Podstawy Programowania

Wykład II

Algorytm, podstawowe notacje, typy danych, wyrażenia i instrukcje

Robert Muszyński
Katedra Cybernetyki i Robotyki, PWr

Zagadnienia: pojęcie algorytmu, przejście od algorytmu do programu, zapis składni programu, typy danych, stałe, zmienne, operatory, wyrażenia, drzewa wyliczania wartości wyrażeń, instrukcje, instrukcja złożona, instrukcja warunkowa i pętli

– Skład FoilTEX –

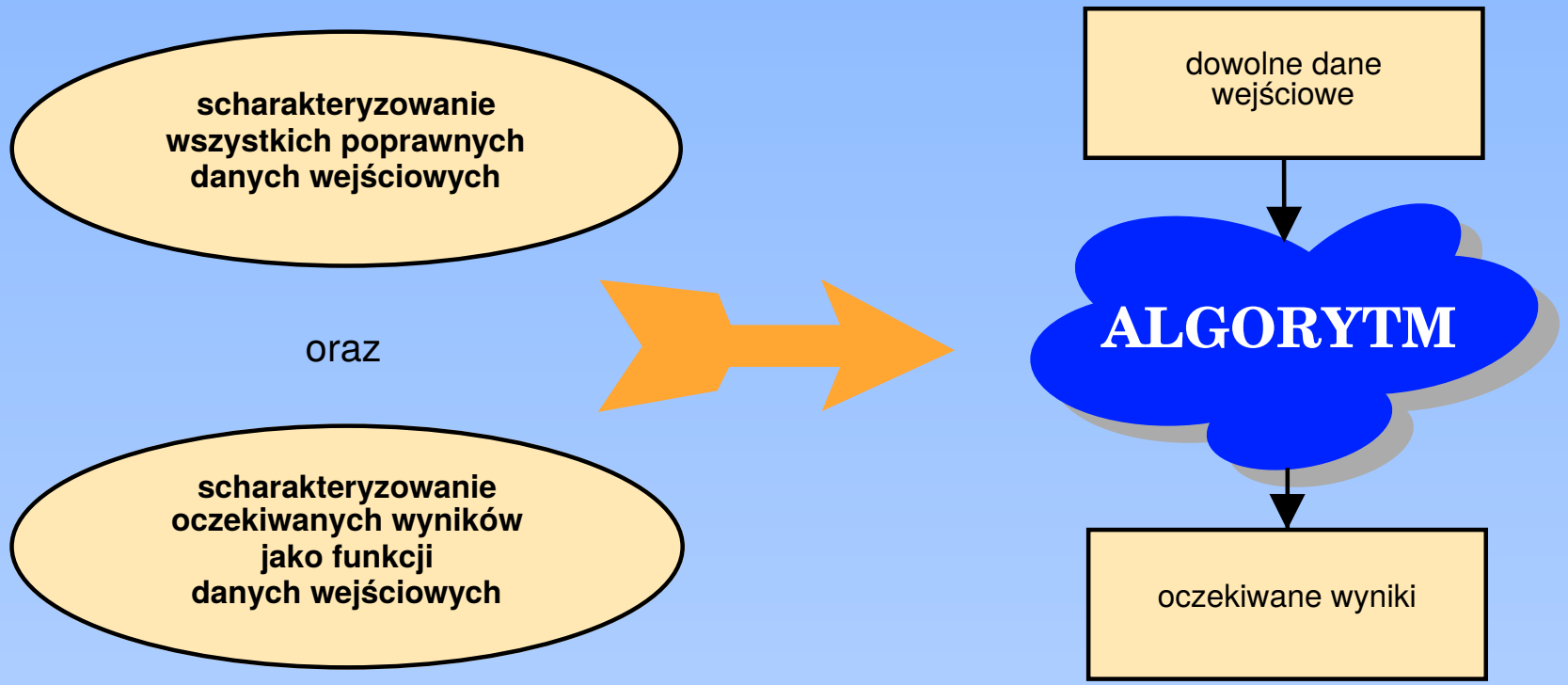
Algorytm

scharakteryzowanie wszystkich poprawnych danych wejściowych

oraz

scharakteryzowanie oczekiwanych wyników jako funkcji danych wejściowych

Algorytm


scharakteryzowanie
wszystkich poprawnych
danych wejściowych
oraz
scharakteryzowanie
oczekiwanych wyników
jako funkcji
danych wejściowych
dowolne dane
wejściowe
ALGORYTM
oczekiwane wyniki

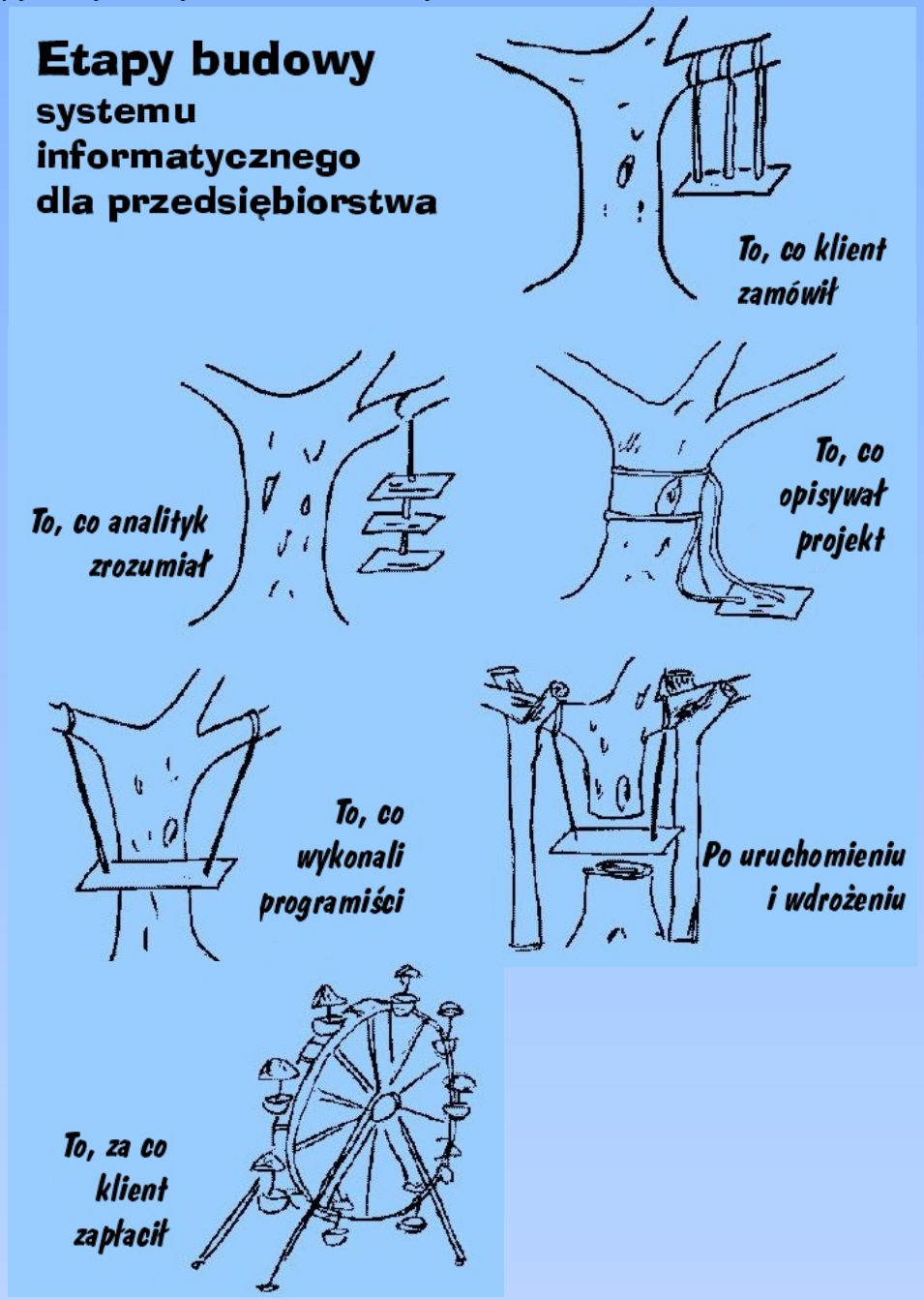
Etapy budowy
systemu
informatycznego
dla przedsiębiorstwa

Etapy budowy
systemu
informatycznego
dla przedsiębiorstwa


Etapy budowy
systemu
informatycznego
dla przedsiębiorstwa
To, co klient zamówił
To, co analityk zrozumiał
To, co opisywał projekt


Etapy budowy
systemu
informatycznego
dla przedsiębiorstwa
To, co klient zamówił
To, co analityk zrozumiał
To, co opisywał projekt
To, co wykonali programiści

