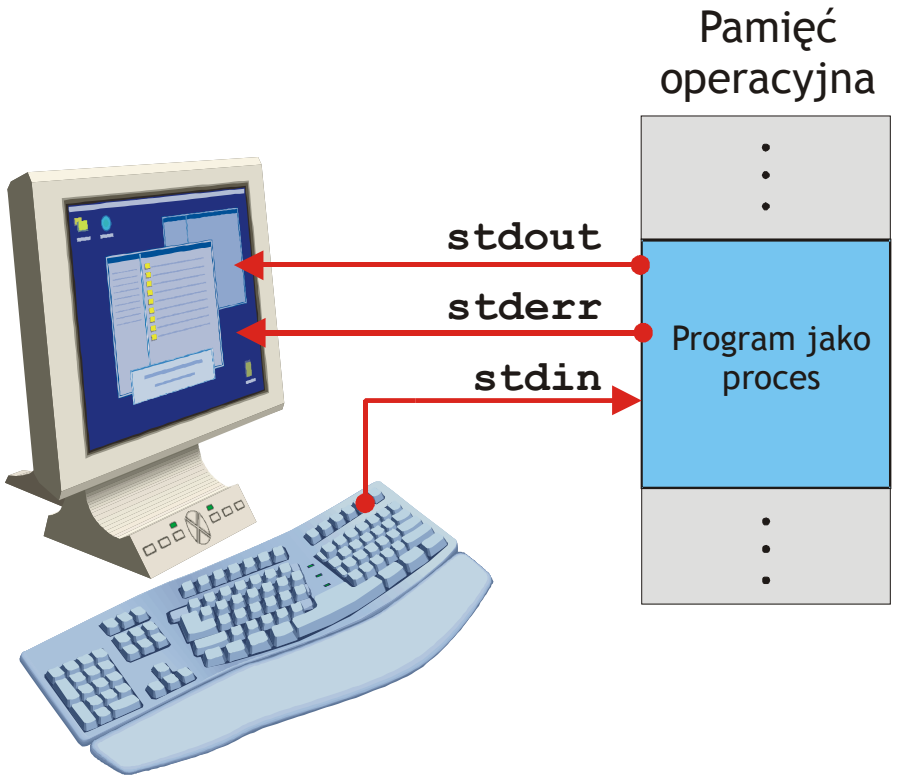
Copyright © Roman Simiński

Suplement III — co to: standardowe strumienie programu?

- *stdin — standardowy strumień wejściowy* programu, jest zwykle skojarzony z klawiaturą;
- *stdout — standardowy strumień wyjściowy* programu, jest zwykle skojarzony z ekranem monitora;
- *stderr — standardowy strumień wyjściowy błędów* programu, jest zwykle skojarzony również z ekranem monitora.

Strumienie *stdin*, *stdout* reprezentują normalny kanał komunikacji programu z użytkownikiem.

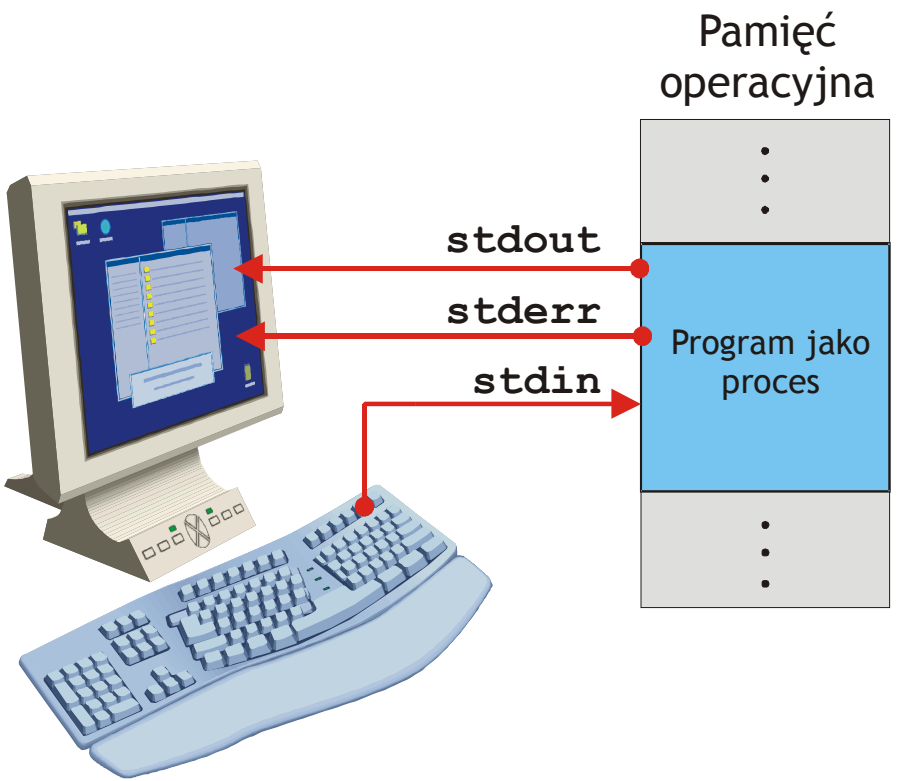
Strumień *stderr* zarezerwowany jest do wyświetlania komunikatów diagnostycznych programu.



Copyright © Roman Simiński

Suplement III — redyrekcja (przekierowanie) strumieni programu

Standardowe skojarzenie strumieni



Redyrekcja (przekierowanie) strumieni z poziomu systemu operacyjnego:

```
C:\>test.exe > wyjscie.txt
```

```
C:\>test.exe < wejscie.txt
```

```
C:\>test.exe < wejscie.txt > wyjscie.txt
```

Copyright © Roman Simiński

Suplement III — strumienie wejścia-wyjścia w języku C++

Strumienie *stdin*, *stdout* oraz *stderr* wykorzystywane są głównie w programach pisanych w języku C.

W języku C++ wykorzystuje się:

- *cin* — strumień reprezentujący standardowe wejście programu, odpowiada strumieniowi *stdin* z C. Strumień *cin* odczytuje dane i zapisuje je do odpowiednich zmiennych.
- *cout* — strumień reprezentujący standardowe wyjście programu. Odpowiada strumieniowi *stdout* z C.
- *cerr* — niebuforowany strumień wyjściowy błędów. Odpowiada strumieniowi *stderr* z C.
- *clog* — buforowany strumień wyjściowy błędów. Odpowiada strumieniowi *stderr* z C.

Wykorzystanie tych strumieni wymaga włączenia do programu pliku nagłówkowego *iostream*.

Copyright © Roman Simiński

Kolejny problem do rozwiązania

Problem

Zadaniem programu jest obliczanie, ile średnio litrów spala paliwa pojazd na trasie 100 km.

Wartość ta — zwana dalej średnim spalaniem — obliczana jest na podstawie ilości zużytego przez pojazd paliwa oraz liczby przejechanych kilometrów.

Scenariusz działania programu:

```
Obliczam ile Twoj pojazd spala paliwa na 100 km
Dystans: 500
Paliwo: 35
Spalanie 7.00 l na 100 km
Nacisnij Enter by zakonczyc program...
```

Kolejny problem do rozwiązania

Analiza

Średnie spalanie wyznaczamy z proporcji:

```
przejechany dystans [km]  –  ilość paliwa [litry]
                100 [km]  –  x [litry]
```

```
spalanie = ( 100 * ilość paliwa ) / przejechany dystans
```

- Do obliczenia średniego zużycia potrzebne są dwie wartości liczbowe, jak wyżej. Powinny to być liczby rzeczywiste.
- Program powinien zapytać użytkownika o te wartości, zapamiętać je, dokonać obliczeń oraz wyświetlić wynik.
- Program powinien reagować na niepoprawne dane wejściowe.

