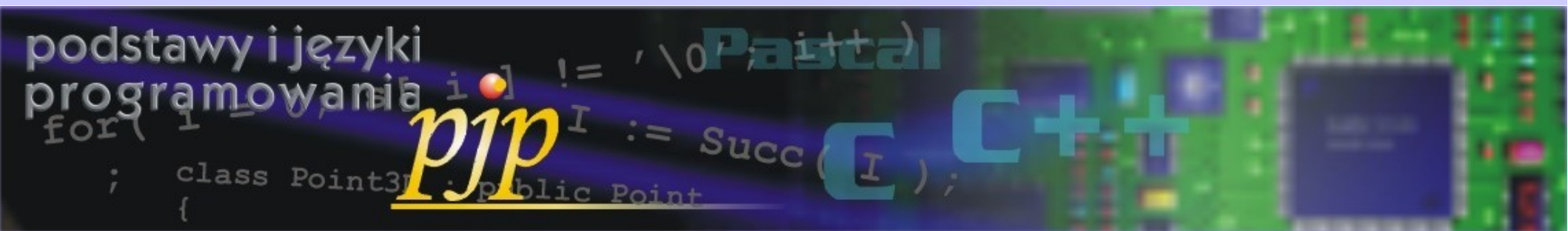
Języki programowania obiektowego

Nieobiektowe elementy języka C++

Roman Simiński
roman.siminski@us.edu.pl
www.programowanie.siminskionline.pl

Struktura programu, funkcja *main*, pliki nagłówkowe, operacje wejścia-wyjścia



Struktura programu w języku C/C++

- Program może składać się z wielu osobnych plików (*modułów*) kompilowanych oddzielnie, ale tworzenie *programów wielomodułowych* zostanie omówione później.
- Program *jednomodułowy* jest *zbiorem funkcji*.
- Język C++ jest językiem *hybrydowym,* pozwala na programowanie *proceduralne*, *obiektowe* oraz na mieszane stosowanie *obu tych podejść*.
- Ponieważ C++ jest językiem hybrydowym, funkcje mogą być funkcjami *składowymi klas* (programowanie obiektowe) jak i zwykłymi funkcjami (programowanie proceduralne).
- Niezależnie od przyjętego podejścia do programowania, każdy program w standardzie C/C++ musi zawierać funkcję **main**.
- Od funkcji **main** rozpoczyna się wykonanie programu. Funkcja ta nie jest związana z żadną klasą i nie ma nic wspólnego z programowaniem obiektowym.