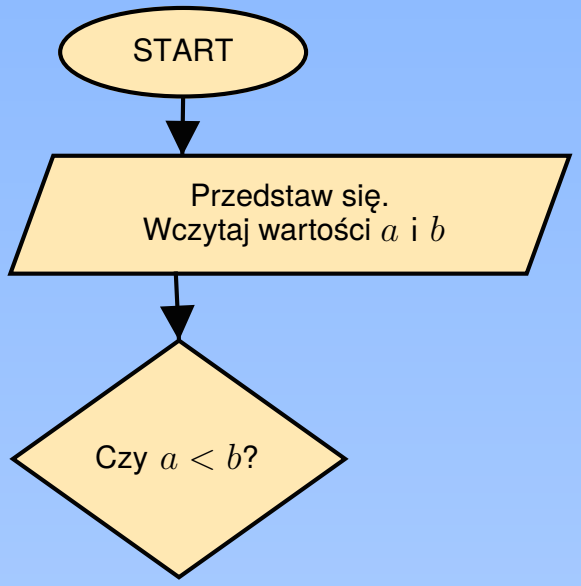

Etapy budowy
systemu
informatycznego
dla przedsiębiorstwa
To, co klient zamówił
To, co analityk zrozumiał
To, co opisywał projekt
To, co wykonali programiści
Po uruchomieniu i wdrożeniu


Etapy budowy
systemu
informatycznego
dla przedsiębiorstwa
To, co klient zamówił
To, co analityk zrozumiał
To, co opisywał projekt
To, co wykonali programiści
Po uruchomieniu i wdrożeniu
To, za co klient zapłacił

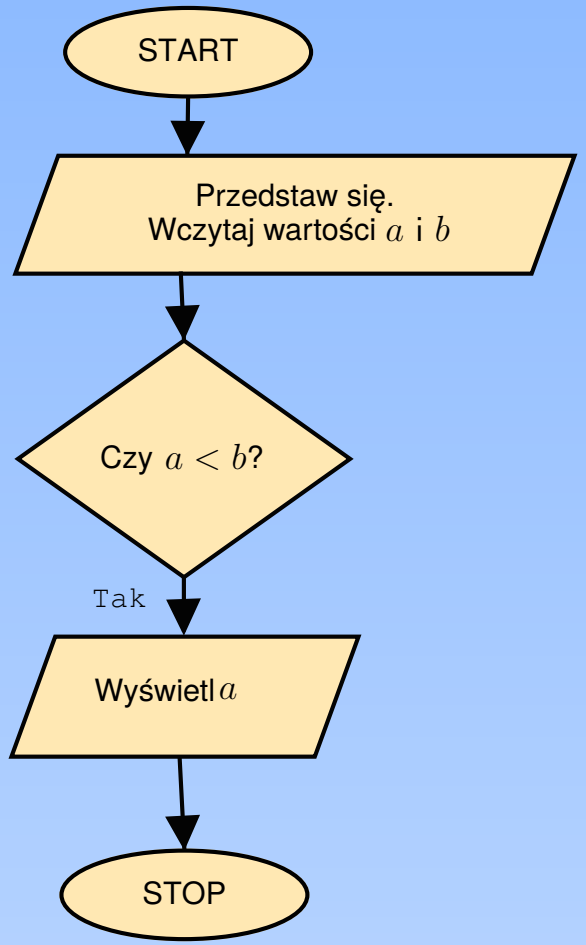

Etapy budowy
systemu
informatycznego
dla przedsiębiorstwa
To, co klient zamówił
To, co analityk zrozumiał
To, co opisywał projekt
To, co wykonali programiści
Po uruchomieniu i wdrożeniu
To, za co klient zapłacił
To, czego klient potrzebował
Autor nieznany