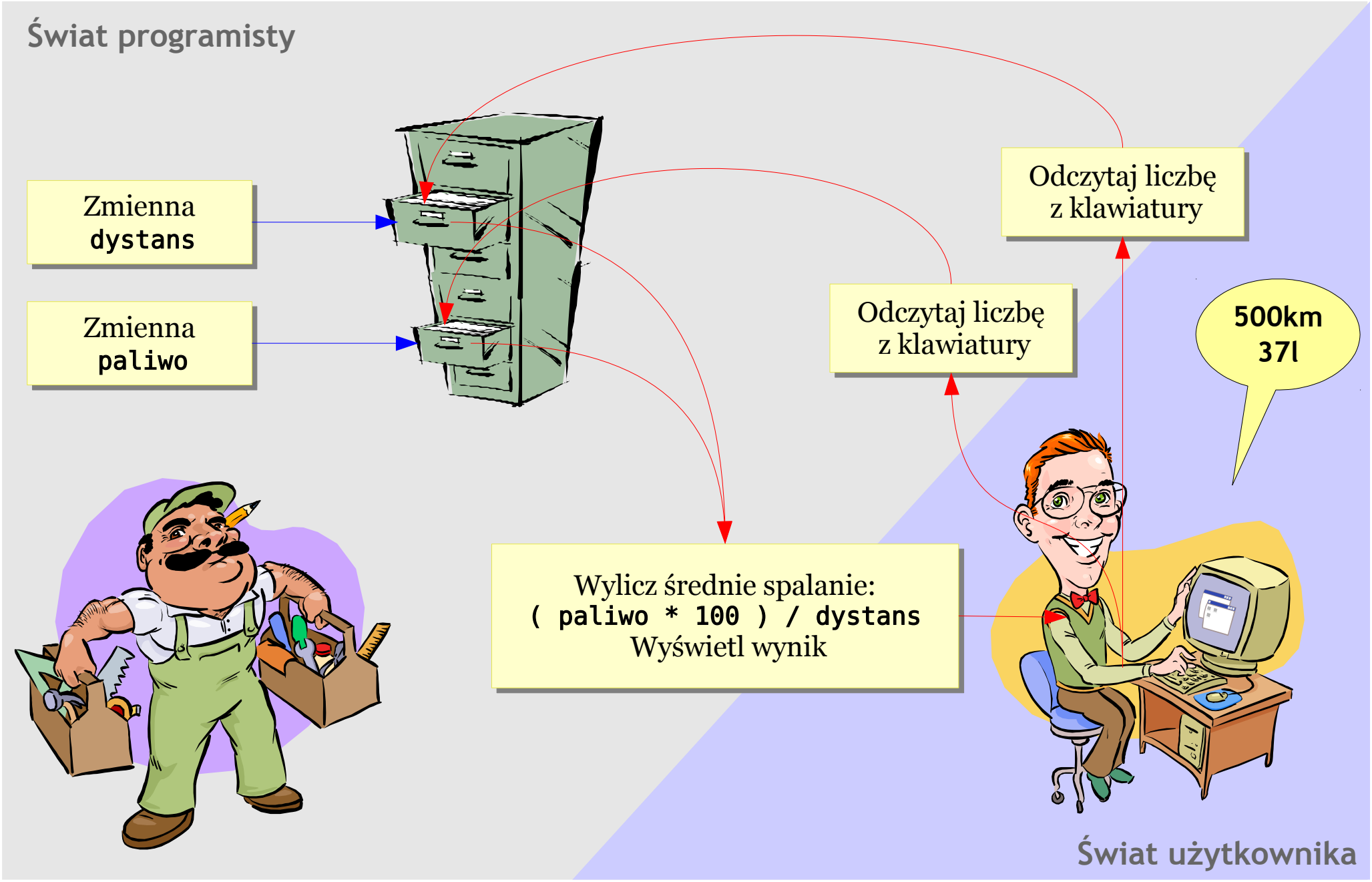
Copyright © Roman Simiński

Algorytm rozwiązania problemu — pierwsza, kiepska wersja

Wyświetl informację wstępną – do czego program służy
Wyświetl zachętę do wprowadzenia przejechanego dystansu
Wczytaj dystans
Wyświetl zachętę do wprowadzenia ilości paliwa
Wczytaj ilość paliwa
Oblicz średnie spalanie wg. wzoru i wyświetl wynik

Copyright © Roman Simiński

Algorytm rozwiązania problemu — ilustracja



Algorytm rozwiązania problemu — druga, niedoskonała wersja

```
Dane:
  Niech dystans to zmienna typu liczba rzeczywista
  Niech paliwo to zmienna typu liczba rzeczywista
Akcje:
  Wyświetl "Obliczam ile Twój pojazd spala paliwa na 100 km" Nowa linia

  Wyświetl "Dystans : "
  Wczytaj liczbę do zmiennej dystans

  Wyświetl "Paliwo : "
  Wczytaj liczbę do zmiennej paliwo

  Wyświetl "Spalanie "
  Wyświetl wartość wyrażenia ( paliwo * 100 ) / dystans
  Wyświetl "  l/100 km" Nowa linia

  Wyświetl " Naciśnij Enter by zakończyć program..." Nowa linia
  Czekaj na naciśnięcie klawisza Enter
```