Przykład programu

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

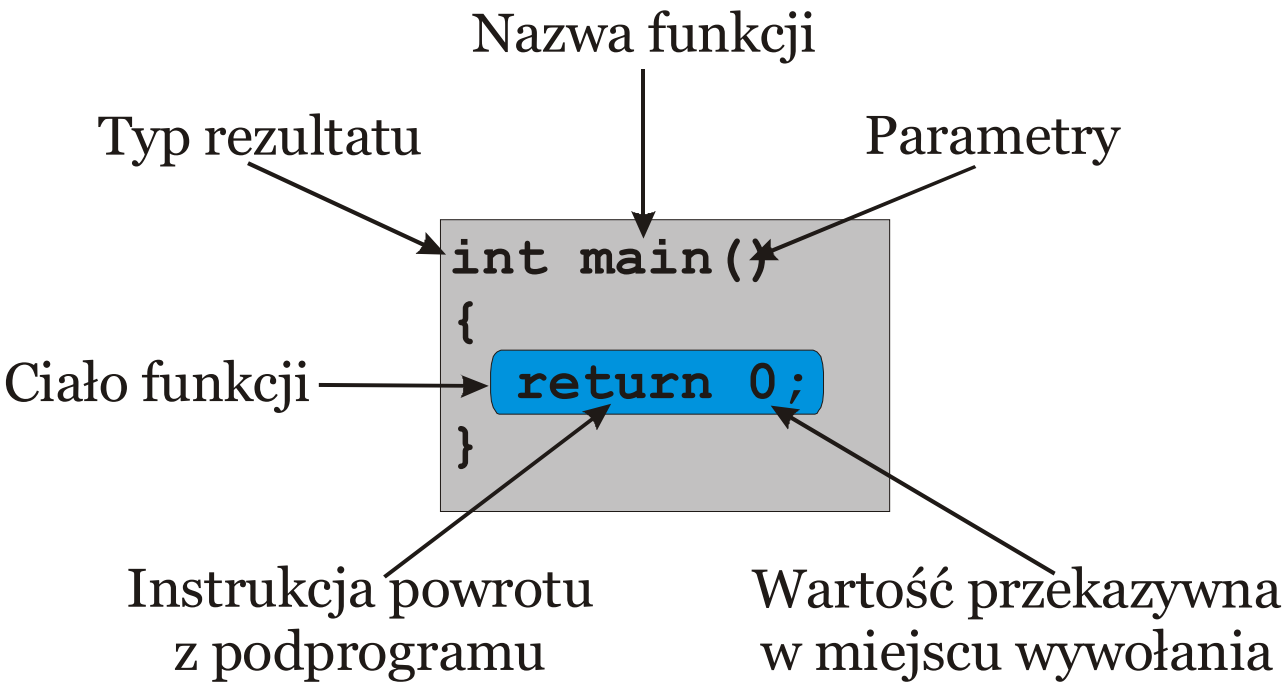
```
Przeliczanie odleglosci wyrazonej w kilometrach na mile
Podaj odleglosc w kilometrach: 100
To w milach: 62.50
Nacisnij Enter by zakonczyc program...
```

Odległość w milach = 0.625 · odległość w kilometrach

Odległość w kilometrach = 1.6 · odległość w milach

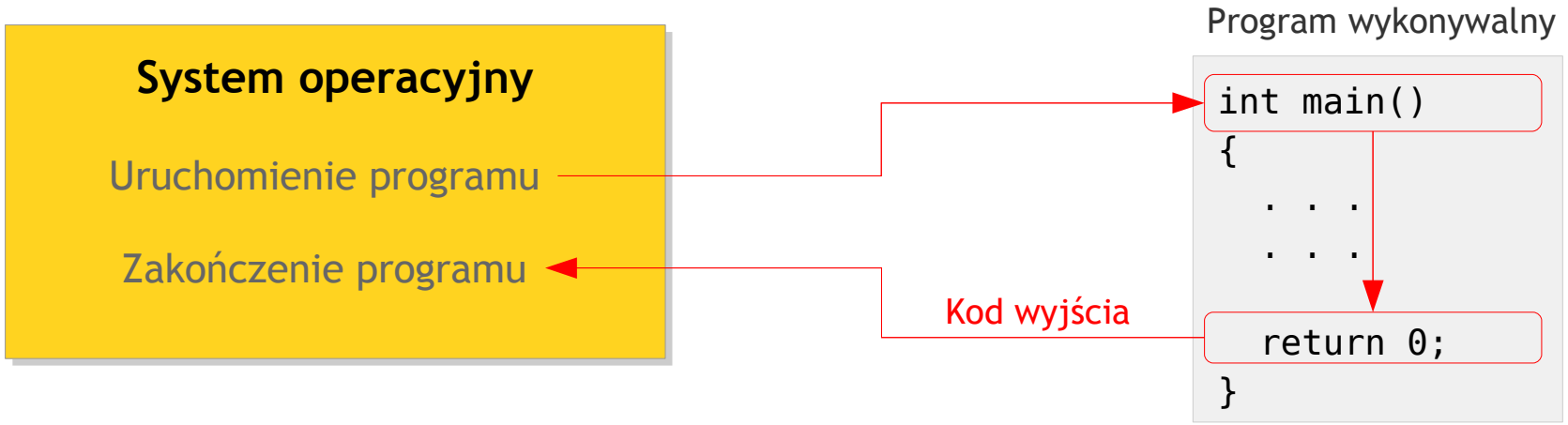
Pierwsza przymiarka — program, który robi nic

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



Kto wywołuje funkcję main?

