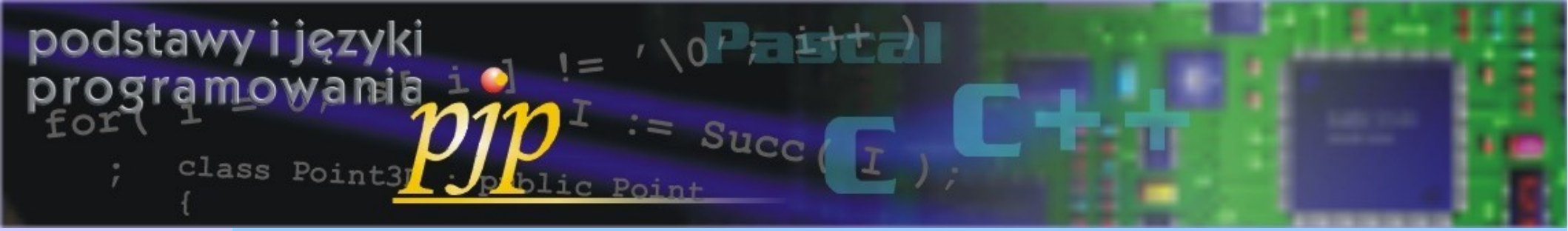


Podstawy programowania w języku C++

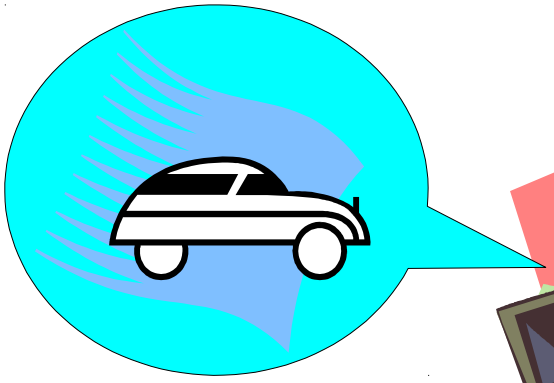
Część dziesiąta

Rekordy w C/C++ — struktury

Autor
Roman Simiński

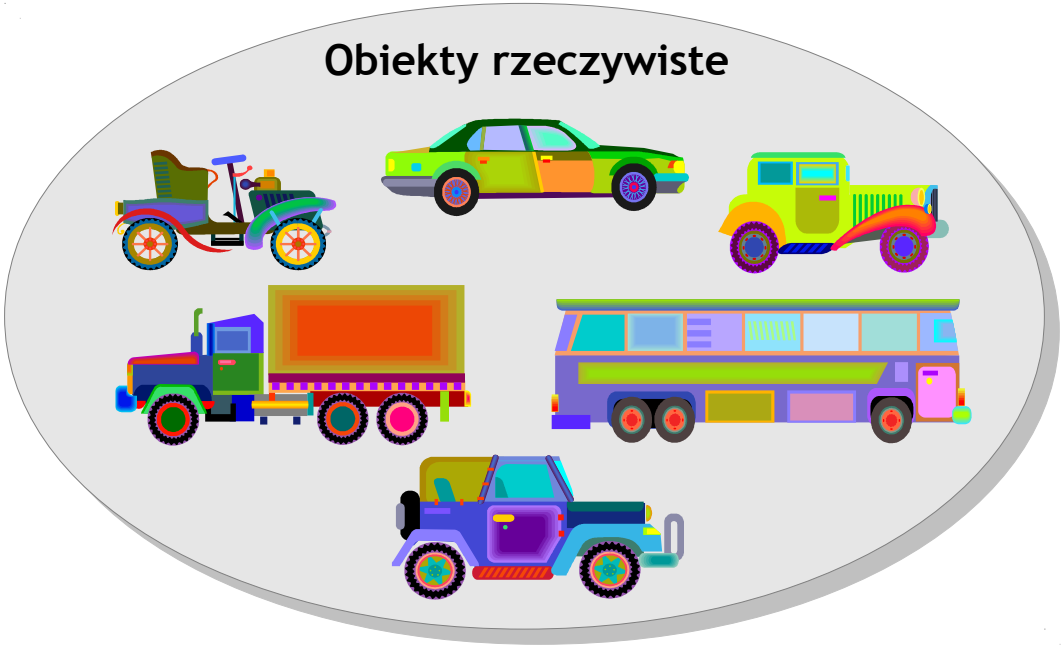
Kontakt
roman.siminski@us.edu.pl
www.programowanie.siminskionline.pl

Programowanie jako tworzenie komputerowego modelu rzeczywistości



Abstrakcyjny model analityczny

Analityk i/lub projektant systemu informatycznego



Obiekty, elementy, pojęcia ze świata zewnętrznego muszą zostać odwzorowane danymi w programie. Dane występujące w programie stanowią uproszczony, komputerowy model rzeczywistości.

Obliczanie średniego spalania raz jeszcze ;-)

Zmienna
`Dystans`

Zmienna
`Paliwo`

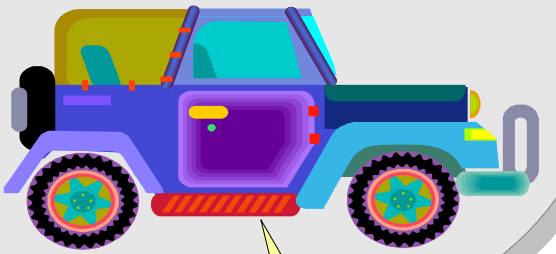
Dane opisujące komputerowy model problemu

Algorytm

Wylicz średnie spalanie:
`(Paliwo*100)/Dystans`
Wyświetl wynik

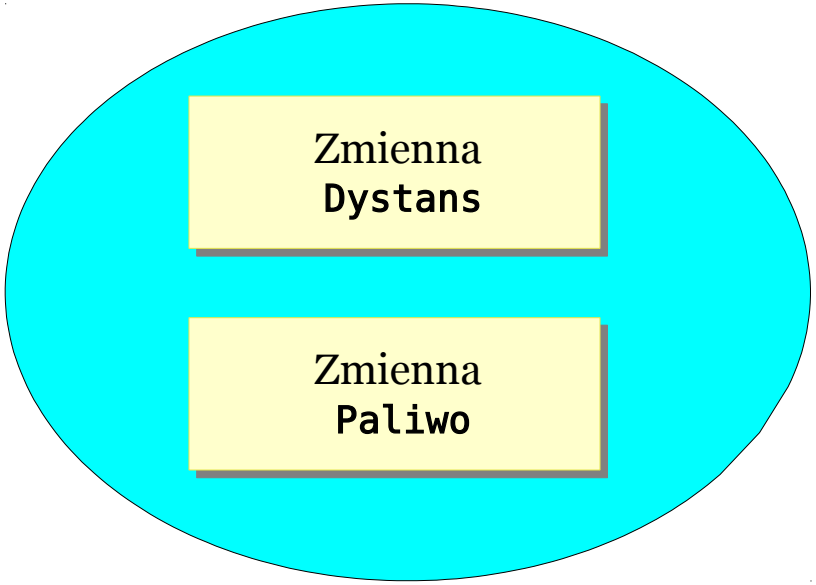
Analityk i/lub projektant systemu informatycznego