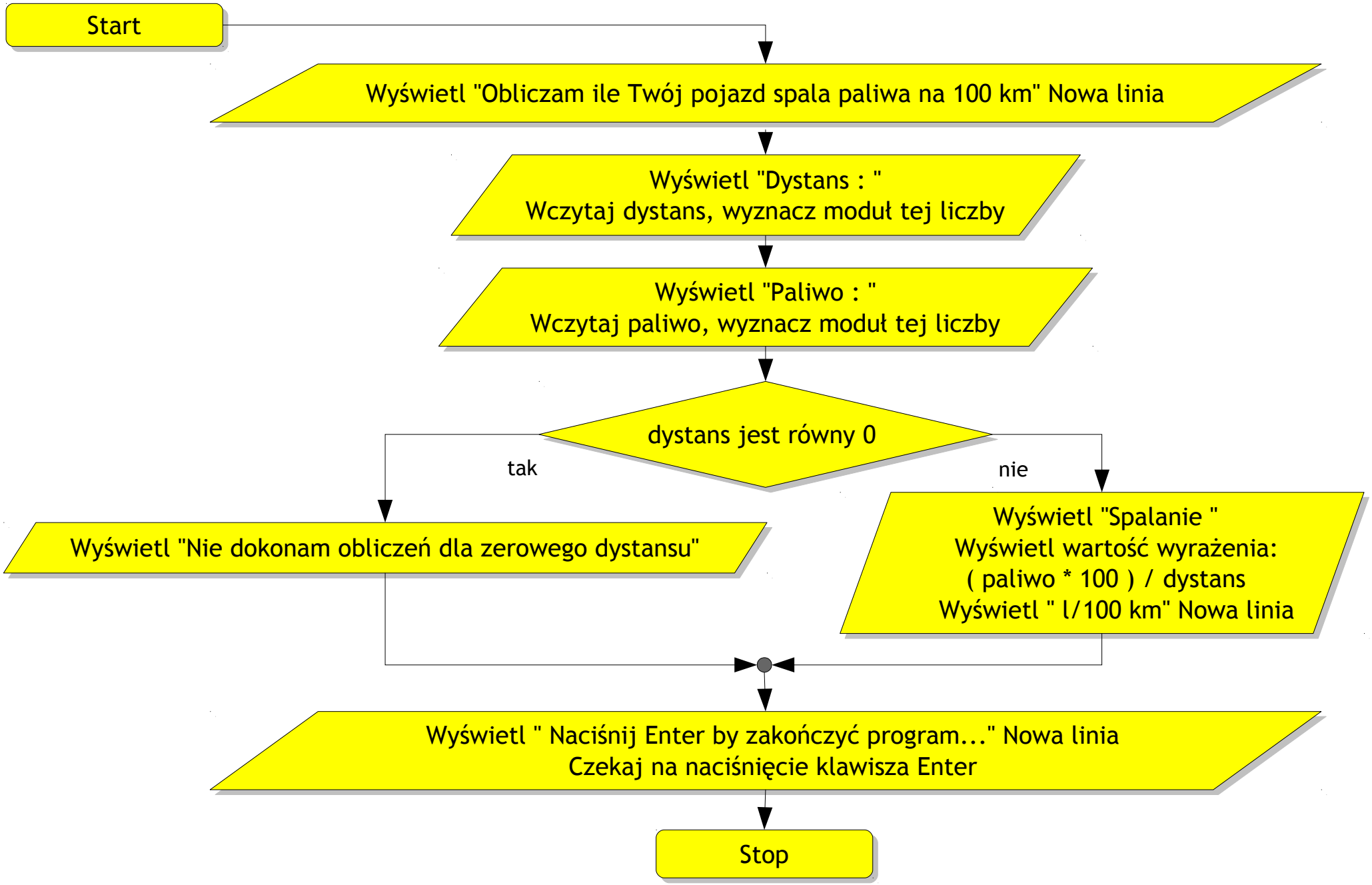
Copyright © Roman Simiński

Algorytm rozwiązania problemu — trzecia, przyzwoita wersja

```
Dane:
  Niech dystans to zmienna typu liczba rzeczywista
  Niech paliwo to zmienna typu liczba rzeczywista
Akcje:
  Wyświetl "Obliczam ile Twój pojazd spala paliwa na 100 km" Nowa linia

  Wyświetl "Dystans : "
  Wczytaj liczbę do zmiennej dystans, wyznacz moduł tej liczby

  Wyświetl "Paliwo : "
  Wczytaj liczbę do zmiennej paliwo, wyznacz moduł tej liczby

  Jeżeli dystans jest równy 0 To
    Wyświetl "Nie dokonam obliczeń dla zerowego dystansu"
  W przeciwnym przypadku
    Wyświetl "Spalanie "
    Wyświetl wartość wyrażenia ( paliwo * 100 ) / dystans
    Wyświetl " l/100 km" Nowa linia

  Wyświetl " Naciśnij Enter by zakończyć program..." Nowa linia
  Czekaj na naciśnięcie klawisza Enter
```