- 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 — np. dla procesów potomnych).
- 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*.



Funkcja main — Java, C/C++, C#

```
class ProgramClass
{
  public static void main( String[] args )
  {
  }
}
```

Java

Informacje o parametrach wywołania programu

```
class ProgramClass
{
  static void Main( string[] args )
  {
  }
}
```

C#

```
int main( int argc, char args[] )
{
  return 0;
}
```

C/C++

W C/C++ jeżeli nie interesują nas parametry wywołania programu, piszemy **main()**

Funkcja main — Java, C/C++, C#, ustalanie kodu wyjścia

Java

```
class ProgramClass
{
  public static void main( String[] args )
  {
    System.exit( 0 );
  }
}
```

C#

```
class ProgramClass
{
  static int Main( string[] args )
  {
    return 0;
  }
}
```

C/C++

```
int main( int argc, char args[] )
{
  return 0;
}
```

Zakończenie funkcji main z wartością 0, przekazywaną „wywoływaczowi”.

Rezultat funkcji main wg normy POSIX

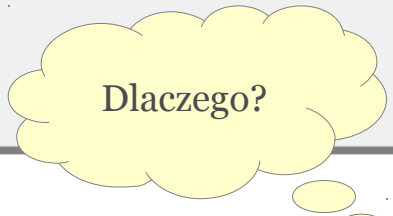
Zamiast rezultatu w postaci bezwzględnych wartości można wykorzystać symbole:

- **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

Pliki nagłówkowe

Symbole:

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

nie są częścią języków C/C++, zapisane są w bibliotekach języka C.