Dane rzeczywistego obiektu



Przejechany dystans: 500km
Zużyte paliwo: 37l

Obliczanie średniego spalania raz jeszcze ;-)



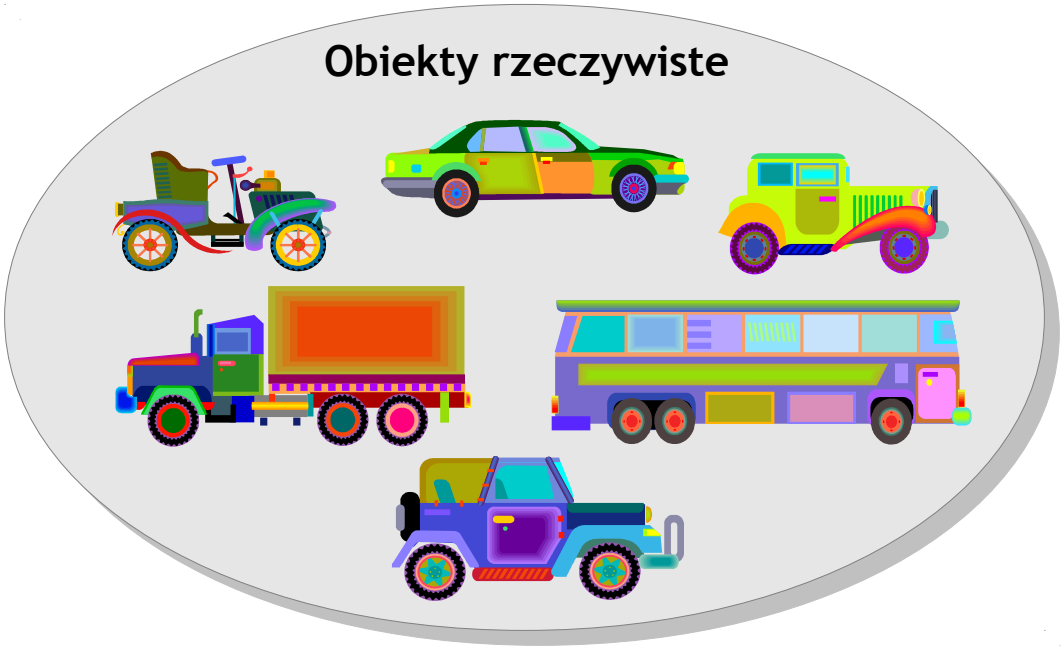
Dane modelu są dwiema, osobnymi zmiennymi liczbowymi

Analityk i/lub projektant systemu informatycznego

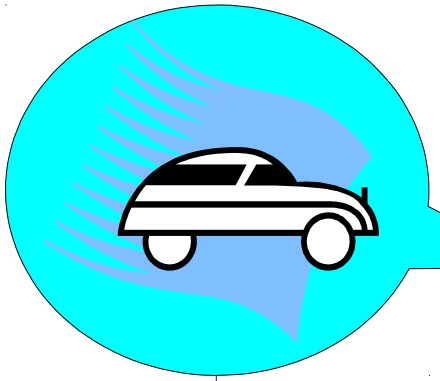
Nowe zadanie — system ewidencji pojazdów dla autokomisu

Jakich danych potrzebujemy?

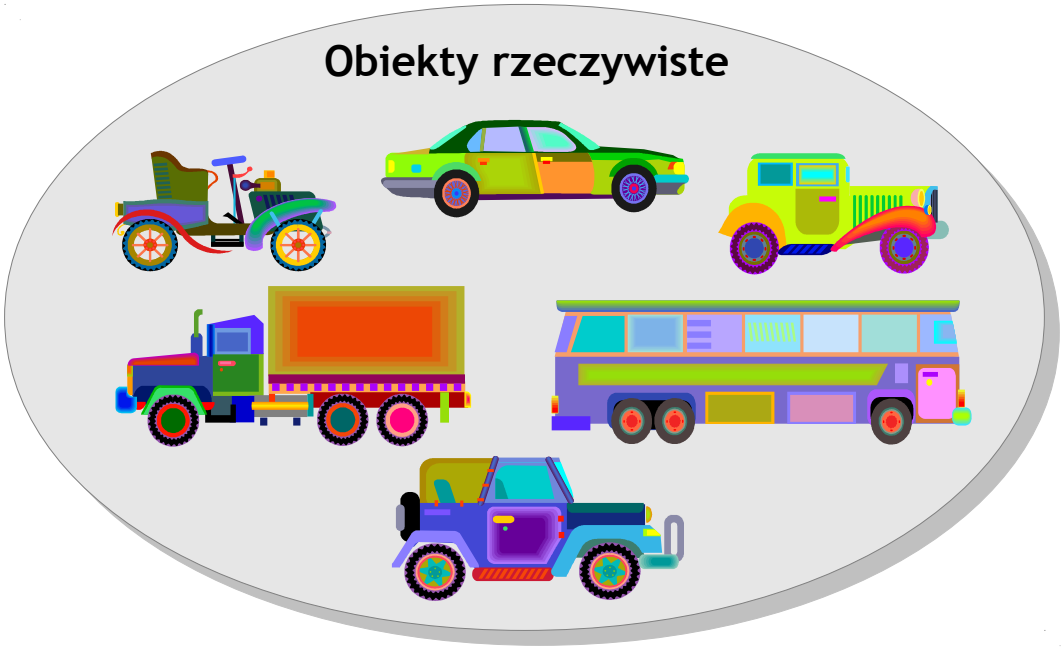
Analityk i/lub projektant systemu informatycznego



Jakie informacje będziemy przetwarzać i przechowywać?