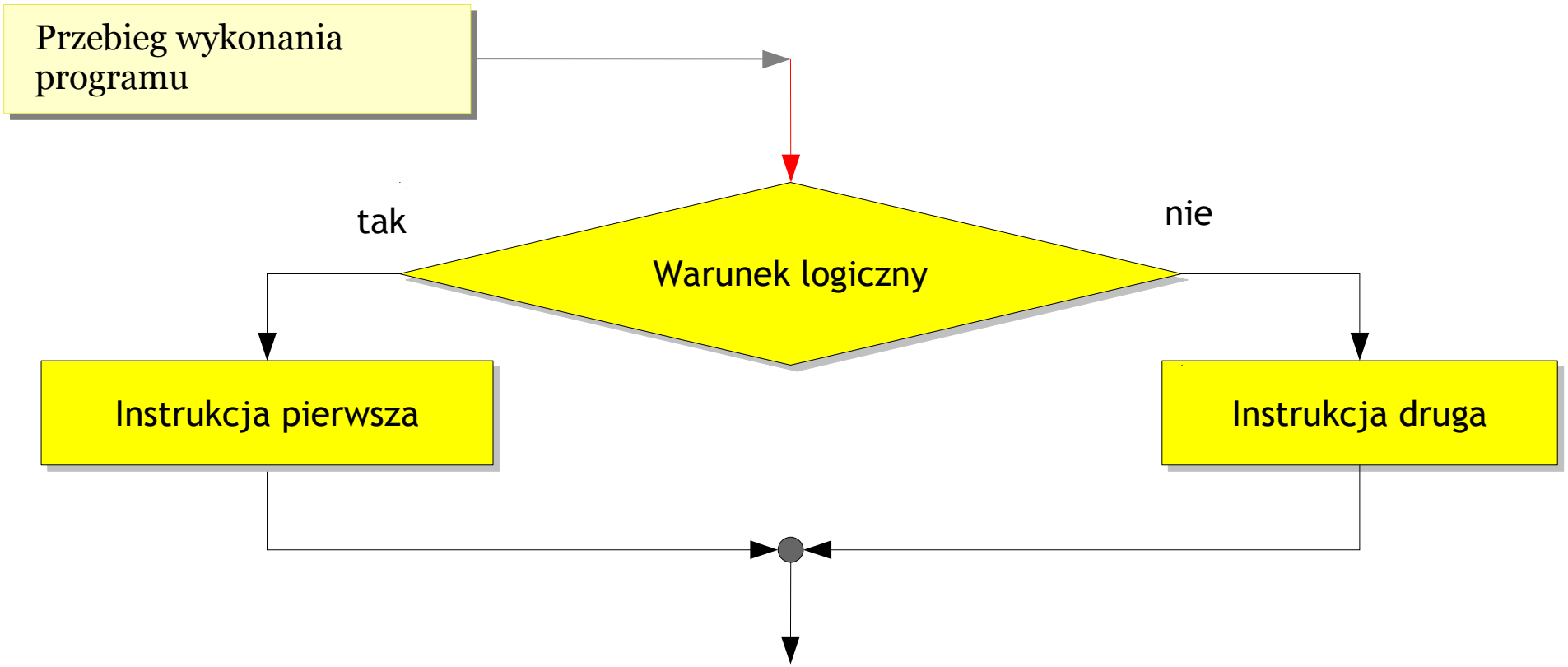
Copyright © Roman Simiński

Algorytm rozwiązania problemu — schemat blokowy



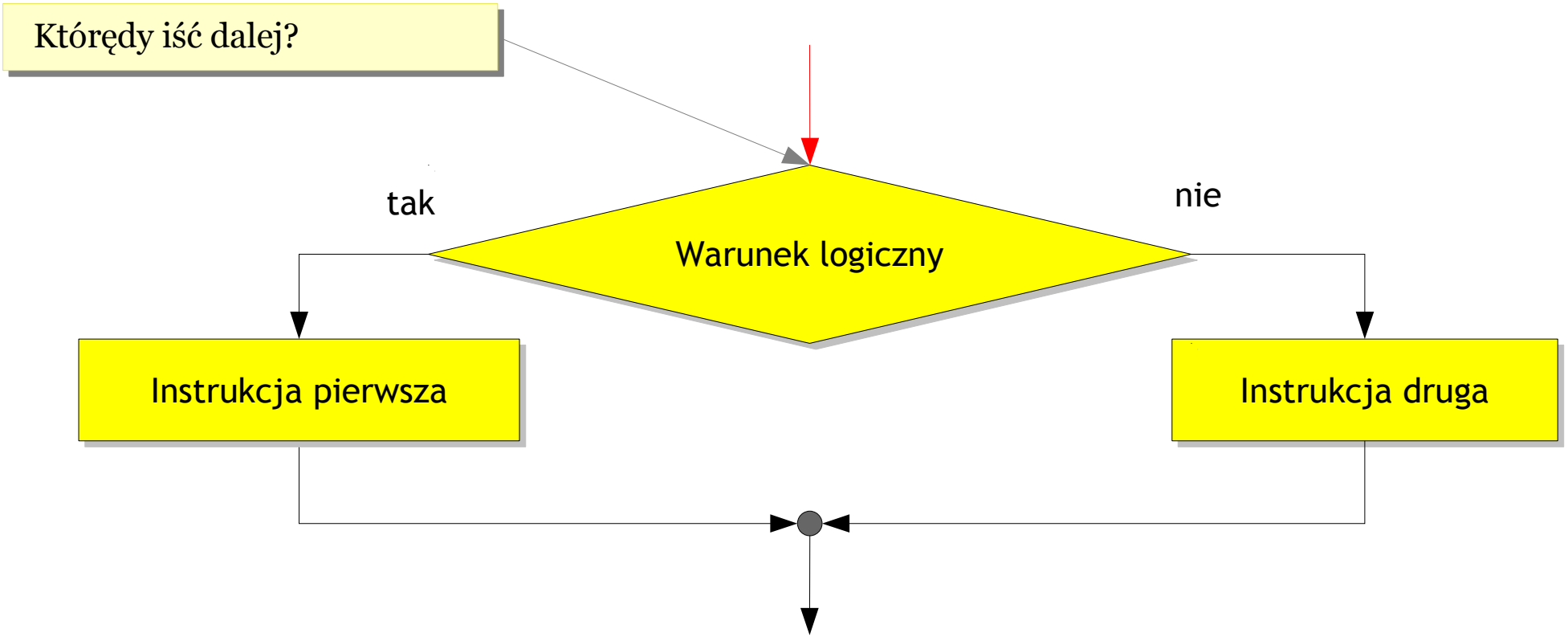
Copyright © Roman Simiński

Nowy element — bloczek decyzyjny



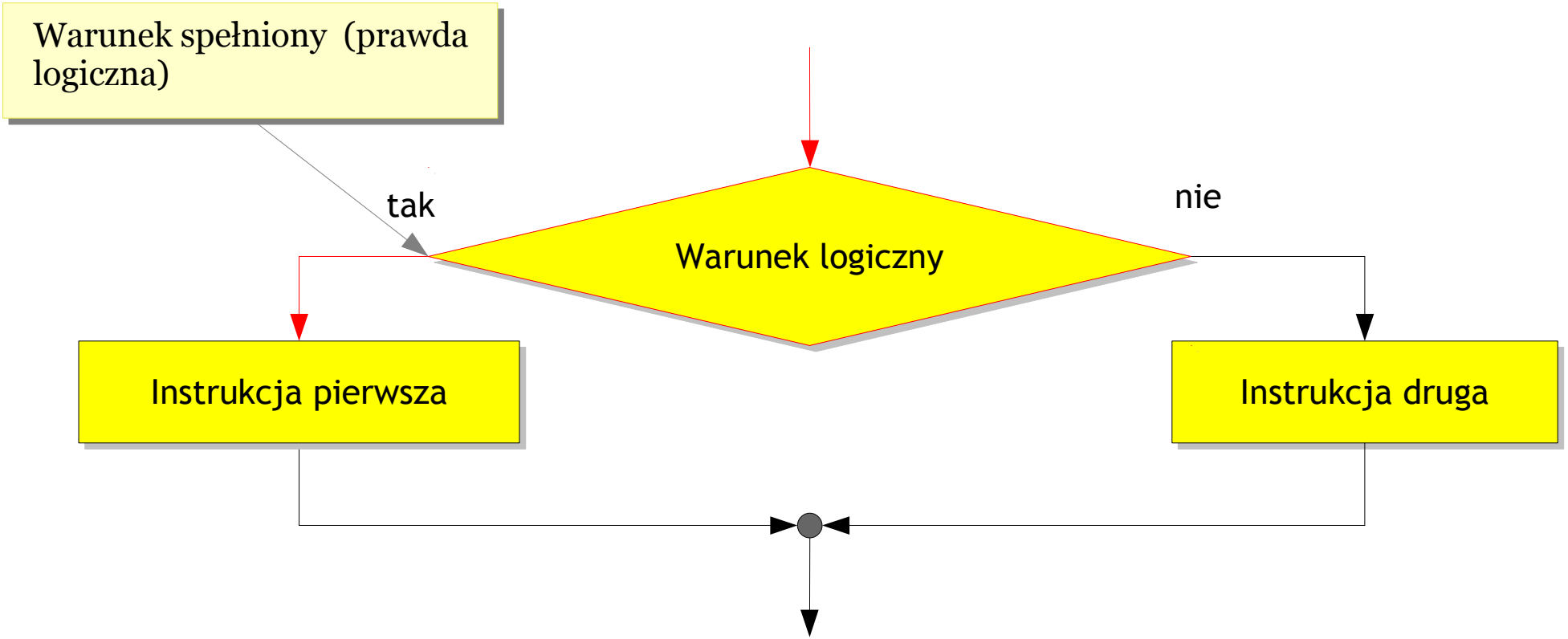
Copyright © Roman Simiński

Nowy element — bloczek decyzyjny



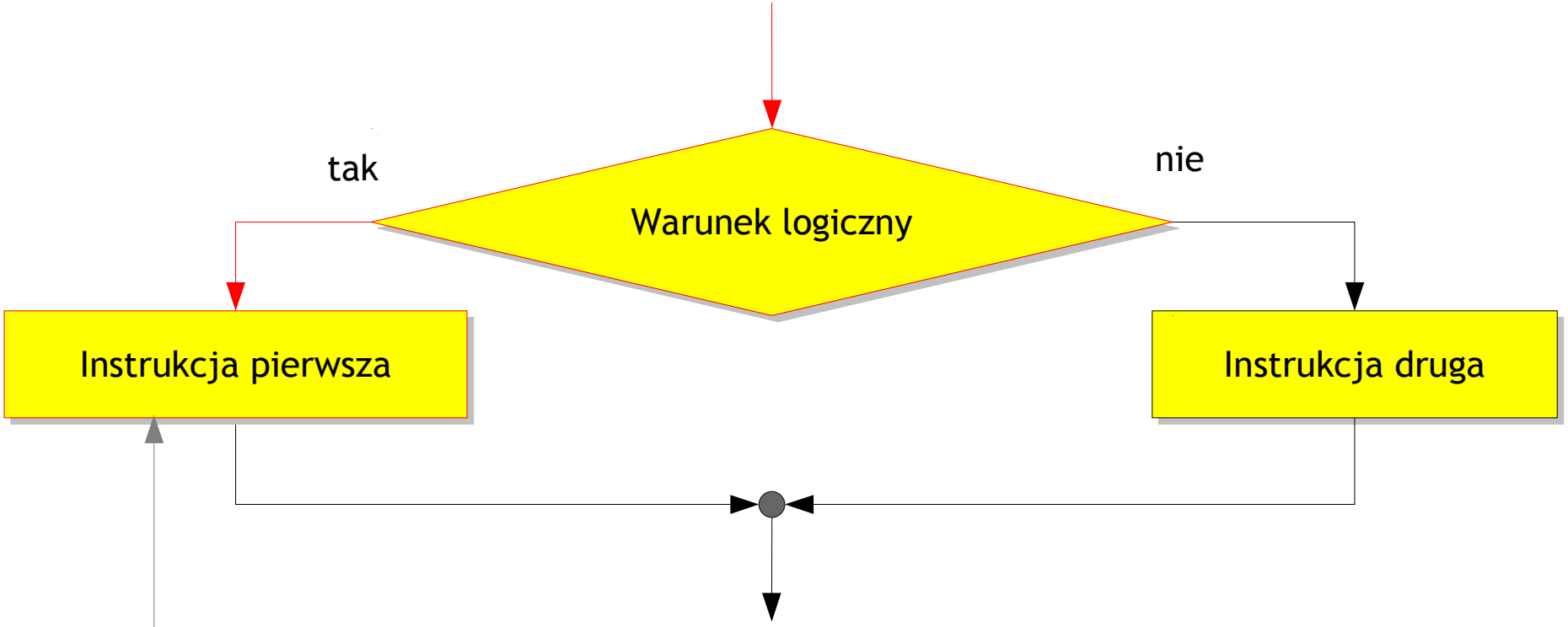
Copyright © Roman Simiński

Nowy element — bloczek decyzyjny



Copyright © Roman Simiński

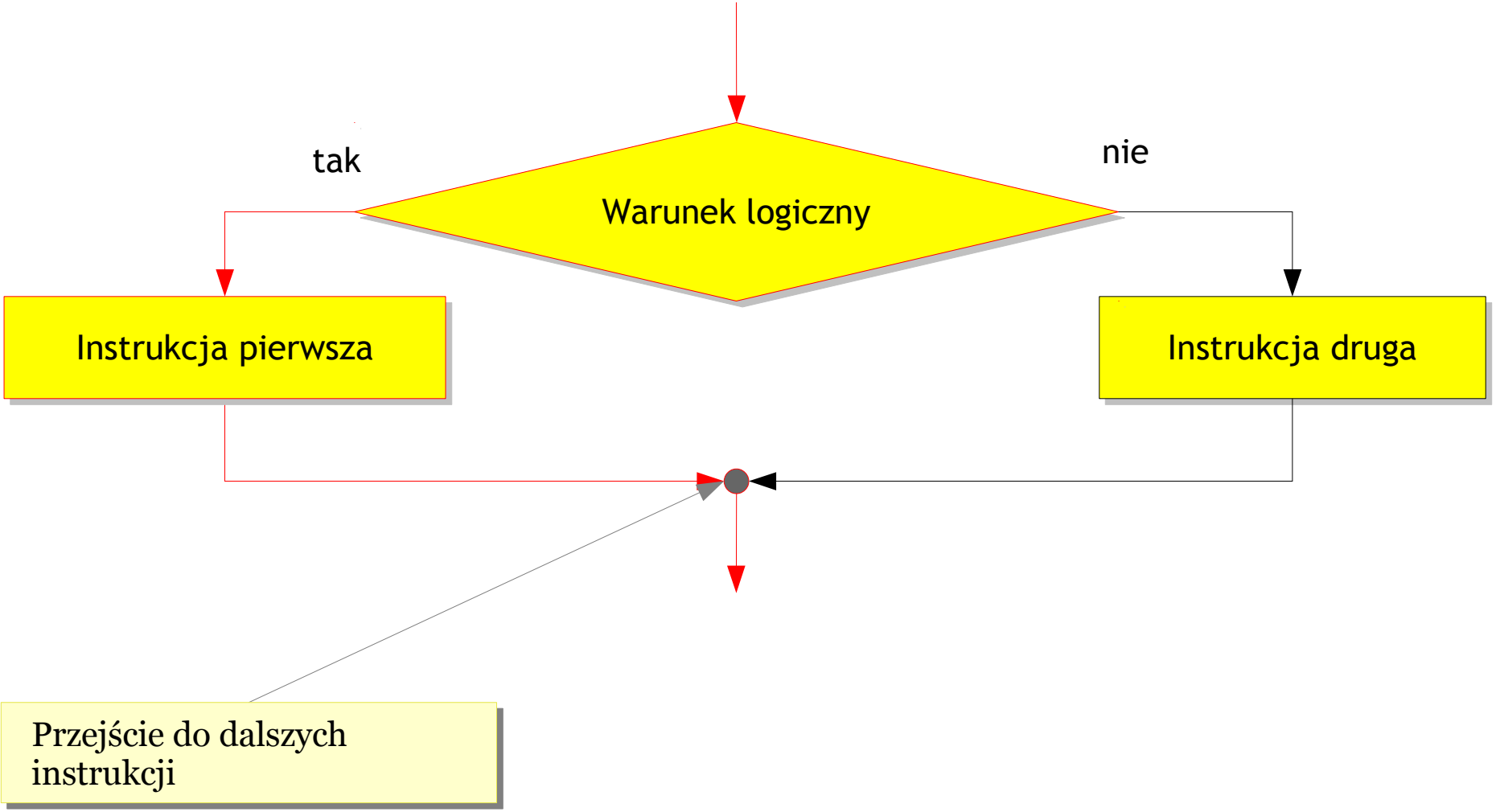
Nowy element — bloczek decyzyjny



Ta instrukcja zostanie wykonana

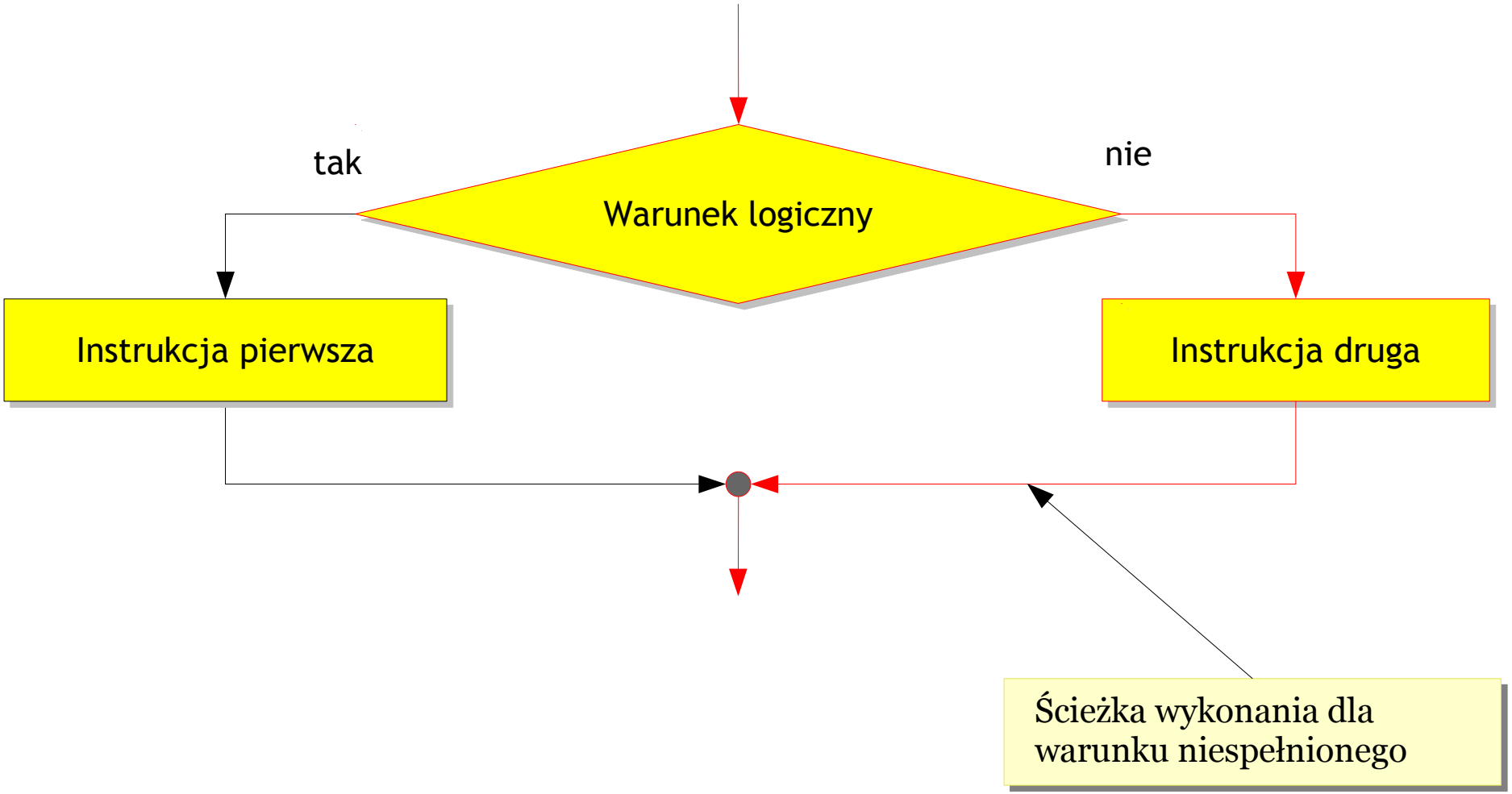
Copyright © Roman Simiński

Nowy element — bloczek decyzyjny



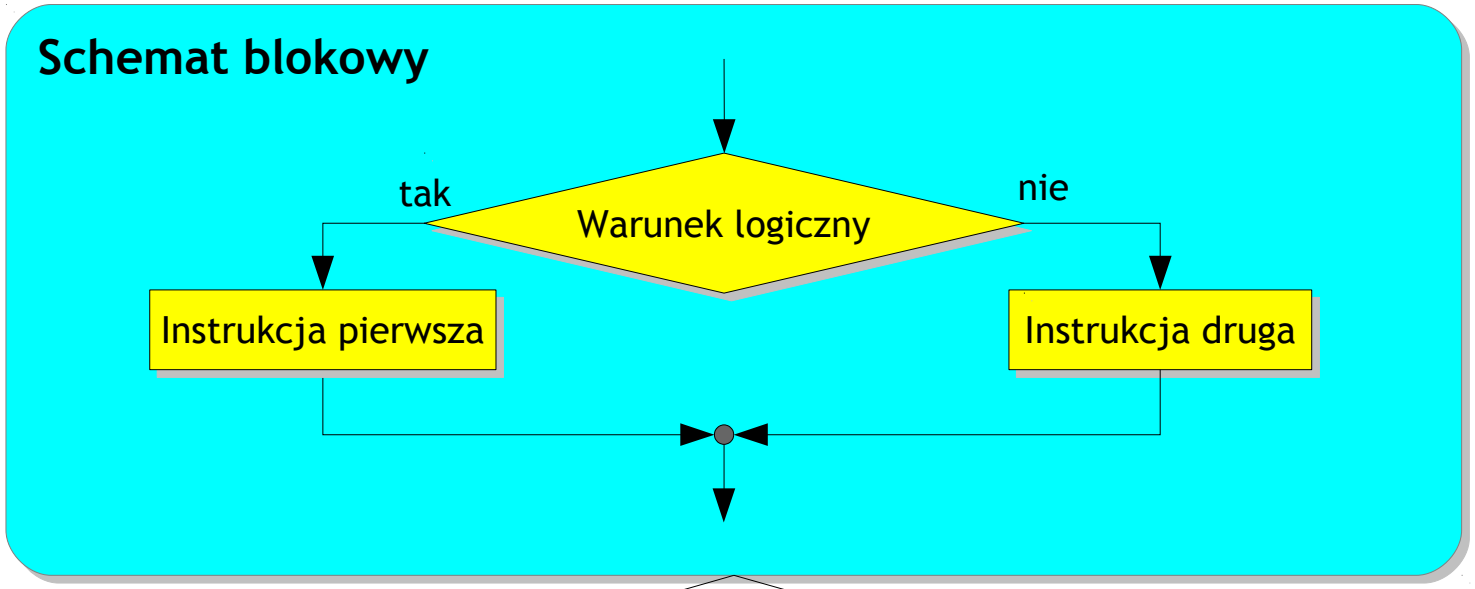
Copyright © Roman Simiński

Nowy element — bloczek decyzyjny



Copyright © Roman Simiński

Blok decyzyjny a instrukcja alternatywy



Pseudokod

Jeżeli Warunek logiczny **To**
  Instrukcja pierwsza
W przeciwnym przypadku
  Instrukcja druga

Język C/C++

if(Warunek logiczny)
  Instrukcja pierwsza
else
  Instrukcja druga

Copyright © Roman Simiński

Program wyznaczający średnie spalanie

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