Przykładowy algorytm i jego realizacja

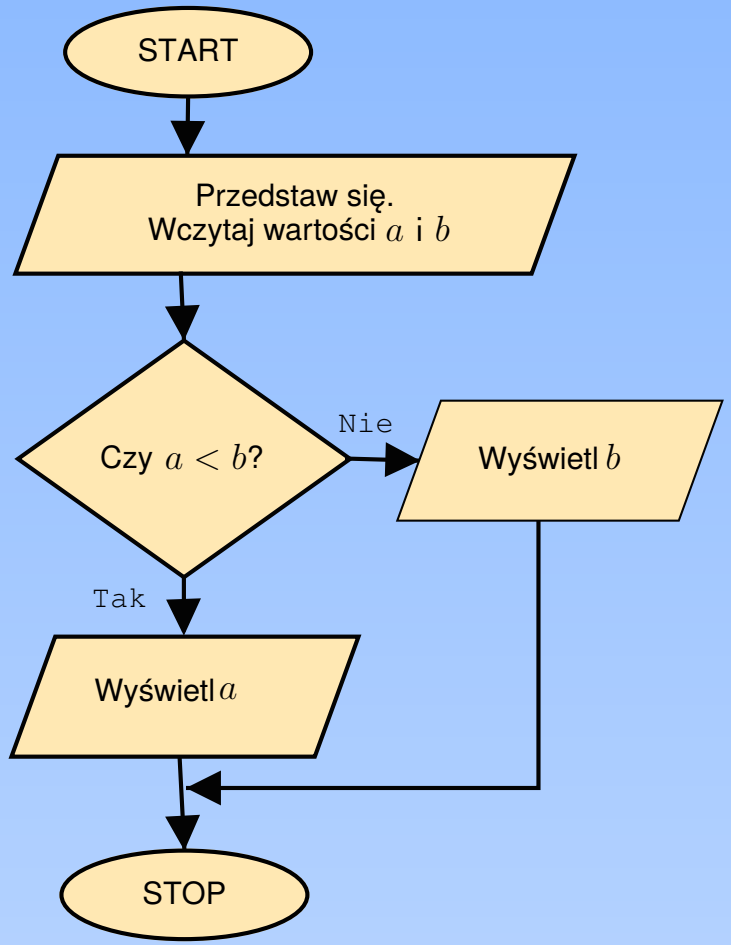
Przykładowy algorytm i jego realizacja



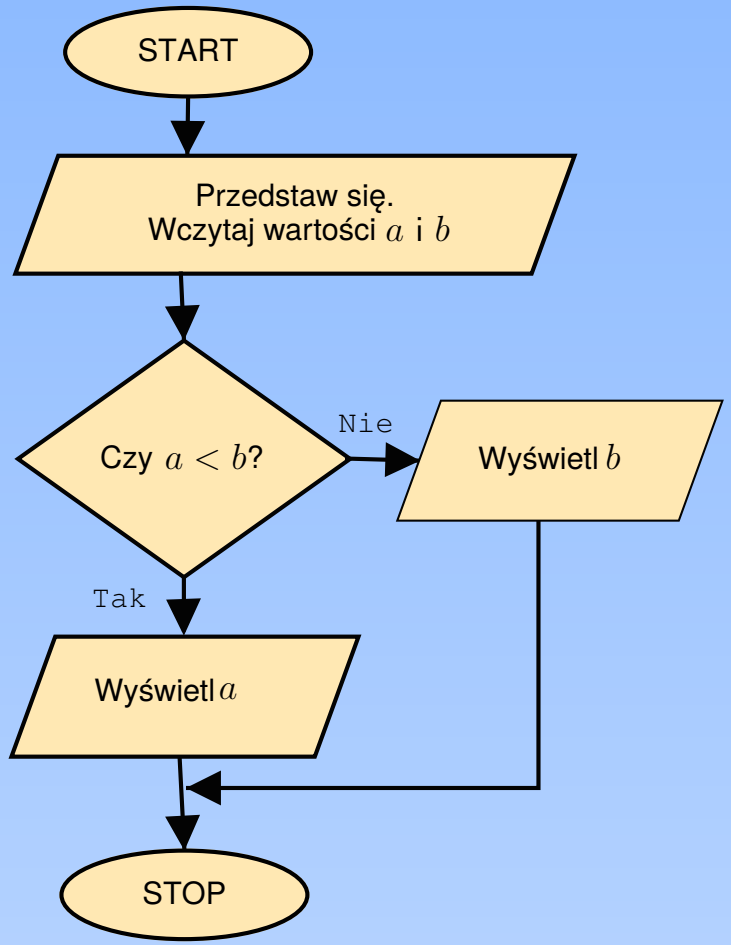
Przykładowy algorytm i jego realizacja



Przykładowy algorytm i jego realizacja



Przykładowy algorytm i jego realizacja



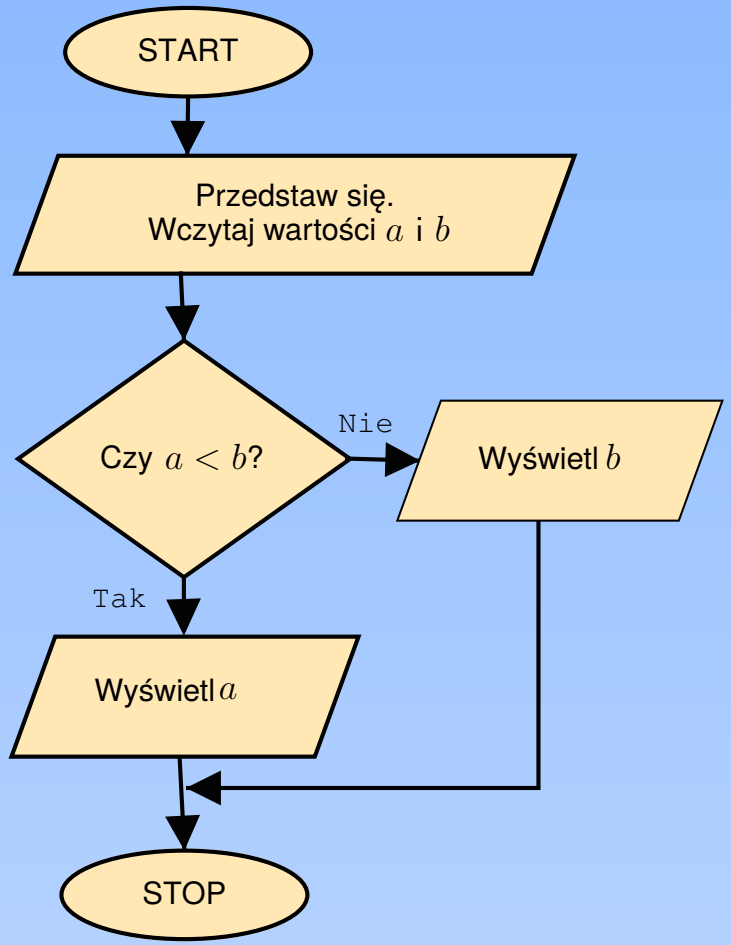

```
#include <stdio.h>

int main() {
  int a, b;

  printf("Program wskazuje mniejsza z ");
  printf("podanych dwoch liczb calkowitych\n");
  printf("Podaj wartosc pierwszej liczby: ");
  scanf("%d", &a);
  printf("Podaj wartosc drugiej liczby: ");
  scanf("%d", &b);

  if (a < b)
    printf("Mniejsza wartoscia jest %d\n",a);
  else
    printf("Mniejsza wartoscia jest %d\n",b);
}
```

Przykładowy algorytm i jego realizacja



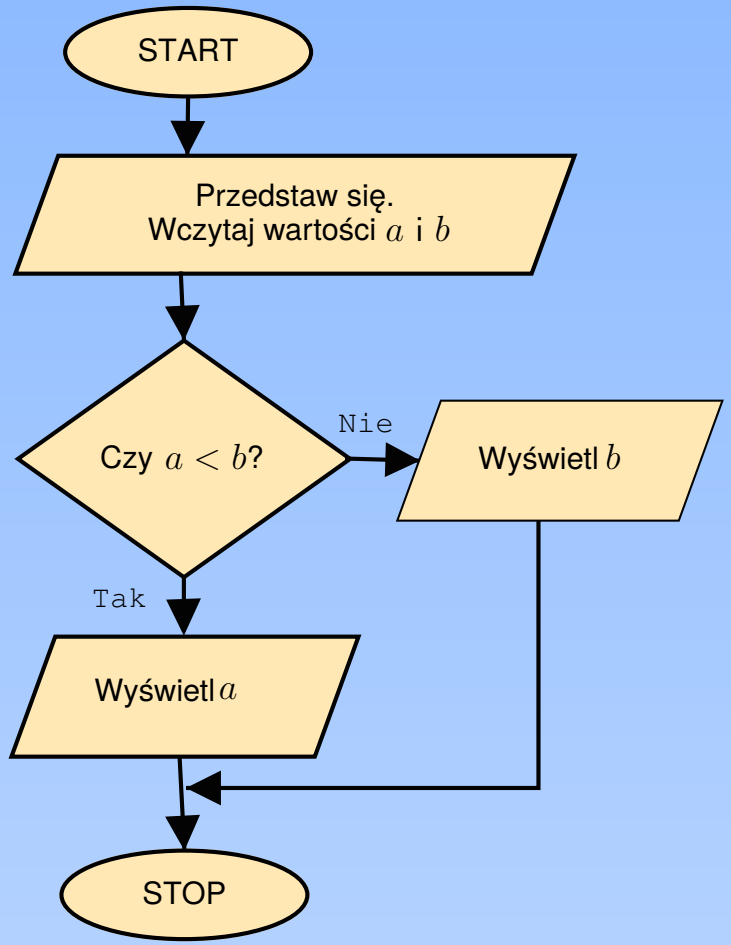

```
#include <stdio.h>

int main() {
  int a, b;

  printf("Program wskazuje mniejsza z ");
  printf("podanych dwoch liczb calkowitych\n");
  printf("Podaj wartosc pierwszej liczby: ");
  scanf("%d", &a);
  printf("Podaj wartosc drugiej liczby: ");
  scanf("%d", &b);

  if (a < b)
    printf("Mniejsza wartoscia jest %d\n",a);
  else
    printf("Mniejsza wartoscia jest %d\n",b);
}
```

Przykładowy algorytm i jego realizacja



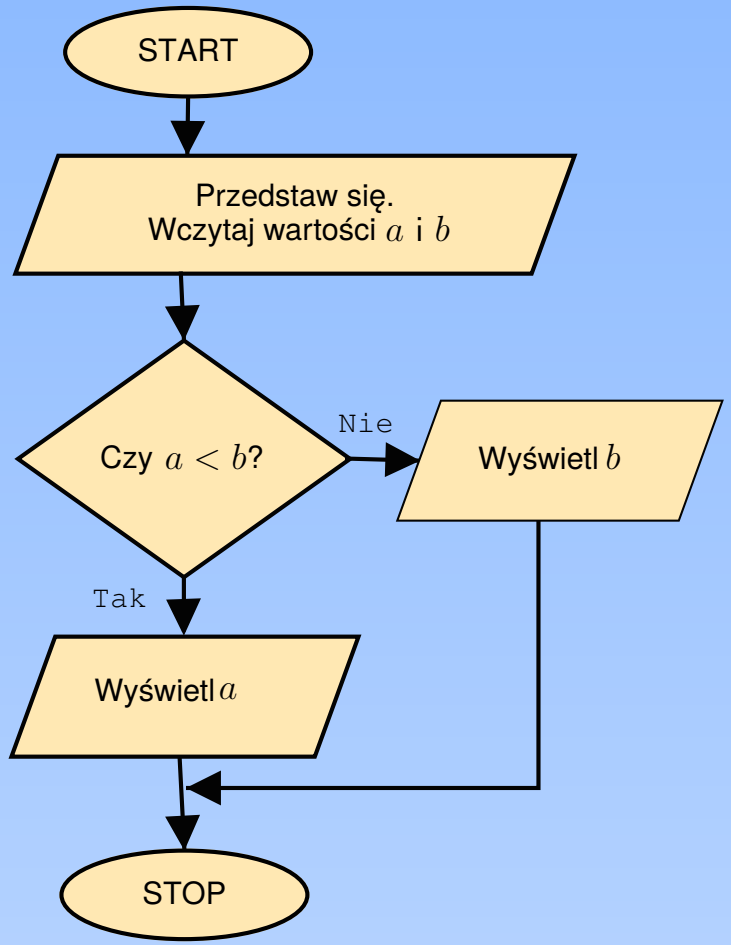

```
#include <stdio.h>

int main() {
  int a, b;

  printf("Program wskazuje mniejsza z ");
  printf("podanych dwoch liczb calkowitych\n");
  printf("Podaj wartosc pierwszej liczby: ");
  scanf("%d", &a);
  printf("Podaj wartosc drugiej liczby: ");
  scanf("%d", &b);

  if (a < b)
    printf("Mniejsza wartoscia jest %d\n",a);
  else
    printf("Mniejsza wartoscia jest %d\n",b);
}
```

Przykładowy algorytm i jego realizacja



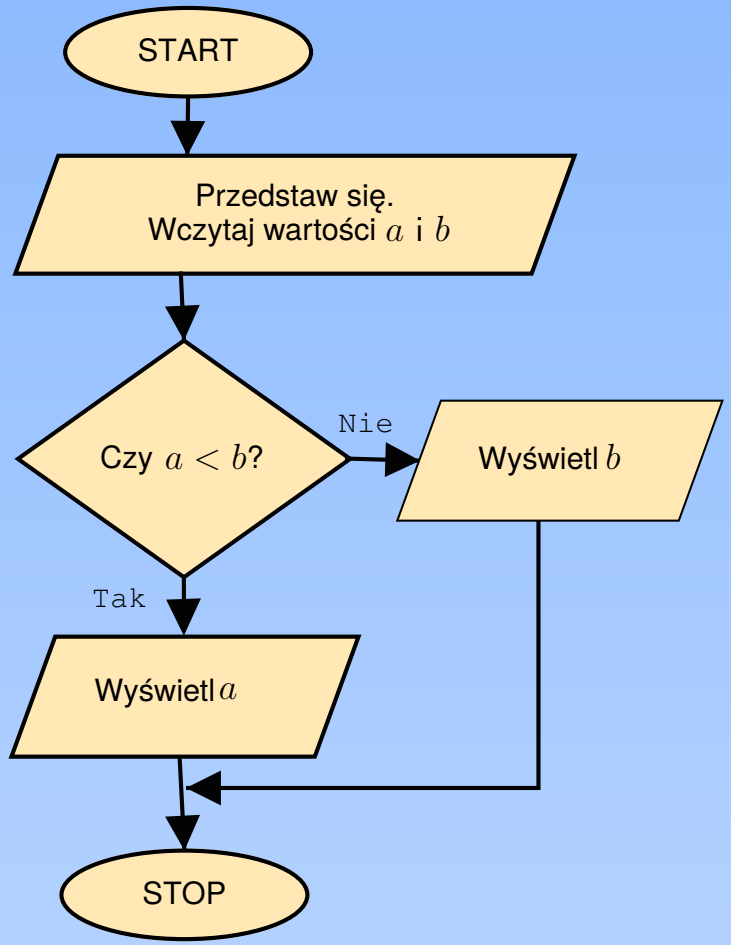

```
#include <stdio.h>

int main() {
  int a, b;

  printf("Program wskazuje mniejsza z ");
  printf("podanych dwoch liczb calkowitych\n");
  printf("Podaj wartosc pierwszej liczby: ");
  scanf("%d", &a);
  printf("Podaj wartosc drugiej liczby: ");
  scanf("%d", &b);

  if (a < b)
    printf("Mniejsza wartoscia jest %d\n",a);
  else
    printf("Mniejsza wartoscia jest %d\n",b);
}
```