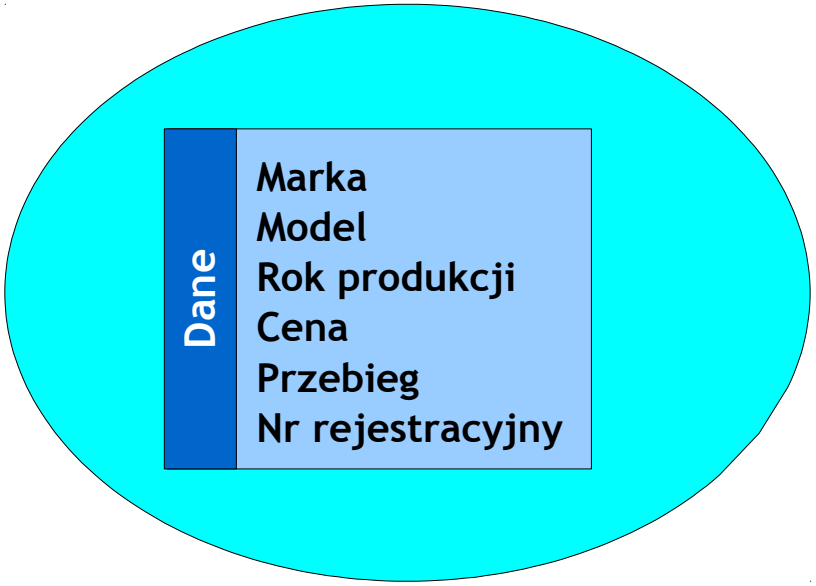
Analityk i/lub
projektant systemu
informatycznego



Dane

- Marka
- Model
- Rok produkcji
- Cena
- Przebieg
- Nr rejestracyjny

Dane opisują jeden pojazd



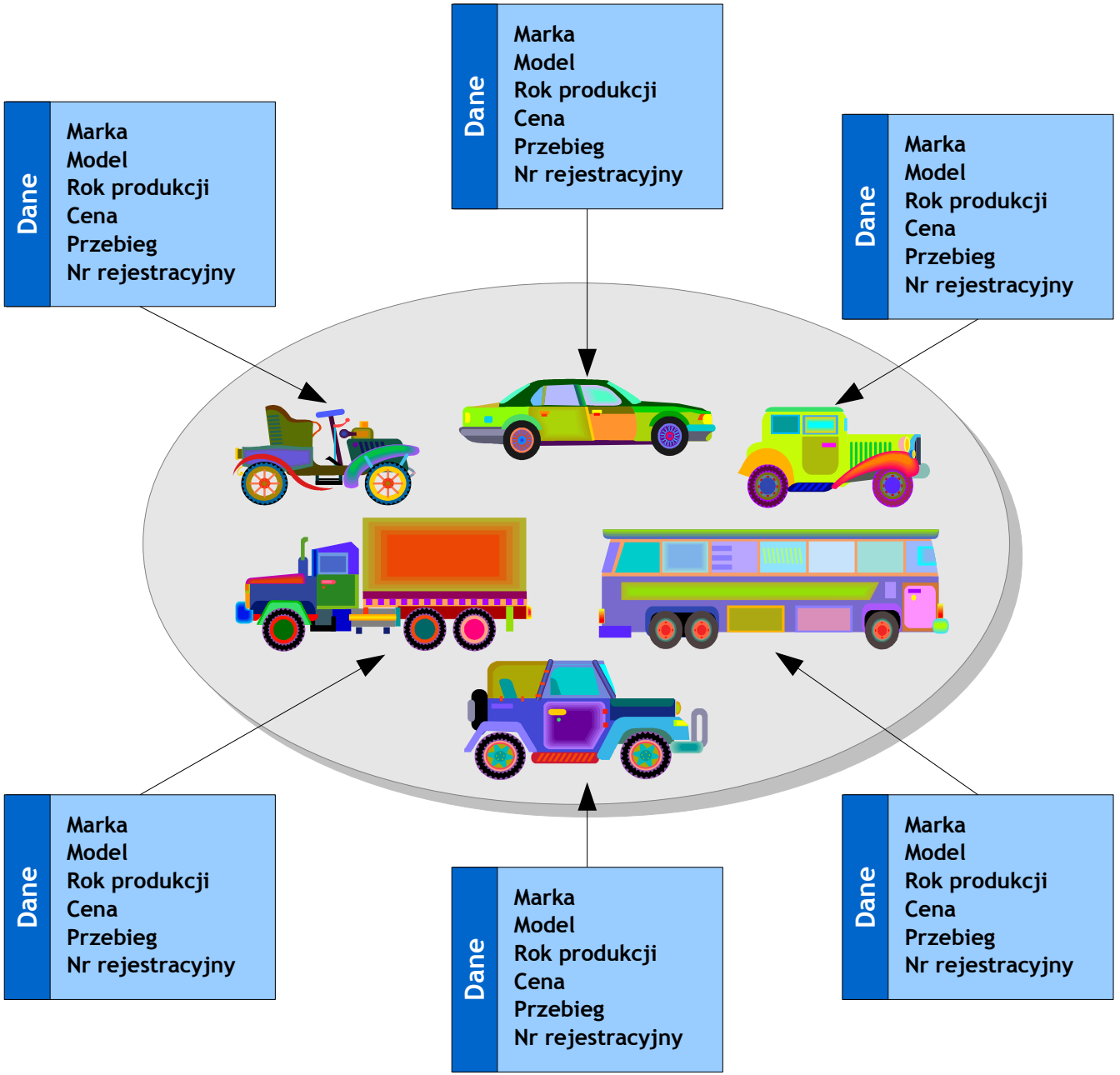
Dane opisujące jeden pojazd to porcja różnych informacji

Analityk i/lub projektant systemu informatycznego

Pojazdów jest wiele...

Potrzeba wiele porcji danych.

Każda z porcji jest złożona i zawiera różne dane opisujące pojazd.