  float dystans, paliwo;

  cout << endl << "Obliczam ile Twoj pojazd spala paliwa na 100 km" << endl;

  cout << "Dystans: " << flush;
  cin >> dystans;
  dystans = fabs( dystans );

  cout << "Paliwo: " << flush;
  cin >> paliwo;
  paliwo = fabs( paliwo );

  if( dystans == 0 )
    cout << "Nie dokonam obliczen dla zerowego dystansu" << endl;
  else
    cout << "Spalanie " << ( paliwo*100 ) / dystans << " l na 100 km" << endl;

  cout << "Nacisnij Enter by zakonczyc program..." << endl;
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

Plik nagłówkowy, który należy włączyć aby wykonywać operacje arytmetyczne – tutaj konieczny dla wykorzystania funkcji *fabs*

Wywołanie funkcji *fabs* – wyznaczenie wartości absolutnej (modułu) liczby *float* zapisanej w zmiennej *dystans* i *paliwo*

Copyright © Roman Simiński

Program wyznaczający średnie spalanie

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
  float dystans, paliwo;

  cout << endl << "Obliczam ile Twoj pojazd spala paliwa na 100 km" << endl;

  cout << "Dystans: " << flush;
  cin >> dystans;
  dystans = fabs( dystans );

  cout << "Paliwo: " << flush;