Przykładowy algorytm i jego realizacja



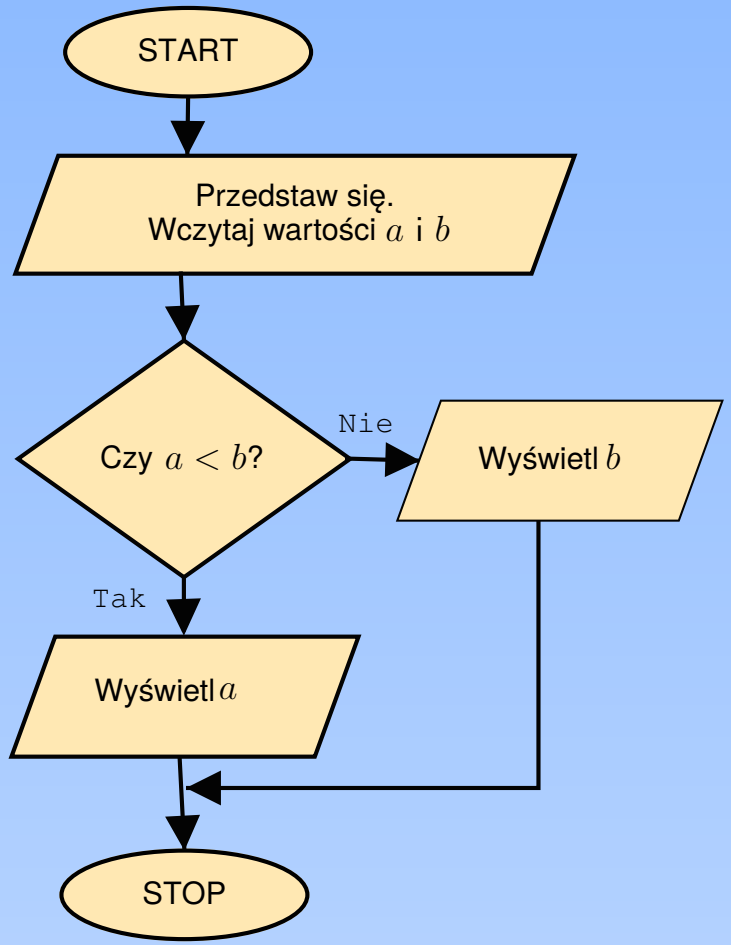

```
#include <stdio.h>

int main() {
  int a, b;

  printf("Program wskazuje mniejsza z ");
  printf("podanych dwoch liczb calkowitych\n");
  printf("Podaj wartosc pierwszej liczby: ");
  scanf("%d", &a);
  printf("Podaj wartosc drugiej liczby: ");
  scanf("%d", &b);

  if (a < b)
    printf("Mniejsza wartoscia jest %d\n",a);
  else
    printf("Mniejsza wartoscia jest %d\n",b);
}
```

Przykładowy algorytm i jego realizacja



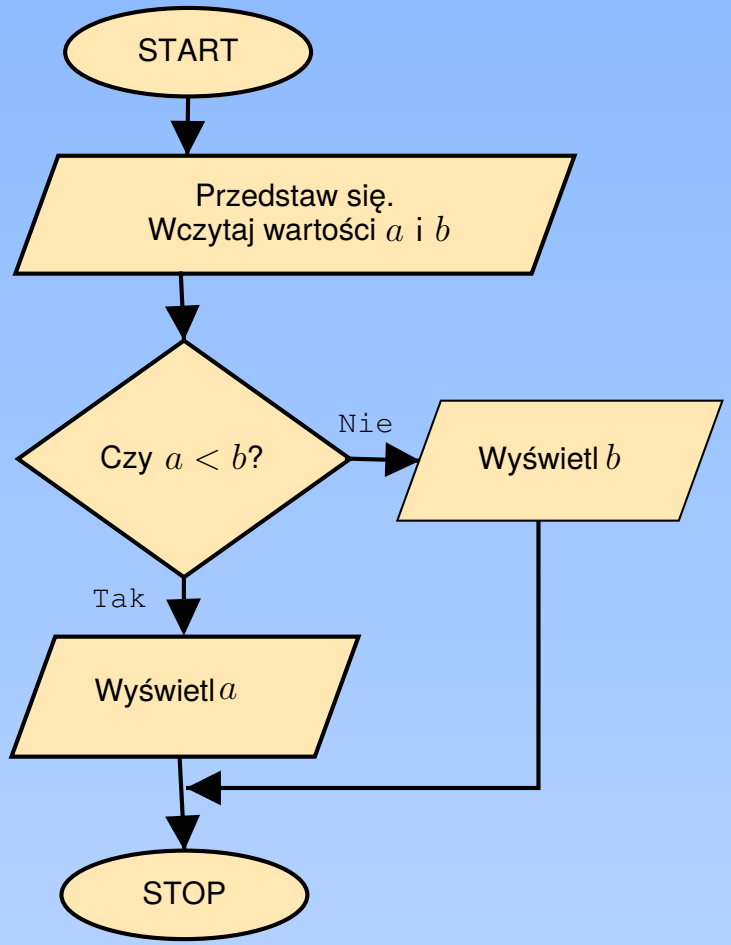

```
#include <stdio.h>

int main() {
  int a, b;

  printf("Program wskazuje mniejsza z ");
  printf("podanych dwoch liczb calkowitych\n");
  printf("Podaj wartosc pierwszej liczby: ");
  scanf("%d", &a);
  printf("Podaj wartosc drugiej liczby: ");
  scanf("%d", &b);

  if (a < b)
    printf("Mniejsza wartoscia jest %d\n",a);
  else
    printf("Mniejsza wartoscia jest %d\n",b);
}
```

Przykładowy algorytm i jego realizacja



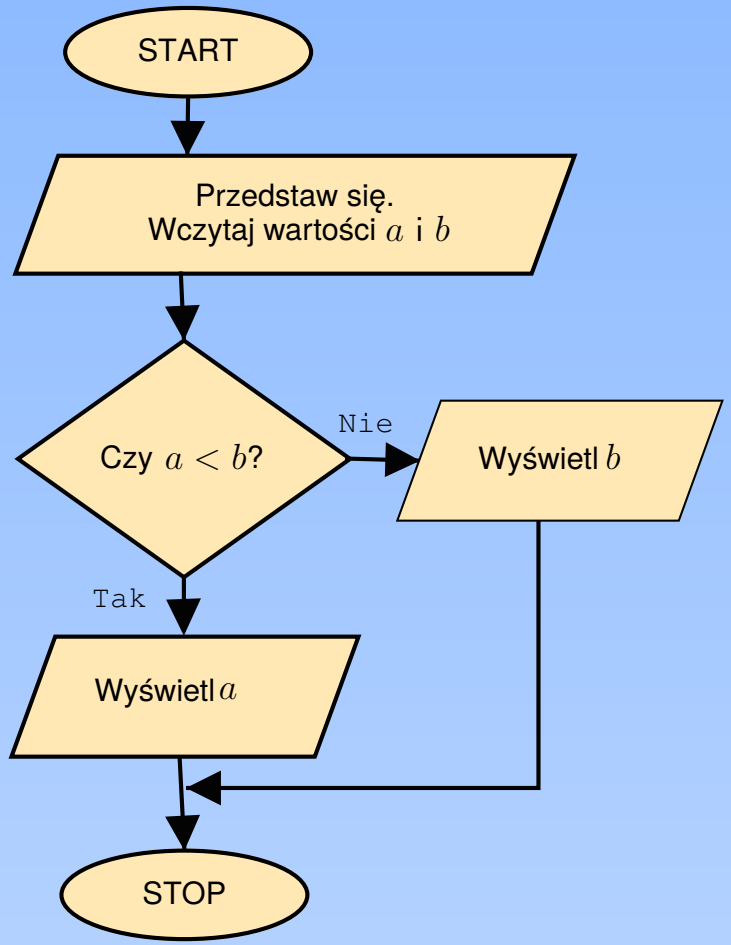

```
#include <stdio.h>

int main() {
  int a, b;

  printf("Program wskazuje mniejsza z ");
  printf("podanych dwoch liczb calkowitych\n");
  printf("Podaj wartosc pierwszej liczby: ");
  scanf("%d", &a);
  printf("Podaj wartosc drugiej liczby: ");
  scanf("%d", &b);

  if (a < b)
    printf("Mniejsza wartoscia jest %d\n",a);
  else
    printf("Mniejsza wartoscia jest %d\n",b);
}
```