Struktury — zmienne do przechowywania różnych danych

Definicja typu strukturalnego

```
struct pojazd
{
  char  marka[ 20 ];
  char  model[ 20 ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ 10 ];
};

. . .

pojazd a;
```

Pola struktury o nazwie **pojazd**

Dane

- **Marka**
- **Model**
- **Rok produkcji**
- **Cena**
- **Przebieg**
- **Nr rejestracyjny**

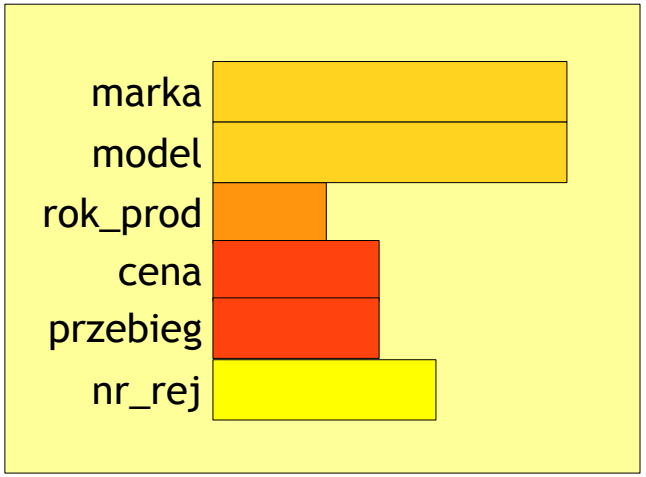
Deklaracja zmiennej strukturalnej o nazwie **a**

Struktury — parametryzacja rozmiarów tablic

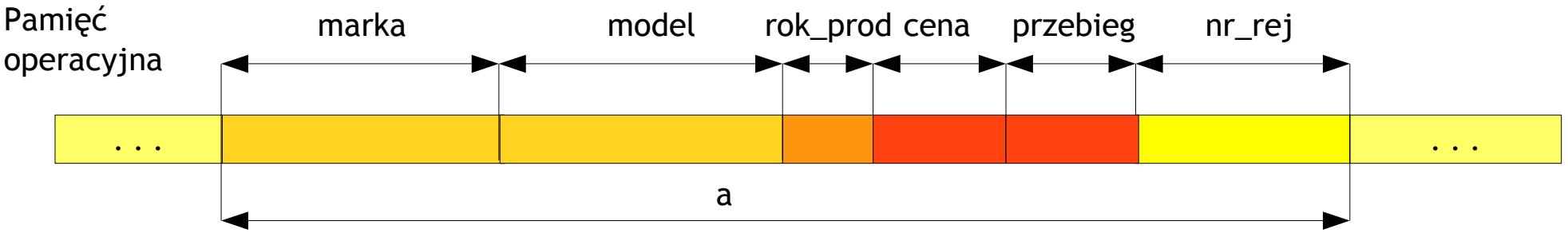
```
const int MAKS_M = 20;
const int MAKS_R = 10;
```

```
struct pojazd
{
  char  marka[ MAKS_M ];
  char  model[ MAKS_M ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ MAKS_R ];
};

. . .

pojazd a;
```

Struktura a jako rekord



Struktury — reprezentacja w pamięci

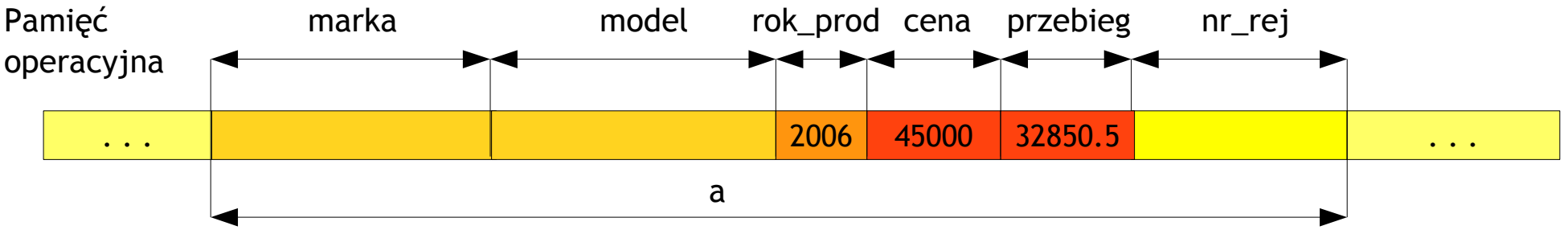



```
struct pojazd
{
  char  marka[ MAKS_M ];
  char  model[ MAKS_M ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ MAKS_R ];
};

. . .

pojazd a;
```

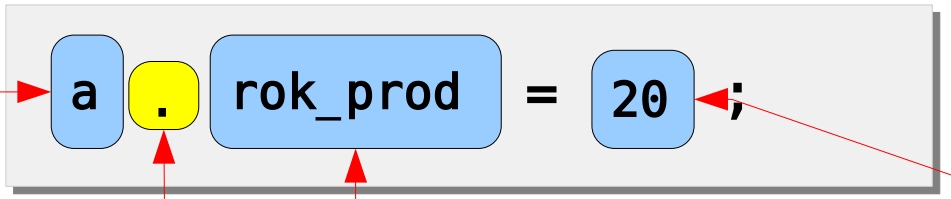
Struktury — odwoływanie się do pól

```
struct pojazd
{
  char  marka[ MAKS_M ];
  char  model[ MAKS_M ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ MAKS_R ];
};

pojazd a;

. . .

a.rok_prod = 2006;
a.przebieg = 32850.5;
a.cena = 45000;
```

Wstawianie wartości do pól zmiennej strukturalnej **a**



Struktury — odwoływanie się do pól, format zapisu



Nazwa zamiennej strukturalnej lub wyrażenie lokalizujące taką zmienną w pamięci operacyjnej.

Operator selekcji pola struktury

Nazwa pola struktury

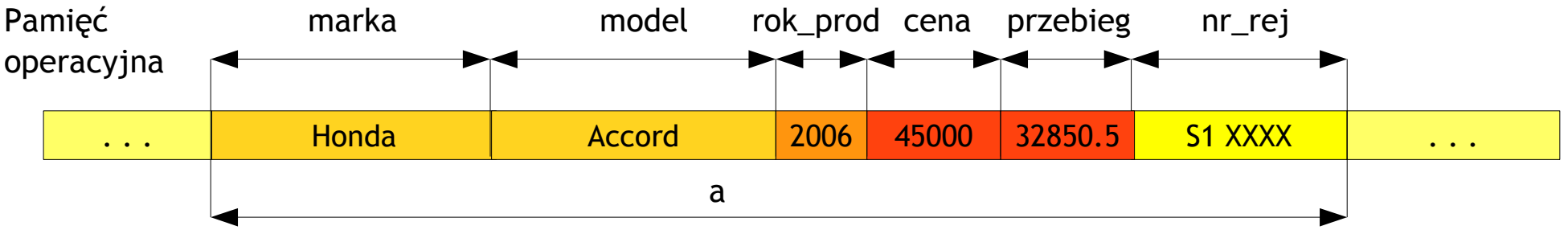
Dowolne wyrażenie typu zgodnego z typem pola struktury.