  cin >> paliwo;
  paliwo = fabs( paliwo );

  if( dystans == 0 )
    cout << "Nie dokonam obliczen dla zerowego dystansu" << endl;
  else
    cout << "Spalanie " << ( paliwo*100 ) / dystans << " l na 100 km" << endl;

  cout << "Nacisnij Enter by zakonczyc program..." << endl;
  cin.ignore();
  cin.get();

  return EXIT_SUCCESS;
}
```

Instrukcja alternatywy

Copyright © Roman Simiński

Instrukcja alternatywy pod lupą

Warunek sterujący wykonaniem iteracji
Operator == to *operator porównania*. Inaczej — testowania równej wartości.
Warunek jest prawdziwy, jeżeli lewa strona jest równa stronie prawej

```
if( dystans == 0 )
  cout << "Nie dokonam obliczen dla zerowego dystansu" << endl;
else
  cout << "Spalanie " << ( paliwo*100 ) / dystans << " l na 100 km" << endl;
```

- Operator porównania == (test na równość wartości):

```
int x, y;

x = 10;
y = 20;

if( x == y )
  cout << "Wartosci zmiennych x i y sa rowne";
else
  cout << "Wartosci zmiennych x i y sa rozne";
```

Copyright © Roman Simiński

Operator == a operator =

- Uwaga! Często zamiast *operatora porównania* == programiści mylnie używają *operatora przypisania* =.