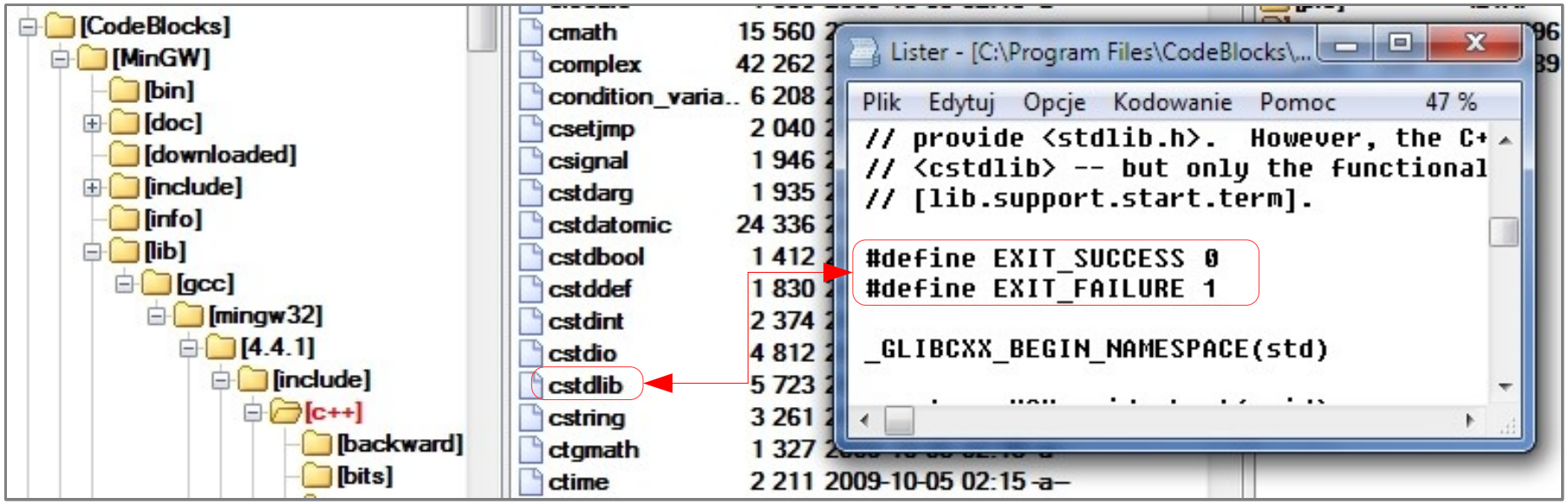
Aby z nich skorzystać, należy uzupełnić program do postaci:

```
#include <cstdlib>

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

Dyrektywa #include powoduje włączenie zawartości pliku *stdlib*, zawierającego definicje symboli *EXIT_SUCCESS* i *EXIT_FAILURE*

Pliki nagłówkowe

- Dyrektywa **#include** powoduje włączenie do kodu źródłowego definicji i deklaracji niezbędnych dla kompilacji.
- Informacje te są przechowywane w *plikach nagłówkowych*, są to pliki tekstowe. Znaleźć je można zwykle w katalogu *include* danego kompilatora.
- W pliku nagłówkowym *cstdlib* zdefiniowane są właśnie symbole **EXIT_SUCCES** oraz **EXIT_FAILURE**.



Dyrektywa #include

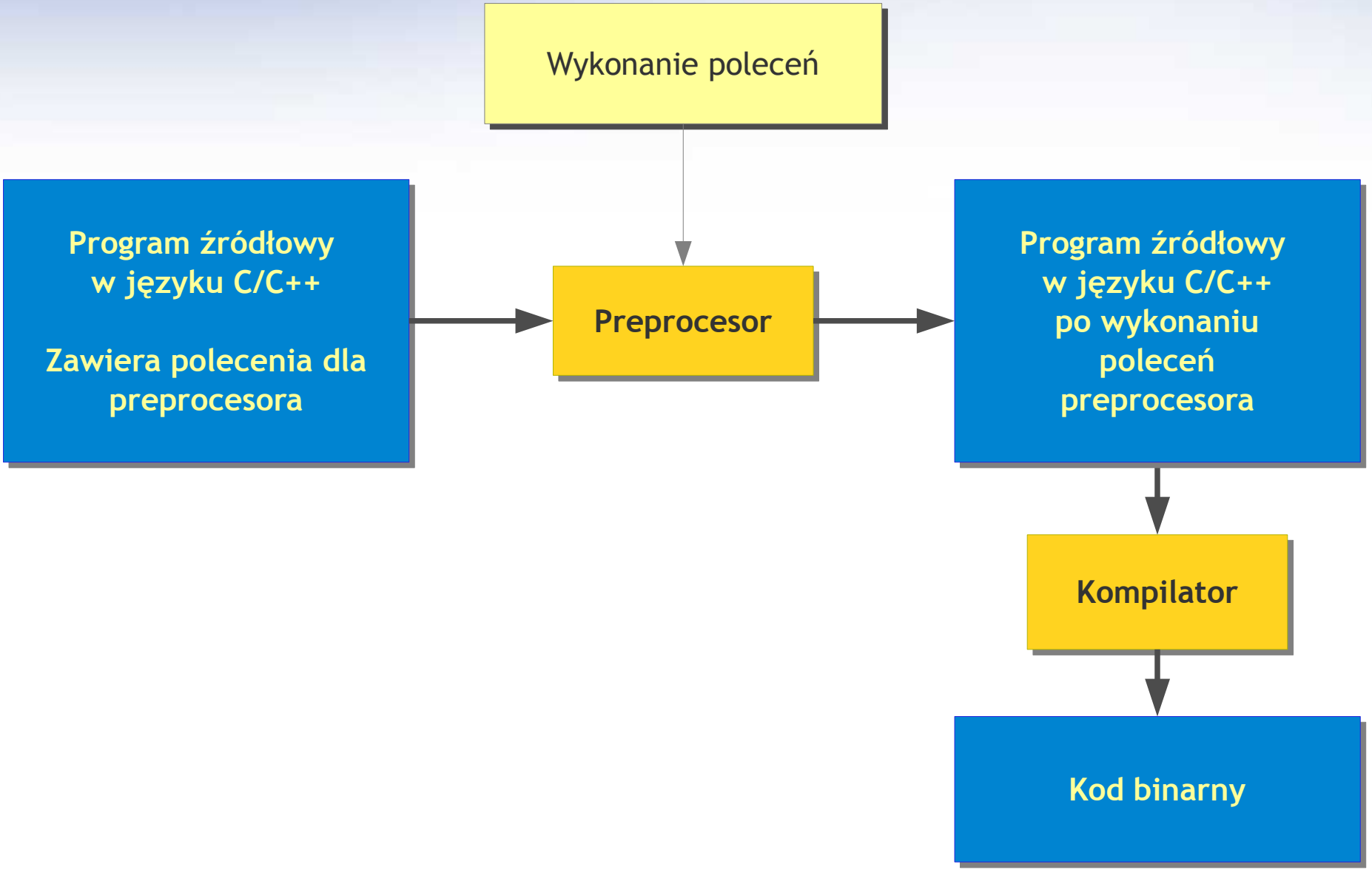
Dyrektywa:

```
#include <cstdlib>
```

powoduje włączenie do kodu programu zawartości pliku nagłówkowego *cstdlib*, zawierającego definicje i deklaracje niezbędne dla wykorzystania elementów *standardowej biblioteki* języka C.

- Dyrektywa **#include** realizowana jest przez *preprocesor* języka C/C++.
- ***Preprocesor*** to zwykle osobny program, realizujący wstępne przetwarzanie kodu źródłowego programu w języku C (C++, Objective-C), realizowane przed przekazaniem programu na wejście kompilatora.
- Zadaniem preprocesora jest wyszukanie w tekście źródłowym programu przeznaczonych dla niego poleceń, oraz ich wykonanie.
- Wykonanie polecenia oznacza zwykle operację na tekście źródłowym — zamianę jednego tekstu na inny, włączenie zawartości jakiegoś innego pliku, pominięcie pewnego fragmentu tekstu itp.

Preprocesor — koncepcja



Preprocesor — działanie

```
#include <cstdlib>

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

Dyrektywa #include powoduje włączenie zawartości całego pliku c*stdlib*, w tym definicje symboli *EXIT_SUCCESS* i *EXIT_FAILURE*

```
. . .
#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1
. . .

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

Dyrektywa #define to także polecenie preprocesora, jej znaczenie omówione zostanie później.

Kłopot z rozszerzeniami plików nagłówkowych

- Pliki nagłówkowe w języku C mają rozszerzenie .h.
- Pliki nagłówkowe w języku C++ mają zwykle rozszerzenie .h, .hpp, .hxx, .h++.
- Wielość możliwych rozszerzeń nazw plików w C++ to kłopot.
- Aktualnie w programach w języku C++ nie pisze się rozszerzeń nazw plików nagłówkowych (standard z r. 2003).
- W programach w języku C++, nazwy plików nagłówkowych z języka C pisze się z prefiksem c.

C , ANSI89

Nagłówek standardowej biblioteki

```
#include <stdlib.h>
```

Nagłówek biblioteki wejścia/wyjścia

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

C++

Nagłówek standardowej biblioteki z C