Struktury — odwoływanie się do pól tablicowych

```
struct pojazd
{
  char  marka[ MAKS_M ];
  char  model[ MAKS_M ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ MAKS_R ];
};

pojazd a;

. . .
```

```
strcpy( a.marka, ”Honda” );
strcpy( a.marka, ”Accord” );
strcpy( a.nr_rej, ”S1 XXXX” );
```

Wstawianie wartości do pól zmiennej strukturalnej **a** będących tablicami znaków



Wyprowadzanie zawartości pól struktury do strumienia wyjściowego

```
struct pojazd
{
  char  marka[ MAKS_M ];
  char  model[ MAKS_M ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ MAKS_R ];
};

pojazd a;

. . .

cout << "\nMarka: " << a.marka;
cout << "\nModel: " << a.model;
cout << "\nRok produkcji: " << a.rok_prod;
cout << "\nCena: " << a.cena;
cout << "\nPrzebieg: " << a.przebieg;
cout << "\nNr rejestracyjny: " << a.nr_rej;
```

Wprowadzanie danych do struktury ze strumienia wejściowego

```
pojazd a;

cout << "\nPodaj dane pojazdu";

cout << "\nMarka: ";
cin >> a.marka;

cout << "Model: ";
cin >> a.model;

cout << "Rok produkcji: ";
cin >> a.rok_prod;

cout << "Cena: ";
cin >> a.cena;

cout << "Przebieg: ";
cin >> a.przebieg;

cout << "Numer rejestracyjny: ";
cin >> a.nr_rej;
```

Uwaga! Ta wersja wprowadzania danych do rekordu jest podatna na błędy przepełnienia bufora

Nazwa struktury — różnice w C89 i C++

W języku C++ nazwa oznacznikowa struktury występująca po słowie *struct* jest pełnoprawną nazwą typu strukturalnego.

```
struct pojazd
{
  char  marka[ MAKS_M ];
  char  model[ MAKS_M ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ MAKS_R ];
};

pojazd a;
```

Nazwa struktury — różnice w C89 i C++

W języku C nazwa występująca po słowie kluczowym *struct* nie jest samodzielną nazwą typu strukturalnego. W deklaracja zmiennych należy użyć słowa kluczowego *struct*.

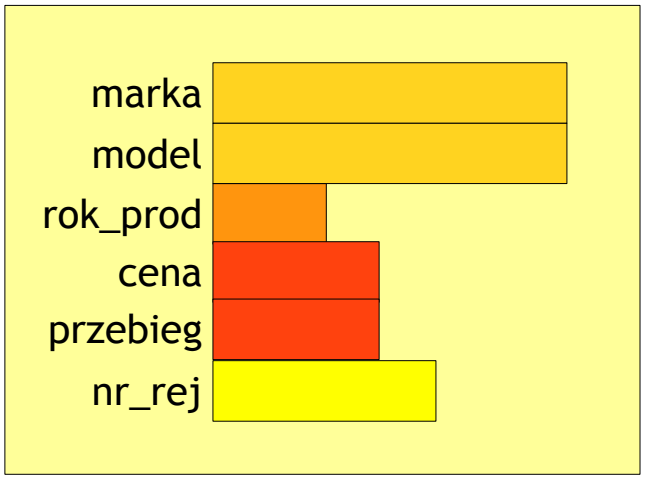
```
struct pojazd
{
  char  marka[ MAKS_M ];
  char  model[ MAKS_M ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ MAKS_R ];
};

struct pojazd a;
```

Struktury — parametryzacja rozmiarów tablic w C

```
#define MAKS_M 20
#define MAKS_R 10
```

```
struct pojazd
{
  char  marka[ MAKS_M ];
  char  model[ MAKS_M ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ MAKS_R ];
};

. . .

struct pojazd a;
```

Struktura a jako rekord



Nazwa struktury — różnice w C89 i C++

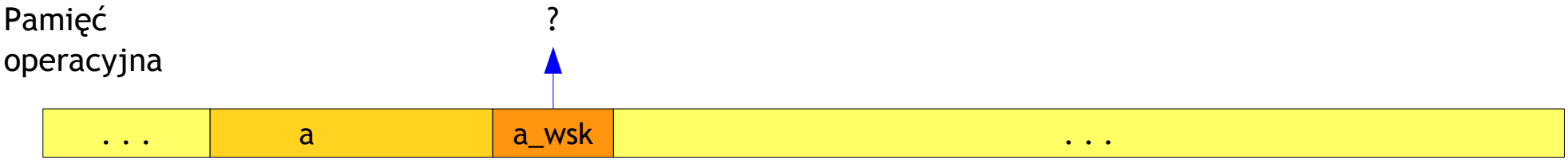
Aby nie pisać słowa kluczowego *struct*, można użyć deklaracji tworzącej synonimiczną nazwę typu: *typedef*.

```
struct _pojazd
{
  char  marka[ MAKS_M ];
  char  model[ MAKS_M ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ MAKS_R ];
};

typedef struct _pojazd pojazd;

pojazd a;
```

```
typedef struct
{
  char  marka[ MAKS_M ];
  char  model[ MAKS_M ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ MAKS_R ];
} pojazd;



pojazd a;
```

Manipulowanie strukturami przy użyciu wskaźników

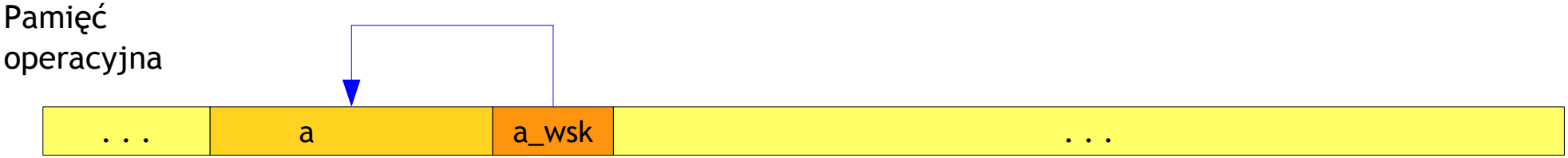
```
struct pojazd
{
  char  marka[ MAKS_M ];
  char  model[ MAKS_M ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ MAKS_R ];
};

pojazd a;       // Deklaracja zmiennej strukturalnej pojazd
pojazd * a_wsk; // Deklaracja zmiennej wskaźnikowej do pojazd
. . .
```