```
int x;

x = 10;
. . .
if( x = 10 )
  cout << "Wartosc zmiennej x jest rowna 10";
else
  cout << "Wartosc zmiennej x jest rozna od 10";
```

Zmienna *x* otrzyma wartość 10! Wyrażenie jest prawidłowe, jego wartością będzie właśnie liczba 10. W języku C/C++ każda wartość niezerowa uznawana jest za prawdziwą

- Jeżeli jeden z argumentów jest wartością stałą, można zawsze zapisywać warunek w sposób specyficzny:

```
. . .
if( 10 == x )
  cout << "Wartosc zmiennej x jest rowna 10";
else
  cout << "Wartosc zmiennej x jest rozna od 10";
```

W przypadku pomyłkowego użycia operatora = kompilator zgłosi błąd, wyrażenie 10 = x jest bowiem nieprawidłowe

Copyright © Roman Simiński

Operator == a operator !=

- *Operatora porównania* == ma operator komplementarny — *operator testu na nierówność* **!=**.

```
. . .
if( x == 10 )
  cout << "Wartosc zmiennej x jest rowna 10";
else
  cout << "Wartosc zmiennej x jest rozna od 10";
```

Test na *równość* wartości, wyrażenie prawdziwe gdy zmienna *x* ma wartość *równą* 10

```
. . .
if( x != 10 )
  cout << "Wartosc zmiennej x jest rozna od 10";
else
  cout << "Wartosc zmiennej x jest rowna 10";
```

Test na *nierówność* wartości, wyrażenie prawdziwe gdy zmienna *x* ma wartość *różną* od 10

Copyright © Roman Simiński

Operator == i != a testowanie zerowego dystansu

- We większości przypadków *test na równość* można zastąpić *testem na nierówność*, należy jednak pamiętać o *zamianie wykonywanych instrukcji*.

```
if( dystans == 0 )
  cout << "Nie dokonam obliczen dla zerowego dystansu" << endl;
else
  cout << "Spalanie " << ( paliwo*100 ) / dystans << " l na 100 km" << endl;
```

```
if( dystans != 0 )
  cout << "Spalanie " << ( paliwo*100 ) / dystans << " l na 100 km" << endl;
else
  cout << "Nie dokonam obliczen dla zerowego dystansu" << endl;
```

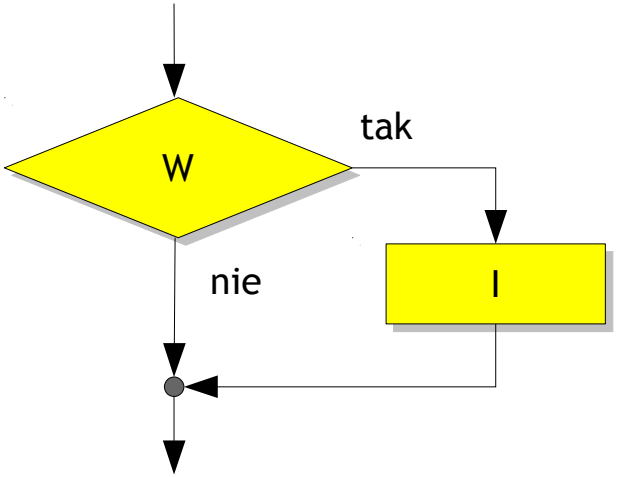
Copyright © Roman Simiński

Suplement IV — instrukcja alternatywy i warunkowa

- Instrukcja *alternatywy* i *warunkowa* należą go grupy *instrukcji sterujących wykonaniem programu*. Wspólnie są nazywane *instrukcjami warunkowymi*.

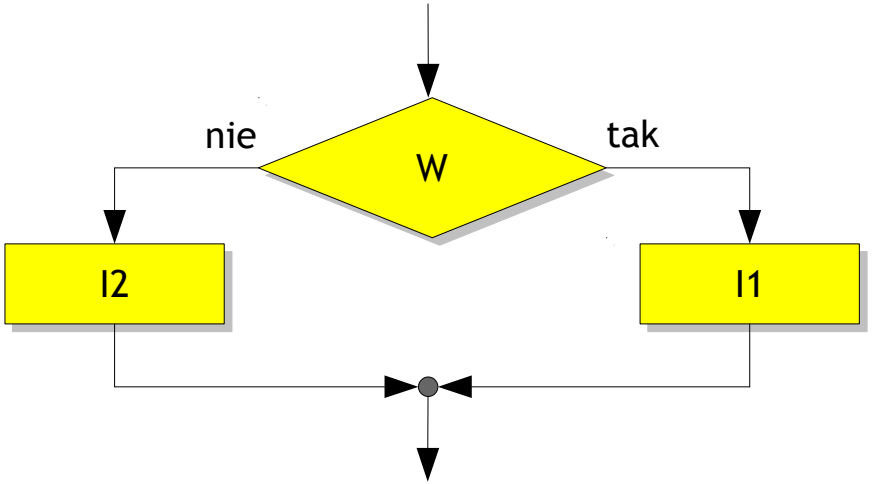
Instrukcja warunkowa

```
if( W )
  I
```



Instrukcja alternatywy

```
if( W )
  I1
else
  I2
```



W językach C/C++ każde wyrażenie o wartości numerycznej 0 jest traktowane jako fałszywe, każde wyrażenie o wartości różnej od zera jako prawdziwe.

Copyright © Roman Simiński

Suplement IV — instrukcje warunkowe a instrukcja blokowa

Instrukcja **`if`** oraz **`if-else`** obejmują swoim zasięgiem *jedną instrukcję* zapisaną dalej! Aby obejmowały *większą liczbę instrukcji*, trzeba je połączyć w instrukcję *grupującą*, tworzącą *blok instrukcji*.

Instrukcja grupująca zwana jest też *instrukcją złożoną*.

```
if( dystans != 0 )
  cout << "Srednie spalanie to " << ( paliwo*100 ) / dystans;
  cout << " l na 100 km" << endl;

cout << "Nacisnij Enter by zakonczyc program...";
. . .
```

```
if( dystans != 0 )
{
  cout << "Srednie spalanie to " << ( paliwo*100 ) / dystans;
  cout << " l na 100 km" << endl;
}
cout << "Nacisnij Enter by zakonczyc program...";
. . .
```

Copyright © Roman Simiński

Suplement IV — instrukcje warunkowe a instrukcja blokowa

- Zastosowanie wcięć w kodzie programów w języku C/C++ nie znaczenia dla kompilatora.