```
#include <cstdlib>
```

Nagłówek biblioteki wejścia/wyjścia z C

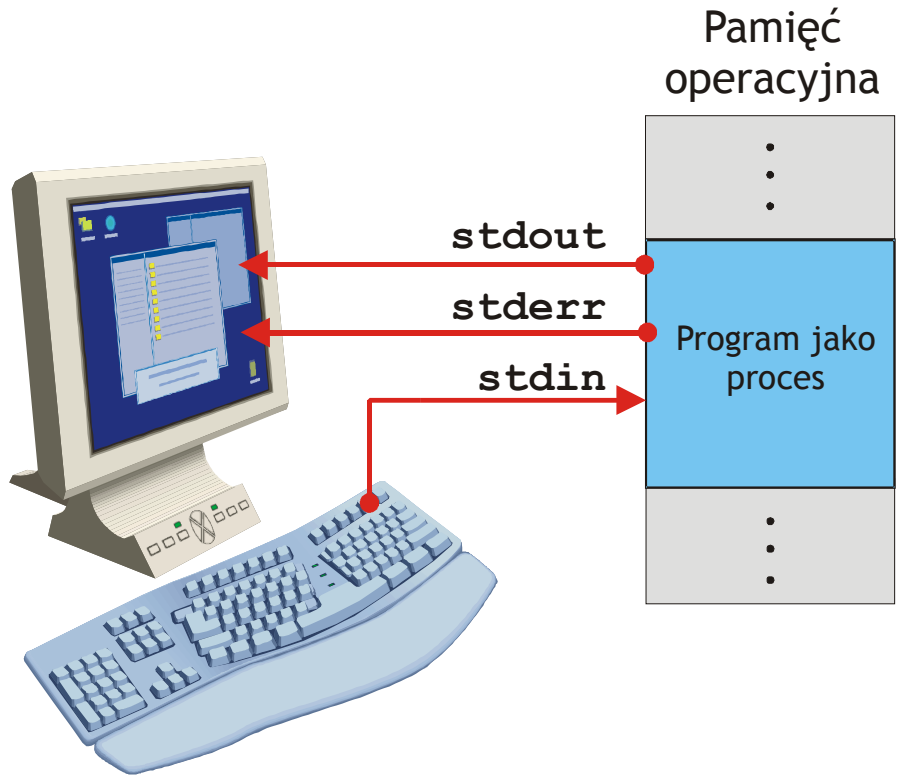
```
#include <cstdio>
```

Standardowe strumienie programu w języku C

- *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.



Standardowe strumienie programu w języku C

Redyrekcja (przekierowanie) strumieni z poziomu systemu operacyjnego:

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

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

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

Standardowe strumienie programu w języku C++

W języku C++ rolę strumieni *We-Wy* przejmują predefiniowane obiekty klas *istream* oraz *ostream*.

Aby skorzystać z obiektów *cin* i *cout* należy włączyć plik nagłówkowy *iostream*:

```
#include <iostream>
```

- *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.

Wracamy do programu przeliczania odległości...

```
#include <cstdlib>
#include <iostream>

using namespace std;

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

  return EXIT_SUCCESS;
}
```

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

Określenie domyślnej przestrzeni nazw jest wygodne

```
#include <cstdlib>
#include <iostream>

using namespace std;

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

  return EXIT_SUCCESS;
}
```

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

Wyprowadzanie komunikatów do strumienia wyjściowego

```
#include <cstdlib>
#include <iostream>

using namespace std;

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

  return EXIT_SUCCESS;
}
```

- **`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.

Specjalne sekwencje znakowe

- W językach C i C++ można używać specjalnych sekwencji znakowych.
- Dodatkowo sekwencje specjalne są wykorzystywane do zapisu pewnych „niewygodnych” stałych znakowych.

Sekwencja	Wartość	Znak	Znaczenie
\a	0x07	BEL	Audible bell
\b	0x08	BS	Backspace
\f	0x0C	FF	Formfeed
\n	0x0A	LF	Newline (linefeed)
\r	0x0D	CR	Carriage return
\t	0x09	HT	Tab (horizontal)
\v	0x0B	VT	Vertical tab
\\	0x5c	\	Backslash
\'	0x27	'	Apostrof
\"	0x22	"	Cudzysłów
\?	0x3F	?	Pytajnik
\O		any	O = łańcuch ósemkowych cyfr
\xH		any	H = łańcuch szesnastkowych cyfr
\XH		any	H = łańcuch szesnastkowych cyfr

Specjalne sekwencje znakowe

- Zamiast manipulatora **endl** można używać znaku \n.
- Manipulator **endl** powoduje opróżnienie bufora strumienia wyjściowego i przejście do nowej linii.
- Sekwencja **\n** powoduje tylko przejście do nowej linii.
- Sekwencje sterujące mogą występować jako literały znakowe lub być częścią literałów łańcuchowych.

Literały znakowe (ograniczone znakiem apostrofa):