Manipulowanie strukturami przy użyciu wskaźników, cd ...

```
struct pojazd
{
  char  marka[ MAKS_M ];
  char  model[ MAKS_M ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ MAKS_R ];
};

pojazd a;       // Deklaracja zmiennej strukturalnej pojazd
pojazd * a_wsk; // Deklaracja zmiennej wskaźnikowej do pojazd
. . .
a_wsk = &a;
```

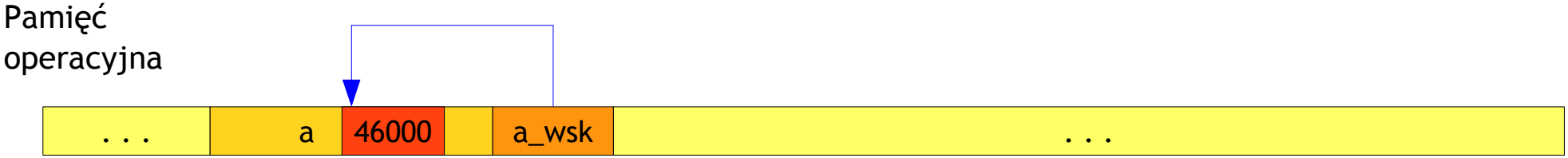
Zmienna **a_wsk** lokalizuje a pamięci zmienną **a**



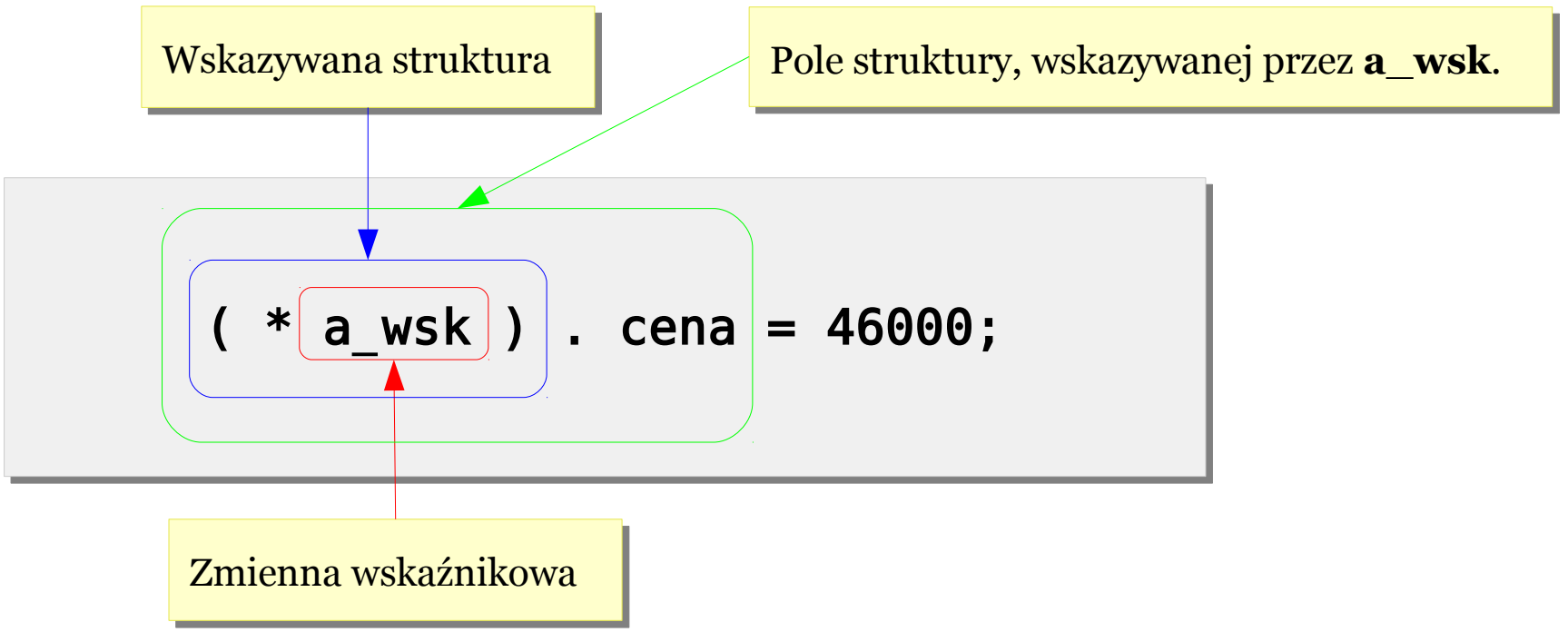
Manipulowanie strukturami przy użyciu wskaźników, cd ...

```
struct auto
{
  char  marka[ MAKS_M ];
  char  model[ MAKS_M ];
  int   rok_prod;
  float cena;
  float przebieg;
  char  nr_rej[ MAKS_R ];
};

pojazd a;       // Deklaracja zmiennej strukturalnej pojazd
pojazd * a_wsk; // Deklaracja zmiennej wskaźnikowej do pojazd
. . .
a_wsk = &a;
(*a_wsk).cena = 46000;
```

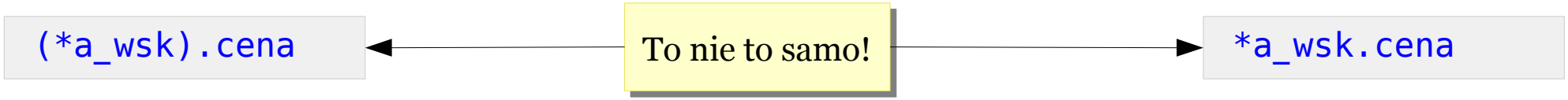
Wyrażenie ***a_wsk** reprezentuje strukturę **a**