Przykładowy algorytm i jego realizacja




```
#include <stdio.h>

int main() {
  int a, b;

  printf("Program wskazuje mniejsza z ");
  printf("podanych dwoch liczb calkowitych\n");
  printf("Podaj wartosc pierwszej liczby: ");
  scanf("%d", &a);
  printf("Podaj wartosc drugiej liczby: ");
  scanf("%d", &b);

  if (a < b)
    printf("Mniejsza wartoscia jest %d\n",a);
  else
    printf("Mniejsza wartoscia jest %d\n",b);
}
```

Składnia programu

- **Notacja MBNF**

LHS = RHS

Składnia programu

- **Notacja MBNF**

LHS = RHS

⋆ Symbole nieterminalne: zdania, grupy podmiotu;

Składnia programu

- **Notacja MBNF**

LHS = RHS

  - ⋆ Symbole nieterminalne: zdania, grupy podmiotu;
  - ⋆ symbole terminalne: „bezbarwne”, „zielone”, „pomysły”, „śpią”, „wściekle”;

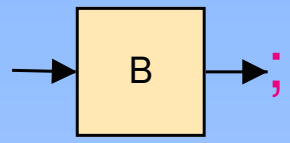
Składnia programu

- **Notacja MBNF**

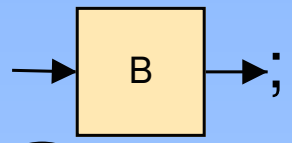
LHS = RHS

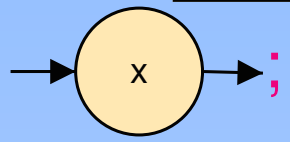
- ⋆ Symbole nieterminalne: zdania, grupy podmiotu;
- ⋆ symbole terminalne: „bezbarwne”, „zielone”, „pomysły”, „śpią”, „wściekle”;
- ⋆ operatory: konkatenacja, alternatywa — `|`, opcja — `[]`, powtórzenie — `{}`, grupowanie — `()`.

Składnia programu

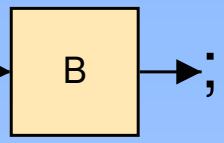
- **Diagramy składni**
  - ⋆ Symbole nieterminalne $\rightarrow$ 

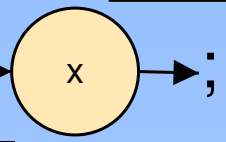
Składnia programu

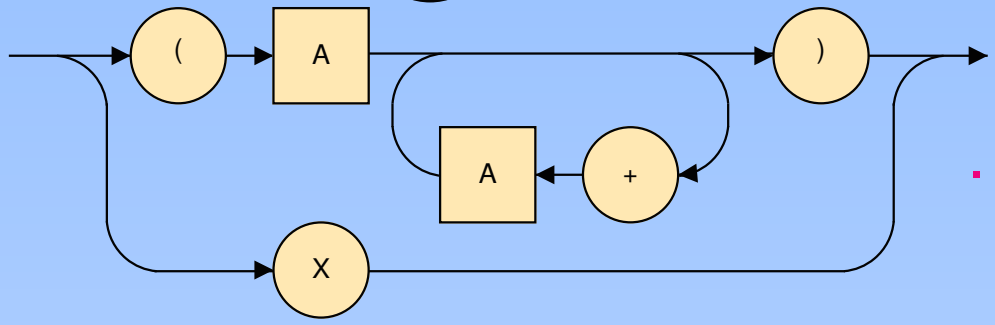
- **Diagramy składni**
  - ⋆ Symbole nieterminalne → 

  - ⋆ symbole terminalne → 

Składnia programu

- **Diagramy składni**
  - ⋆ Symbole nieterminalne → 

  - ⋆ symbole terminalne → 

  - ⋆ operatory → 

.

Przykładowe konstrukcje

- **Notacja MBNF**

```
liczba-calkowita = [ znak-liczby ]
                         liczba-calkowita-bez-znaku .
```

Przykładowe konstrukcje

- **Notacja MBNF**

```
liczba-calkowita = [ znak-liczby ]
                         liczba-calkowita-bez-znaku .
znak-liczby = "+" | "-" .
```

Przykładowe konstrukcje

- **Notacja MBNF**

```
liczba-calkowita = [ znak-liczby ]
                         liczba-calkowita-bez-znaku .
znak-liczby = "+" | "-" .
liczba-calkowita-bez-znaku = ciag-cyfr .
```

Przykładowe konstrukcje

- **Notacja MBNF**

```
liczba-calkowita = [ znak-liczby ]
                         liczba-calkowita-bez-znaku .
znak-liczby = "+" | "-" .
liczba-calkowita-bez-znaku = ciag-cyfr .
ciag-cyfr = cyfra { cyfra } .
```

Przykładowe konstrukcje

- **Notacja MBNF**

```
liczba-calkowita = [ znak-liczby ]
                          liczba-calkowita-bez-znaku .
znak-liczby = "+" | "-" .
liczba-calkowita-bez-znaku = ciag-cyfr .
ciag-cyfr = cyfra { cyfra } .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6"
      | "7" | "8" | "9" .
```

Przykładowe konstrukcje

- **Notacja MBNF**

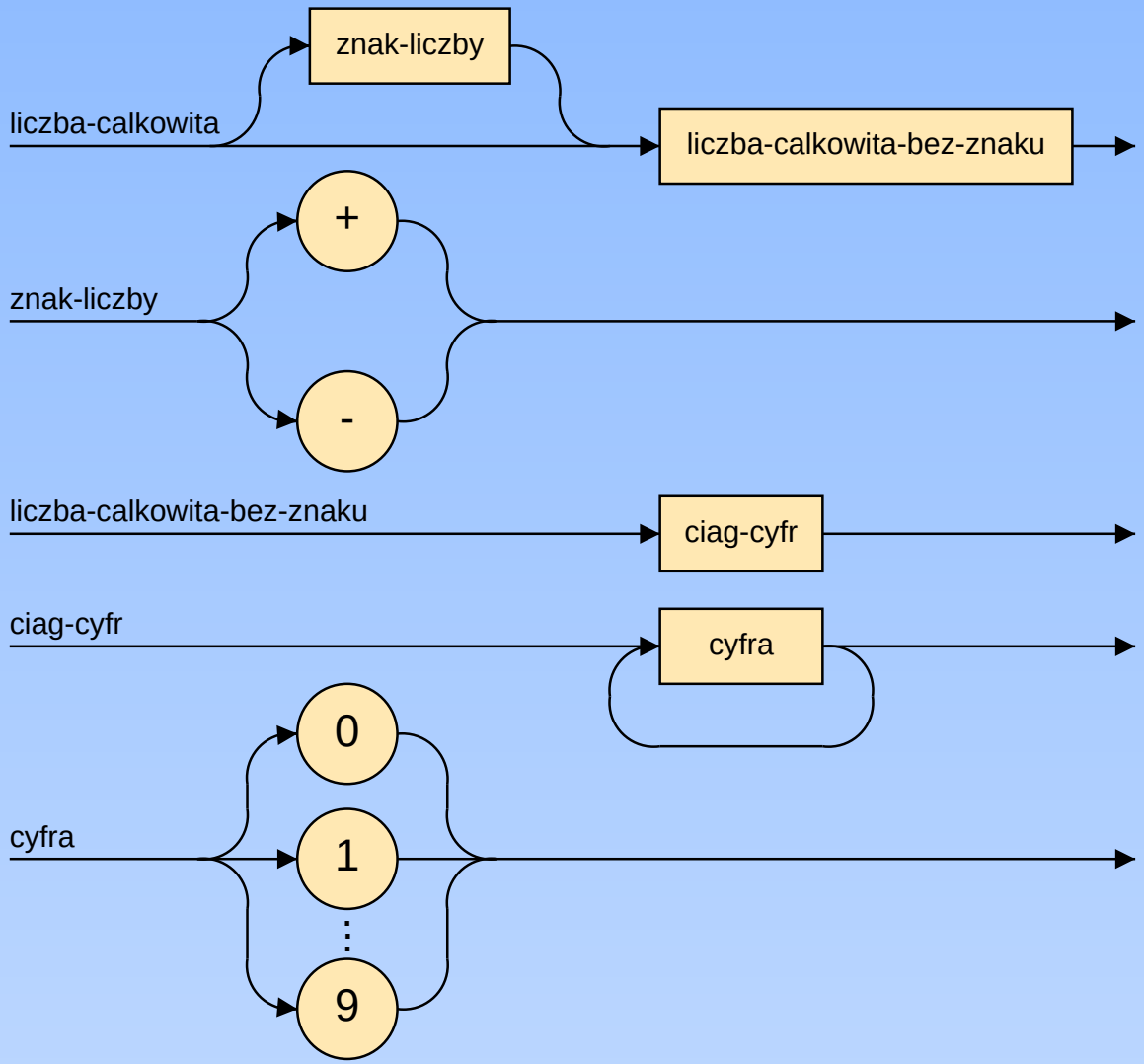
```
liczba-calkowita = [ znak-liczby ]
                         liczba-calkowita-bez-znaku .
znak-liczby = "+" | "-" .
liczba-calkowita-bez-znaku = ciag-cyfr .
ciag-cyfr = cyfra { cyfra } .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6"
      | "7" | "8" | "9" .
```

Porównaj

```
ciag-cyfr = { cyfra } .
```

Przykładowe konstrukcje

- **Diagram składni**



Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
                          dzien "-" miesiac "-" rok .
```

Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
                              dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
```

Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
                              dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
```

Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
                              dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
```

Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
                              dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
                                        "7" | "8" | "9" .
```

Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
                              dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
                                        "7" | "8" | "9" .
```

Które z poniższych ciągów znaków spełniają definicję `data`? Jeśli nie spełniają to dlaczego?