```
'a'    '*'    '\n'       '\t'       '\a'       '\\'       '\''       '\"'
```

Literały łańcuchowe (ograniczone znakiem cudzysłowa):

```
"Jezyk C++"    "c:\\system\\trash"    "\nRaz\nDwa\nTrzy"    "\"Cudzyslow\""
```

Do reprezentowania znaków w językach C/C++ służy typ **char.**

Gdyby nie było domyślnej przestrzeni nazw std...

```
#include <cstdlib>
#include <iostream>

using namespace std;

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

  return EXIT_SUCCESS;
}
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

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

  return EXIT_SUCCESS;
}
```

Pełna nazwa obiektu **cout** to **std::cout**, gdzie **std** to nazwa przestrzeni w jakiej zdefiniowano obiekt **cout**, **::** to operator zakresu.

Wprowadzamy zmienne

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  float kilometry, wynik;

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

  return EXIT_SUCCESS;
}
```

- *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.

Zapytaj o odległość w kilometrach

```
#include <cstdlib>
#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

Wczytanie odległości w kilometrach, obiekt cin

```
#include <cstdlib>
#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;
}
```

- `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.

Wczytanie odległości w kilometrach, obiekt cin

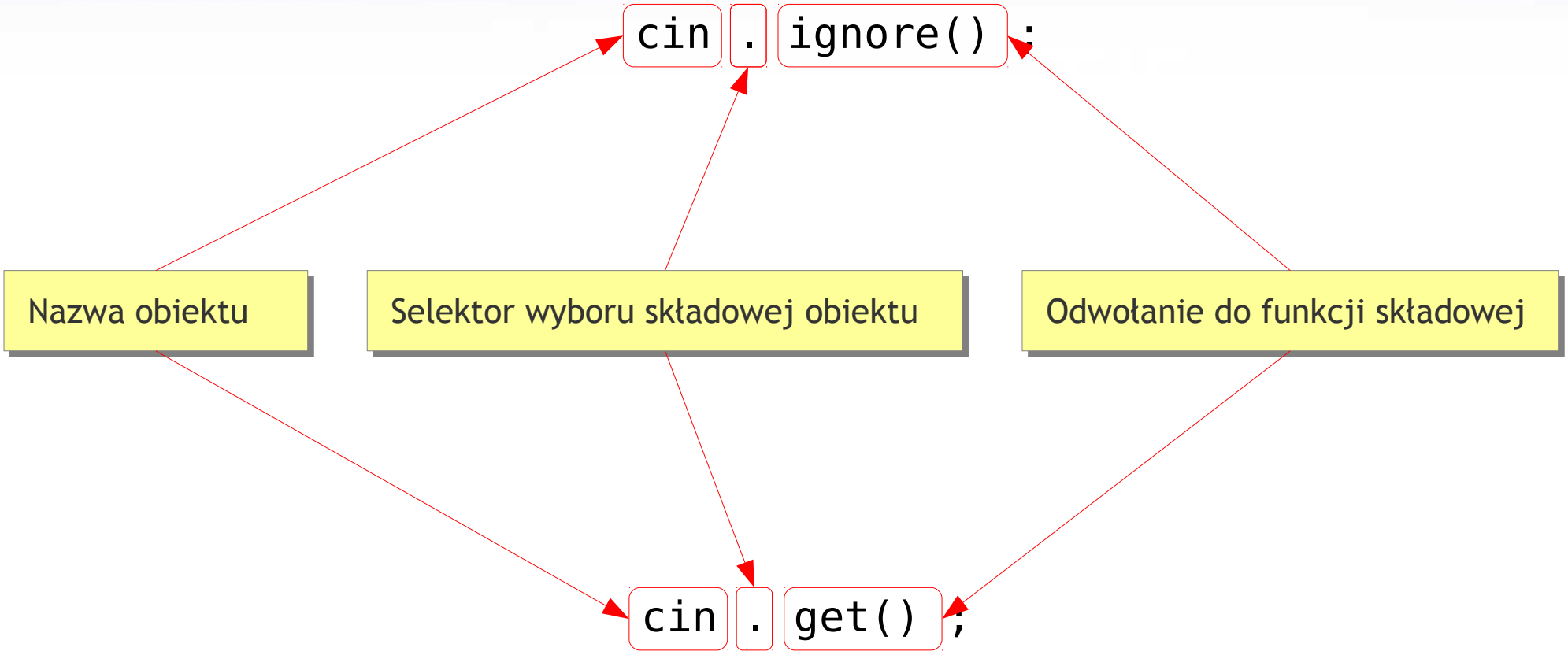
```
#include <cstdlib>
#include <iostream>