```
if( dystans != 0 )
  cout << "Srednie spalanie to " << ( paliwo*100 ) / dystans;
  cout << " l na 100 km" << endl;

cout << "Nacisnij Enter by zakonczyc program...";
. . .
```

Tylko ta instrukcja wykona się warunkowo

```
if( dystans != 0 )
  cout << "Srednie spalanie to " << ( paliwo*100 ) / dystans;
cout << " l na 100 km" << endl;

cout << "Nacisnij Enter by zakonczyc program...";
. . .
```

Copyright © Roman Simiński

Suplement IV — instrukcje warunkowe a instrukcja blokowa

- Przykład *instrukcji alternatywy* i *instrukcji grupującej*:

```
if( dystans == 0 )
{
  cout << "Wprowadzono zerowy dystans" << endl;
  cout << "Nie mozna wykonac obliczen" << endl;
}
else
{
  cout << "Srednie spalanie to " << ( paliwo*100 ) / dystans;
  cout << " l na 100 km" << endl;
}
```

Copyright © Roman Simiński

Podsumowanie

- Każdy program w języku C/C++ musi posiadać funkcję *main*. Od niej rozpoczyna się wykonanie programu.
- Funkcja *main* według standardu ANSI C powinna „zwracać” wartość całkowitą, stanowiącą kod zakończenia programu przekazywany systemowi operacyjnemu lub procesowi — rodzicowi.
- Każda funkcja posiada swoje ciało ujęte w nawiasy klamrowe { }.
- Zakończenie wykonania funkcji następuje wraz z osiągnięciem klamry zamykającej ciało funkcji lub po napotkaniu instrukcji powrotu z podprogramu — instrukcji *return*. Wartość umieszczona po tej instrukcji stanowi rezultat funkcji.
- Instrukcja *return* może wystąpić w ciele funkcji wielokrotnie, w miejscach dozwolonych syntaktyką języka.
- W języku C zmienne wewnątrz funkcji wolno deklarować na początku każdego bloku. Na etapie deklaracji zmienne mogą być inicjowane.

Copyright © Roman Simiński

Podsumowanie, cd...

- W języku C++ zmienne wewnątrz funkcji wolno deklarować w dowolnym sensowym miejscu. Na etapie deklaracji zmienne mogą być inicjowane.
- W języku C/C++ nie ma funkcji standardowych. Wszystkie funkcje pochodzą z bibliotek lub są napisane przez programistę.
- Aby kompilator mógł sprawdzić poprawność wywołania funkcji bibliotecznych wykorzystuje się *pliki nagłówkowe*. Zawierają one informacje o typie rezultatu funkcji, parametrach i są włączane do kodu źródłowego programu dyrektywą *#include*.
- W języku C wywołanie funkcji bez włączenia odpowiedniego pliku nagłówkowego nie spowoduje *błędu kompilacji* (ang. *error*) a jedynie *ostrzeżenie* (ang. *warning*). W języku C++ wywołanie funkcji bez włączenia odpowiedniego pliku nagłówkowego spowoduje powstanie *błędu kompilacji* (ang. *error*).
- W instrukcji *if-else*, przed słowem kluczowym *else* występuje średnik. W języku C/C++ średnik nie rozdziela instrukcji, a kończy je.

Copyright © Roman Simiński

Ćwiczenia

- **Ćwiczenie 1**. Napisać program wyliczający pole trójkąta — program wczytuje wysokość h, podstawę a (są to dowolne liczby rzeczywiste), oblicza pole (P=(a * h)/2) i wyświetla wynik. Zakładamy, że długości a i h są wyrażone w centymetrach, wynik ma być wyrażony w metrach kw.. Program powinien na samym początku wyświetlić krótką informację o jego przeznaczeniu oraz zatrzymać swoje wykonanie po wyświetleniu wyniku — do czasu naciśnięcia klawisza Enter.

- **Ćwiczenie 2**. Napisać program wyliczający pole koła oraz kwadratu na nim opisanego — program wczytuje promień r (to dowolna liczba rzeczywista), oblicza pole ($P= \pi * r^2$), długość boku a kwadratu opisanego na takim okręgu, oraz jego pole ($P=a^2$) i wyświetla te wyniki. Program powinien na samym początku wyświetlić krótką informację o jego przeznaczeniu oraz zatrzymać swoje wykonanie po wyświetleniu wyniku — do czasu naciśnięcia klawisza Enter.

Copyright © Roman Simiński

Ćwiczenia

- **Ćwiczenie 3**. Funkcja liniowa ma równanie $y=ax+b$. Napisać program wyliczający miejsce zerowe dla dowolnego równania liniowego $ax+b=0$ — program wczytuje współczynniki a i b (są to dowolne liczby rzeczywiste) i wyświetla wynik. Program powinien na samym początku wyświetlić krótką informację o jego przeznaczeniu oraz zatrzymać swoje wykonanie po wyświetleniu wyniku — do czasu naciśnięcia klawisza Enter.

- **Ćwiczenie 4**. Prędkość w ruchu jednostajnym prostoliniowym może być określona uproszczonym wzorem $v=s/t$, gdzie s to droga przebyta w czasie t. Napisać program wyliczający prędkość v — program wczytuje drogę s i czas jej przebycia t (są to dowolne liczby rzeczywiste) i wyświetla wynik. Program powinien na samym początku wyświetlić krótką informację o jego przeznaczeniu oraz zatrzymać swoje wykonanie po wyświetleniu wyniku — do czasu naciśnięcia klawisza Enter.

Copyright © Roman Simiński

Ćwiczenia

- **Ćwiczenie 5**. Cena brutto to cena netto powiększona o pewien podatek, wyrażony procentowo. Jeżeli coś kosztuje netto 100zł, a kwota podatku to 22%, cena brutto wynosi 122zł. Napisać program, który wyznaczy cenę brutto na podstawie ceny netto oraz podatku wyrażonego procentowo — program wczytuje cenę netto, podatek wyrażony procentowo (są to dowolne liczby rzeczywiste) i wyświetla wynik. Program powinien na samym początku wyświetlić krótką informację o jego przeznaczeniu oraz zatrzymać swoje wykonanie po wyświetleniu wyniku— do czasu naciśnięcia klawisza Enter.

- **Ćwiczenie 6**. Wynagrodzenie pewnego pracownika to liczba przepracowanych godzin przemnożona przez stawkę godzinową. Napisać program, który wyznaczy wynagrodzenie pracownika po wczytaniu liczby przepracowanych godzin oraz stawki (są to dowolne liczby rzeczywiste). Dodatkowo program ma wyznaczyć, ile pracownik zarobił na dniówkę, zakładając, że pracuje zawsze, równo 8 godzin. Program powinien na samym początku wyświetlić krótką informację o jego przeznaczeniu oraz zatrzymać swoje wykonanie po wyświetleniu wyników— do czasu naciśnięcia klawisza Enter.

Copyright © Roman Simiński

Ćwiczenia

- **Ćwiczenie 7**. Szybkostrzelność teoretyczna karabinka automatycznego AK (Automat Kałasznikowa) wynosi 600strzałów/minutę. Magazynek karabinka mieści 30 naboi. Napisać program, który wczyta wyrażony w sekundach czas (dowolna liczba całkowita) trwania ognia ciągłego, prowadzonego z takiego karabinka. Ćwiczeniem programu jest wyznaczyć liczbę magazynków, które trzeba by wymienić, aby strzelać ogniem ciągłym przez wprowadzony czas. Program powinien na samym początku wyświetlić krótką informację o jego przeznaczeniu oraz zatrzymać swoje wykonanie po wyświetleniu wyniku— do czasu naciśnięcia klawisza Enter.

- **Ćwiczenie 8**. Sportowiec w trakcie jednego treningu spala średnio 1500 kalorii. Napisać program, który wczyta: ile razy sportowiec trenuje w tygodniu, i ile planuje tygodni trenować (dowolne liczby całkowite). Na tej podstawie program ma wyliczyć ile kilokalorii sportowiec spali w tym okresie czasu. Program powinien na samym początku wyświetlić krótką informację o jego przeznaczeniu oraz zatrzymać swoje wykonanie po wyświetleniu wyniku— do czasu naciśnięcia klawisza Enter.

Copyright © Roman Simiński