12-12-12

Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
                              dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
                                        "7" | "8" | "9" .
```

Które z poniższych ciągów znaków spełniają definicję `data`? Jeśli nie spełniają to dlaczego?

`12-12-12`

`3.10.67`

Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
                              dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
                                        "7" | "8" | "9" .
```

Które z poniższych ciągów znaków spełniają definicję `data`? Jeśli nie spełniają to dlaczego?

`12-12-12`

`3.10.67`

`0-0-00`

Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
                              dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
                                        "7" | "8" | "9" .
```

Które z poniższych ciągów znaków spełniają definicję `data`? Jeśli nie spełniają to dlaczego?

`12-12-12` `00-00-00`

`3.10.67`

`0-0-00`

Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
                              dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
                                        "7" | "8" | "9" .
```

Które z poniższych ciągów znaków spełniają definicję `data`? Jeśli nie spełniają to dlaczego?

`12-12-12` `00-00-00`

`3.10.67` `47-19-99`

`0-0-00`

Notacja MBNF — dalsze przykłady

```
data = dzien "." miesiac "." rok |
                              dzien "-" miesiac "-" rok .
dzien = cyfra | cyfra cyfra .
miesiac = [ "1" ] cyfra .
rok = cyfra cyfra | cyfra cyfra cyfra cyfra .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" |
                                        "7" | "8" | "9" .
```

Które z poniższych ciągów znaków spełniają definicję `data`? Jeśli nie spełniają to dlaczego?

`12-12-12`	`00-00-00`
`3.10.67`	`47-19-99`
`0-0-00`	`12-2-997`

Notacja MBNF — dalsze przykłady

```
ciag = "A" [ ciag ] "A" | "B" [ ciag ] "B" | "A" | "B" .
```

Notacja MBNF — dalsze przykłady

```
ciag = "A" [ ciag ] "A" | "B" [ ciag ] "B" | "A" | "B" .
```

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

ABBA

Notacja MBNF — dalsze przykłady

```
ciag = "A" [ ciag ] "A" | "B" [ ciag ] "B" | "A" | "B" .
```

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

ABBA

BABA

Notacja MBNF — dalsze przykłady

```
ciag = "A" [ ciag ] "A" | "B" [ ciag ] "B" | "A" | "B" .
```

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

ABBA

BABA

ABABA

Notacja MBNF — dalsze przykłady

```
ciag = "A" [ ciag ] "A" | "B" [ ciag ] "B" | "A" | "B" .
```

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

ABBA

BABA

ABABA

AAABAAA

Notacja MBNF — dalsze przykłady

```
ciag = "A" [ ciag ] "A" | "B" [ ciag ] "B" | "A" | "B" .
```

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

ABBA

BABA

ABABA

AAABAAA

ABABA

Notacja MBNF — dalsze przykłady

```
ciag = "A" [ ciag ] "A" | "B" [ ciag ] "B" | "A" | "B" .
```

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

ABBA

BABA

ABABA

AAABAAA

ABABA

AAABBAABABBABBABAABBAAA

Notacja MBNF — dalsze przykłady

```
liczba = [ znak-liczby ] liczba-bez-znaku .
```

Notacja MBNF — dalsze przykłady

```
liczba = [ znak-liczby ] liczba-bez-znaku .
znak-liczby = "+" | "-" .
```

Notacja MBNF — dalsze przykłady

```
liczba = [ znak-liczby ] liczba-bez-znaku .
znak-liczby = "+" | "-" .
liczba-bez-znaku = cyfra { cyfra } .
```

Notacja MBNF — dalsze przykłady

```
liczba = [ znak-liczby ] liczba-bez-znaku .
znak-liczby = "+" | "-" .
liczba-bez-znaku = cyfra { cyfra } .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" .
```

Notacja MBNF — dalsze przykłady

```
liczba = [ znak-liczby ] liczba-bez-znaku .
znak-liczby = "+" | "-" .
liczba-bez-znaku = cyfra { cyfra } .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" .
```

Czy w sensie powyższych reguł poprawna jest liczba?

13

Notacja MBNF — dalsze przykłady

```
liczba = [ znak-liczby ] liczba-bez-znaku .
znak-liczby = "+" | "-" .
liczba-bez-znaku = cyfra { cyfra } .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" .
```

Czy w sensie powyższych reguł poprawna jest liczba?

13 +13

Notacja MBNF — dalsze przykłady

```
liczba = [ znak-liczby ] liczba-bez-znaku .
znak-liczby = "+" | "-" .
liczba-bez-znaku = cyfra { cyfra } .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" .
```

Czy w sensie powyższych reguł poprawna jest liczba?

13 +13 +7

Notacja MBNF — dalsze przykłady

```
liczba = [ znak-liczby ] liczba-bez-znaku .
znak-liczby = "+" | "-" .
liczba-bez-znaku = cyfra { cyfra } .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" .
```

Czy w sensie powyższych reguł poprawna jest liczba?

13 +13 +7 -666

Notacja MBNF — dalsze przykłady

```
jedzonko = ( ( bc ) | rmw | bcrmw )
                          ( p | h [ k ] ) [ d ] .
b = "bigos". c = "chleb". r = "rogalik". m = "maslo".
w = "wedlina". p = "piwo". h = "herbata".
k = "cukier". d = "deser".
```

Które z poniższych ciągów znaków spełniają definicję? Jeśli nie spełniają to dlaczego?

```
p
pd
brmpd
rmwpk
bcrmwh
bcp
```

Notacja MBNF — dalsze przykłady

```
piwo = "Piast" | "EB" | "Lech" | "Zywiec" | "Okocim" .
menu = piwo { piwo } .
```

Zamówienie: Dwa piwa prosze

```
zamowienie = piwo piwo .
```

Zamówienie: Obojętne co i czy w ogóle, ale jeśli cokolwiek to jedno

```
zamowienie = [ piwo ] .
```

Zamówienie: Dwa Żywce lub EB i Lecha prosze

```
zamowienie = [ "Zywiec" "Zywiec" ] | ( "EB" "Lech") .
```

Notacja MBNF — dalsze przykłady

Czy poniższe konstrukcje definiują liczby całkowite?

```
liczba-calkowita = [ znak-liczby ]
                          liczba-calkowita-bez-znaku .
znak-liczby = "+" | "-" .
liczba-calkowita-bez-znaku = ciag-cyfr .
ciag-cyfr = cyfra { cyfra } .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6"
      | "7" | "8" | "9" .
konstrukcja = cyfra .
konstrukcja = "+" cyfra cyfra cyfra .
konstrukcja = [ "-" | "+" ] cyfra { cyfra } .
konstrukcja = [ "-" | "+" ] { cyfra } .
konstrukcja = [ ( "+" | "-" ) [ cyfra [ cyfra ]]] .
```

Notacja MBNF — rownoważność reguł

Czy poniższe pary reguł są sobie równoważne?

```
(1) cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" .
```

```
(2) cyfra = mala-cyfra | duza-cyfra .
    mala-cyfra = "0" | "1" | "2" | "3" | "4" .
    duza-cyfra = "3" | "4" | 5" | "6" .
```

```
(1) czas = godzina [ ":" minuta ] [ ":" sekunda ] .
```

```
(2) czas = godzina [ ":" minuta [ ":" sekunda ] ] .
```

```
(1) ciag-cyfr = { cyfra } .
```

```
(2) ciag-cyfr = "" | cyfra ciag-cyfr .
```

Notacja MBNF — rownoważność reguł

Czy poniższe pary reguł są sobie równoważne?

```
(1) liczba = cyfra cyfra { cyfra cyfra } .

(2) liczba = cyfra cyfra .
    liczba = liczba liczba .
```