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 wartość

Wyprowadź wyniki

Niestety, wyniku zwykle nie zobaczymy, bo okno konsoli zostanie zamknięte... .

Obiekt cin posiadający wbudowane funkcje składowe



Niech użytkownik ma szansę zobaczyć wyniki

```
#include <cstdlib>
#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;
}
```

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

Uzupełnienie — zmienna wynik jest właściwie zbędna...

```
. . .
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();
. . .
```

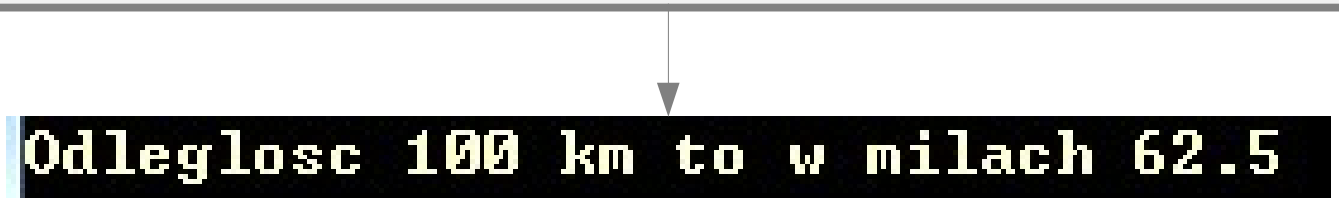
Uzupełnienie — wyprowadzanie do cout można uprościć

```
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
```

Podsumowanie

- Każdy program w języku C/C++ musi posiadać funkcję *main*. Od niej rozpoczyna się wykonanie programu.
- Rezultatem funkcji *main* według standardu jest wartość całkowita, stanowiąca kod zakończenia programu przekazywany systemowi operacyjnemu lub procesowi — rodzicowi.
- 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.
- W języku C++ zmienne można definiować na w każdym sensownym miejscu.

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.

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.

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.

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.

Ćwiczenia

- **Ćwiczenie 9**. Samochody z silnikiem wysokoprężnym (diesel) są zwykle droższe od tych, z silnikiem benzynowym. Jednak spalają średnio mniej paliwa. Niestety od jakiegoś czasu olej napędowy jest droższy od benzyny. Należy napisać program, który wyliczy po ilu latach różnica w cenie zakupu auta z silnikiem diesla zwróci się z powodu mniejszego spalania. Program wczytuje:
  - cenę auta z silnikiem benzynowym i jego średnie spalanie na 100 km,
  - cenę auta z silnikiem wysokoprężnym i jego średnie spalanie na 100 km,
  - cenę benzyny,
  - cenę oleju napędowego,
  - szacowany roczny przebieg w kilometrach.

Dziękuję za uwagę

Pytania? Polemiki?
Teraz, albo:
roman.siminski@us.edu.pl