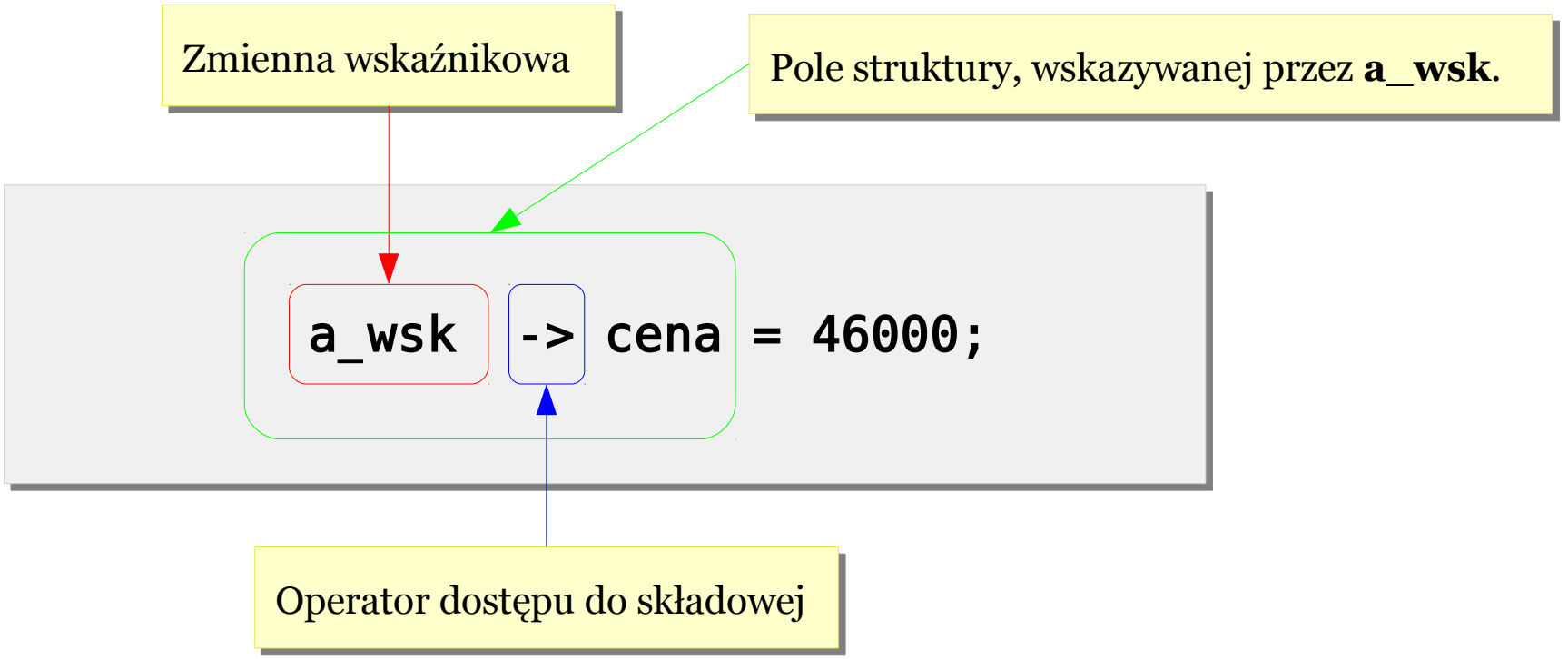
Odwoływanie się do pól struktury via wskaźnik



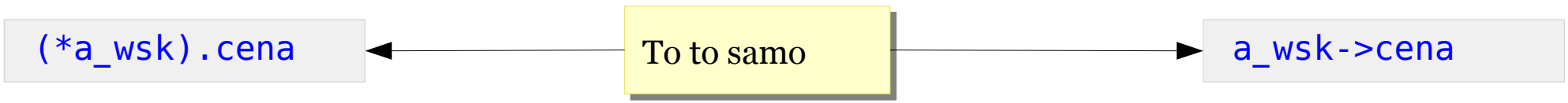
Uwaga! Ze względu na priorytet i łączność operatorów, nawiasy w powyższym wyrażeniu są niezbędne.



Odwoływanie się do pól struktury via wskaźnik, operator ->



Operator dostępu do składowej -> stosujemy dla struktur, uni i obiektów.

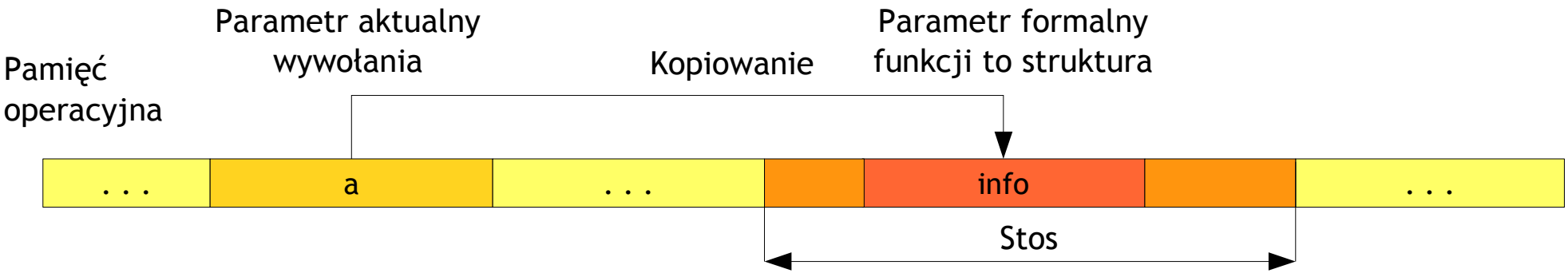


Funkcja wyprowadzająca zawartość struktury do stdout

```
void pokaz_info( pojazd info )
{
  cout << "\nMarka: " << info.marka;
  cout << "\nModel: " << info.model;
  cout << "\nRok produkcji: " << info.rok_prod;
  cout << "\nCena: " << info.cena;
  cout << "\nPrzebieg: " << info.przebieg;
  cout << "\nNr rejestracyjny: " << info.nr_rej;
}

. . .

pojazd a;
a.cena = 25000;
. . .

pokaz_info( a );
```