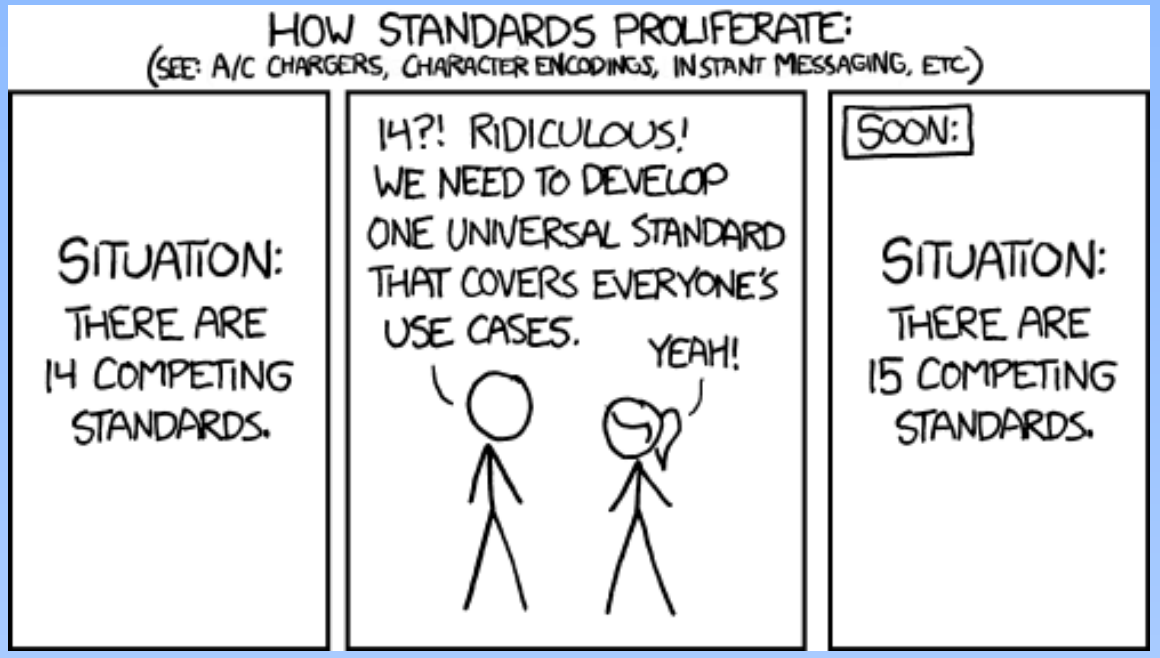
```
(1) nr-telefonu = cyf cyf "-" cyf cyf "-" cyf cyf .

(2) nr-telefonu = cyf cyf cyf "-" cyf cyf cyf .
```

```
(1) w1 = cyfra { cyfra } .

(2) w2 = { cyfra } cyfra .
```

O standaryzacji (by xkcd)



Kategorie składniowe C – komentarze

```
komentarz   = "/*" ciag-znakow "*/" .
```

Kategorie składniowe C – komentarze

```
komentarz    = "/*" ciag-znakow "*/" .
ciag-znakow = { znak } .
```

Kategorie składniowe C – komentarze

```
komentarz    = "/*" ciag-znakow "*/" .
ciag-znakow = { znak } .
```

znak = dowolny znak .

Kategorie składniowe C – komentarze

```
komentarz    = "/*" ciag-znakow "*/" .
ciag-znakow = { znak } .
znak =  dowolny znak .
```

Przykłady:

```
lower = 0     /* dolna granica temperatur */
```

Kategorie składniowe C – komentarze

komentarz = "/*" ciag-znakow "*/" .

ciag-znakow = { znak } .

znak = dowolny znak .

Przykłady:

```
 lower = 0    /* dolna granica temperatur */
```

```
/* wypisanie zestawienia temperatur w roznych skalach dla wartosci
   znajdujacych sie w zakresie 0..300; wersja zmiennopozycyjna */
```

Kategorie składniowe C – komentarze

komentarz = "/*" ciag-znakow "*/" .

ciag-znakow = { znak } .

znak = dowolny znak .

Przykłady:

```
 lower = 0    /* dolna granica temperatur */

/* wypisanie zestawienia temperatur w roznych skalach dla wartosci
   znajdujacych sie w zakresie 0..300; wersja zmiennopozycyjna */

/* to jest komentarz */ ale to nie jest juz komentarz */
```

Kategorie składniowe C – komentarze

komentarz = "/*" ciag-znakow "*/" .

ciag-znakow = { znak } .

znak = dowolny znak .

Przykłady:

```
 lower = 0    /* dolna granica temperatur */

/* wypisanie zestawienia temperatur w roznych skalach dla wartosci
   znajdujacych sie w zakresie 0..300; wersja zmiennopozycyjna */

/* to jest komentarz */ ale to nie jest juz komentarz */

/* to jest komentarz /* i to tez */ ale to niestety nie */
```

Kategorie składniowe C – komentarze

```
komentarz    = "/*" ciag-znakow "*/" .
ciag-znakow = { znak } .
znak =  dowolny znak .
```

Przykłady:

```
 lower = 0    /* dolna granica temperatur */

/* wypisanie zestawienia temperatur w roznych skalach dla wartosci
   znajdujacych sie w zakresie 0..300; wersja zmiennopozycyjna */

/* to jest komentarz */ ale to nie jest juz komentarz */

/* to jest komentarz /* i to tez */ ale to niestety nie */

int/*/ integer * /*/*var;     /* a tu gdzie jest komentarz :) */
```

Kategorie składniowe C – identyfikatory

```
identyfikator = ( litera | podkreslenie )
                { litera | podkreslenie | cyfra } .
```

Kategorie składniowe C – identyfikatory

```
identyfikator = ( litera | podkreslenie )
                { litera | podkreslenie | cyfra } .
```

`litera` = małe i duże litery alfabetu łacińskiego

`podkreslenie = "_" .`

`cyfra` = cyfry dziesiętne

Identyfikator nie może pokrywać się ze słowami kluczowymi.

Kategorie składniowe C – identyfikatory

```
identyfikator = ( litera | podkreslenie )
                { litera | podkreslenie | cyfra } .
```

`litera` = małe i duże litery alfabetu łacińskiego

`podkreslenie = "_" .`

`cyfra` = cyfry dziesiętne

Identyfikator nie może pokrywać się ze słowami kluczowymi.

Które z napisów są poprawnymi identyfikatorami?

`moja_dana` `Moja_Dana` `MOJA_DANA` `moja_dana2`

Kategorie składniowe C – identyfikatory

```
identyfikator = ( litera | podkreslenie )
                { litera | podkreslenie | cyfra } .
```

`litera` = małe i duże litery alfabetu łacińskiego

`podkreslenie = "_" .`

`cyfra` = cyfry dziesiętne

Identyfikator nie może pokrywać się ze słowami kluczowymi.

Które z napisów są poprawnymi identyfikatorami?

`moja_dana` `Moja_Dana` `MOJA_DANA` `moja_dana2`

`moja` `moja_` `moja__dana` `moja-dana`

Kategorie składniowe C – identyfikatory

```
identyfikator = ( litera | podkreslenie )
                { litera | podkreslenie | cyfra } .
```

`litera` = małe i duże litery alfabetu łacińskiego

`podkreslenie = "_" .`

`cyfra` = cyfry dziesiętne

Identyfikator nie może pokrywać się ze słowami kluczowymi.

Które z napisów są poprawnymi identyfikatorami?

```
moja_dana  Moja_Dana  MOJA_DANA  moja_dana2
moja       moja_      moja__dana moja-dana
moja dana
```

Kategorie składniowe C – identyfikatory

```
identyfikator = ( litera | podkreslenie )
                { litera | podkreslenie | cyfra } .
```

`litera` = małe i duże litery alfabetu łacińskiego

`podkreslenie = "_" .`

`cyfra` = cyfry dziesiętne

Identyfikator nie może pokrywać się ze słowami kluczowymi.

Które z napisów są poprawnymi identyfikatorami?

```
moja_dana  Moja_Dana  MOJA_DANA  moja_dana2
moja       moja_      moja__dana moja-dana
moja dana  moja^dana
```

Kategorie składniowe C – identyfikatory

```
identyfikator = ( litera | podkreslenie )
                { litera | podkreslenie | cyfra } .
```

`litera` = małe i duże litery alfabetu łacińskiego

`podkreslenie = "_" .`

`cyfra` = cyfry dziesiętne

Identyfikator nie może pokrywać się ze słowami kluczowymi.

Które z napisów są poprawnymi identyfikatorami?

```
moja_dana  Moja_Dana  MOJA_DANA  moja_dana2
moja       moja_      moja__dana moja-dana
moja dana  moja^dana  tmp21d3233
```

Kategorie składniowe C – identyfikatory

```
identyfikator = ( litera | podkreslenie )
                { litera | podkreslenie | cyfra } .
```

`litera` = małe i duże litery alfabetu łacińskiego

`podkreslenie = "_" .`

`cyfra` = cyfry dziesiętne

Identyfikator nie może pokrywać się ze słowami kluczowymi.

Które z napisów są poprawnymi identyfikatorami?

```
moja_dana  Moja_Dana  MOJA_DANA  moja_dana2
moja       moja_      moja__dana moja-dana
moja dana  moja^dana  tmp21d3233 12tmp
```

Kategorie składniowe C – identyfikatory

```
identyfikator = ( litera | podkreslenie )
                { litera | podkreslenie | cyfra } .
```

`litera` = małe i duże litery alfabetu łacińskiego

`podkreslenie = "_" .`

`cyfra` = cyfry dziesiętne

Identyfikator nie może pokrywać się ze słowami kluczowymi.