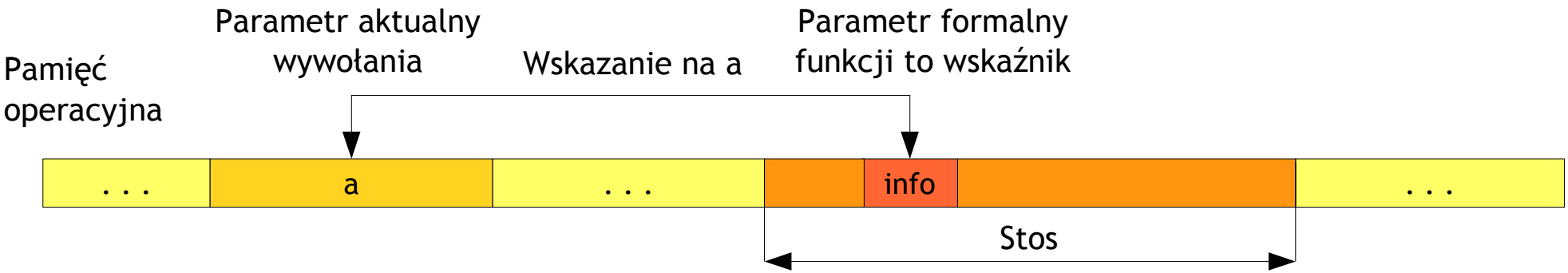
Przekazywanie struktur przez wartość

```
void pokaz_info( pojazd info )
{
  cout << ”\nMarka: ” << info.marka;
  cout << ”\nModel: ” << info.model;
  cout << ”\nRok produkcji: ” << info.rok_prod;
  cout << ”\nCena: ” << info.cena;
  cout << ”\nPrzebieg: ” << info.przebieg;
  cout << ”\nNr rejestracyjny: ” << info.nr_rej;;
}

. . .

pojazd a;
a.cena = 25000;
. . .

pokaz_info( a );
```



Przekazywanie wskaźnika do struktury

```
void pokaz_info( pojazd * info )
{
  cout << "\nMarka: " << info->marka;
  cout << "\nModel: " << info->model;
  cout << "\nRok produkcji: " << info->rok_prod;
  cout << "\nCena: " << info->cena;
  cout << "\nPrzebieg: " << info->przebieg;
  cout << "\nNr rejestracyjny: " << info->nr_rej;
}

. . .

pojazd a;
a.cena = 25000;
. . .

pokaz_info( &a );
```



Przekazywanie przez wartość a przekazywanie wskaźnika do struktury

Zmienne strukturalne mają często duży rozmiar. Przekazywanie ich przez wartość:

- *zabiera dodatkową pamięć* — parametr formalny (alokowany na stosie) jest pełnowymiarową kopią parametru formalnego,
- *trwa* — robienie kopii *parametru aktualnego* wywołania wymaga przesyłu pamięć-pamięć, potencjalnie dużej liczby bajtów,
- *blokuje modyfikacje* — modyfikacje wykonane na parametrze formalnym funkcji nie przenoszą się na parametr wywołania funkcji.

Przekazywanie przez wartość a przekazywanie wskaźnika do struktury

Przekazywanie wskaźników do struktury:

- *oszczędza pamięć* — parametr formalny (alokowany na stosie) zawiera jedynie adres parametru formalnego,
- *jest szybkie* — przekazanie wskaźnika to przesył małej liczby bajtów,
- *pozwala na modyfikacje* — modyfikacje wykonane na parametrze formalnym funkcji przenoszą się na parametr wywołania funkcji. Jeżeli modyfikacje obiektu wskazywanego mają być zabronione, używamy słowa kluczowego *const* w deklaracji parametru.

```
void pokaz_info( const pojazd * info )
{
  cout << "\nMarka: " << info->marka;
  cout << "\nModel: " << info->model;
  cout << "\nRok produkcji: " << info->rok_prod;
  cout << "\nCena: " << info->cena;
  cout << "\nPrzebieg: " << info->przebieg;
  cout << "\nNr rejestracyjny: " << info->nr_rej;
}
```

Przekazywanie referencji do struktur

W języku C++ można przekazywać parametry referencyjne. Nie trzeba wtedy używać wskaźników, a działanie jest analogiczne. Referencja *ustalona* (*const*) nie pozwala na niezamierzoną modyfikację parametru aktualnego wywołania.

```
void pokaz_info( pojazd & info )
{
  cout << "\nMarka: " << info.marka;
  cout << "\nModel: " << info.model;
  cout << "\nRok produkcji: " << info.rok_prod;
  cout << "\nCena: " << info.cena;
  cout << "\nPrzebieg: " << info.przebieg;
  cout << "\nNr rejestracyjny: " << info.nr_rej;
}
```

```
void pokaz_info( const pojazd & info )
{
  cout << "\nMarka: " << info.marka;
  cout << "\nModel: " << info.model;
  cout << "\nRok produkcji: " << info.rok_prod;
  cout << "\nCena: " << info.cena;
  cout << "\nPrzebieg: " << info.przebieg;
  cout << "\nNr rejestracyjny: " << info.nr_rej;
}
```

Funkcja wczytująca zawartość struktury ze strumienia wejściowego

```
void czytaj_info( pojazd * info )
{
 cout << "\nMarka: ";
 cin >> info->marka;

 cout << "Model: ";
 cin >> info->model;

 cout << "Rok produkcji: ";
 cin >> info->rok_prod;

 cout << "Cena: ";
 cin >> info->cena;

 cout << "Przebieg: ";
 cin >> info->przebieg;

 cout << "Numer rejestracyjny: ";
 cin >> info->nr_rej;
}
```

Wersja wprowadzania danych do rekordu jest podatna na błędy przepełnienia bufora

Suplement I — operacje wejścia/wyjścia w konwencji języka C

```
void pokaz_info( const pojazd * info )
{
  printf( ”\nMarka: %s”, info->marka );
  printf( ”\nModel: %s”, info->model );
  printf( ”\nRok produkcji: %d”, info->rok_prod );
  printf( ”\nCena: %g”, info->cena );
  printf( ”\nPrzebieg: %g”, info->przebieg );
  printf( ”\nNr rejestracyjny: %s”, info->nr_rej );
}
```

```
void czytaj_info( pojazd * info )
{
  char bufor[ 128 ];

  printf( "\nMarka: " ); gets( bufor );
  if( strlen( bufor ) >= MAKS_M ) bufor[ MAKS_M - 1 ] = '\0';
  strcpy( info->marka, bufor );

  printf( "Model: " );  gets( bufor );
  if( strlen( bufor ) >= MAKS_M ) bufor[ MAKS_M - 1 ] = '\0';
  strcpy( info->model, bufor );

  printf( "Rok produkcji: " ); gets( bufor );
  info->rok_prod = atoi( bufor );

  printf( "Cena: " ); gets( bufor );
  info->cena = atof( bufor );

  printf( "Przebieg: " ); gets( bufor );
  info->przebieg = atof( bufor );

  printf( "Numer rejestracyjny: " ); gets( bufor );
  if( strlen( bufor ) >= MAKS_R ) bufor[ MAKS_R - 1 ] = '\0';
  strcpy( info->nr_rej, bufor );
}
```