Które z napisów są poprawnymi identyfikatorami?

```
moja_dana  Moja_Dana  MOJA_DANA  moja_dana2
moja       moja_      moja__dana moja-dana
moja dana  moja^dana  tmp21d3233 12tmp
```

A przy poniższej definicji?

```
identyfikator = ( litera | podkreslenie )
                { [ podkreslenie ] ( litera | cyfra ) } .
```

Kategorie składniowe C – słowa kluczowe

```
slowo-kluczowe   = "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"
                 | "union" | "unsigned" | "void" | "volatile"
                 | "while" .
```

Kategorie składniowe C – literały

```
literal = literal_arytmetyczny | literal_znakowy
                               | literal_tekstowy .
```

Kategorie składniowe C – literały

```
literal = literal_arytmetyczny | literal_znakowy
                               | literal_tekstowy .
literal_arytmetyczny = literal_stalopozycyjny
                     | literal_zmiennopozycyjny .
```

Kategorie składniowe C – literały

```
literal = literal_arytmetyczny | literal_znakowy
                               | literal_tekstowy .
literal_arytmetyczny = literal_stalopozycyjny
                     | literal_zmiennopozycyjny .
```

```
literal_stalopozycyjny =
      [ znak-liczby ] ( ciag-cyfr | ciag-cyfr-szesnastkowych )
      [ "l" | "L" | "u" | "U" | "ul" | "UL" ] .
```

Kategorie składniowe C – literały

```
literal = literal_arytmetyczny | literal_znakowy
                               | literal_tekstowy .
literal_arytmetyczny = literal_stalopozycyjny
                     | literal_zmiennopozycyjny .
```

```
literal_stalopozycyjny =
      [ znak-liczby ] ( ciag-cyfr | ciag-cyfr-szesnastkowych )
      [ "l" | "L" | "u" | "U" | "ul" | "UL" ] .
znak-liczby = "+" | "-" .
```

Kategorie składniowe C – literały

```
literal = literal_arytmetyczny | literal_znakowy
                               | literal_tekstowy .

literal_arytmetyczny = literal_stalopozycyjny
                     | literal_zmiennopozycyjny .
```

```
literal_stalopozycyjny =
      [ znak-liczby ] ( ciag-cyfr | ciag-cyfr-szesnastkowych )
      [ "l" | "L" | "u" | "U" | "ul" | "UL" ] .

znak-liczby = "+" | "-" .

ciag-cyfr = cyfra { cyfra } .
```

Kategorie składniowe C – literały

```
literal = literal_arytmetyczny | literal_znakowy
                               | literal_tekstowy .

literal_arytmetyczny = literal_stalopozycyjny
                     | literal_zmiennopozycyjny .
```

```
literal_stalopozycyjny =
      [ znak-liczby ] ( ciag-cyfr | ciag-cyfr-szesnastkowych )
      [ "l" | "L" | "u" | "U" | "ul" | "UL" ] .

znak-liczby = "+" | "-" .

ciag-cyfr = cyfra { cyfra } .

ciag-cyfr-szesnastkowych = ( "0x" | "0X" ) cyfra-szesnastkowa
                                           { cyfra-szesnastkowa } .
```

Kategorie składniowe C – literały

```
literal = literal_arytmetyczny | literal_znakowy
                               | literal_tekstowy .

literal_arytmetyczny = literal_stalopozycyjny
                     | literal_zmiennopozycyjny .
```

```
literal_stalopozycyjny =
      [ znak-liczby ] ( ciag-cyfr | ciag-cyfr-szesnastkowych )
      [ "l" | "L" | "u" | "U" | "ul" | "UL" ] .

znak-liczby = "+" | "-" .

ciag-cyfr = cyfra { cyfra } .

ciag-cyfr-szesnastkowych = ( "0x" | "0X" ) cyfra-szesnastkowa
                                           { cyfra-szesnastkowa } .

cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6"
      | "7" | "8" | "9" .

cyfra-szesnastkowa = cyfra | "a" | "b" | "c" | "d" | "e" | "f"
                           | "A" | "B" | "C" | "D" | "E" | "F" .
```

Kategorie składniowe C – literały cd.

```
literal-zmiennopozycyjny =
          [ znak-liczby ] liczba-rzeczywista-bez-znaku .
```

Kategorie składniowe C – literały cd.

```
literal-zmiennopozycyjny =
          [ znak-liczby ] liczba-rzeczywista-bez-znaku .
liczba-rzeczywista-bez-znaku =
  ( ciag-cyfr "." [ ciag-cyfr ] [ ("e"|"E") mnoznik-skalujacy ]
```

Kategorie składniowe C – literały cd.

```
literal-zmiennopozycyjny =
          [ znak-liczby ] liczba-rzeczywista-bez-znaku .

liczba-rzeczywista-bez-znaku =
  ( ciag-cyfr "." [ ciag-cyfr ] [ ("e"|"E") mnoznik-skalujacy ]
  | "."  ciag-cyfr  [ ("e"|"E") mnoznik-skalujacy ]
  | ciag-cyfr ("e"|"E") mnoznik-skalujacy ) [ "f" | "F" | "l" | "L" ] .
```

Kategorie składniowe C – literały cd.

```
literal-zmiennopozycyjny =
          [ znak-liczby ] liczba-rzeczywista-bez-znaku .

liczba-rzeczywista-bez-znaku =
  ( ciag-cyfr "." [ ciag-cyfr ] [ ("e"|"E") mnoznik-skalujacy ]
  | "."  ciag-cyfr  [ ("e"|"E") mnoznik-skalujacy ]
  | ciag-cyfr ("e"|"E") mnoznik-skalujacy ) [ "f" | "F" | "l" | "L" ] .

mnoznik-skalujacy = [ znak-liczby ] ciag-cyfr .
```

Kategorie składniowe C – literały cd.

```
literal-zmiennopozycyjny =
          [ znak-liczby ] liczba-rzeczywista-bez-znaku .
liczba-rzeczywista-bez-znaku =
  ( ciag-cyfr "." [ ciag-cyfr ] [ ("e"|"E") mnoznik-skalujacy ]
  | "."  ciag-cyfr  [ ("e"|"E") mnoznik-skalujacy ]
  | ciag-cyfr ("e"|"E") mnoznik-skalujacy ) [ "f" | "F" | "l" | "L" ] .
mnoznik-skalujacy = [ znak-liczby ] ciag-cyfr .
```

```
literal_znakowy = "'" ( znak-bez-\ | sekwencja-specjalna ) "'" .
```

Kategorie składniowe C – literały cd.

```
literal-zmiennopozycyjny =
          [ znak-liczby ] liczba-rzeczywista-bez-znaku .

liczba-rzeczywista-bez-znaku =
  ( ciag-cyfr "." [ ciag-cyfr ] [ ("e"|"E") mnoznik-skalujacy ]
  | "."  ciag-cyfr  [ ("e"|"E") mnoznik-skalujacy ]
  | ciag-cyfr ("e"|"E") mnoznik-skalujacy ) [ "f" | "F" | "l" | "L" ] .

mnoznik-skalujacy = [ znak-liczby ] ciag-cyfr .
```

```
literal_znakowy = "'" ( znak-bez-\ | sekwencja-specjalna ) "'" .
```

`znak-bez-\` = dowolny znak za wyjątkiem znaku „\” .

Kategorie składniowe C – literały cd.

```
literal-zmiennopozycyjny =
          [ znak-liczby ] liczba-rzeczywista-bez-znaku .

liczba-rzeczywista-bez-znaku =
  ( ciag-cyfr "." [ ciag-cyfr ] [ ("e"|"E") mnoznik-skalujacy ]
  | "."  ciag-cyfr  [ ("e"|"E") mnoznik-skalujacy ]
  | ciag-cyfr ("e"|"E") mnoznik-skalujacy ) [ "f" | "F" | "l" | "L" ] .

mnoznik-skalujacy = [ znak-liczby ] ciag-cyfr .
```

```
literal_znakowy = "'" ( znak-bez-\ | sekwencja-specjalna ) "'" .
```

znak-bez-\ = dowolny znak za wyjątkiem znaku „\” .

```
sekwencja-specjalna = "\\" | "\'" | "\"" | "\?" | "\a" | "\b"
                    | "\f" | "\n" | "\r" | "\t" | "\v"
    | "\" (( cyfra-osemkowa [ cyfra-osemkowa ] [ cyfra-osemkowa ] )
          | ciag-cyfr-szesnastkowych ) .
```

Kategorie składniowe C – literały cd.

```
literal-zmiennopozycyjny =
          [ znak-liczby ] liczba-rzeczywista-bez-znaku .

liczba-rzeczywista-bez-znaku =
  ( ciag-cyfr "." [ ciag-cyfr ] [ ("e"|"E") mnoznik-skalujacy ]
  | "."  ciag-cyfr  [ ("e"|"E") mnoznik-skalujacy ]
  | ciag-cyfr ("e"|"E") mnoznik-skalujacy ) [ "f" | "F" | "l" | "L" ] .

mnoznik-skalujacy = [ znak-liczby ] ciag-cyfr .
```

```
literal_znakowy = "'" ( znak-bez-\ | sekwencja-specjalna ) "'" .

znak-bez-\ =  dowolny znak za wyjątkiem znaku „\” .

sekwencja-specjalna = "\\" | "\'" | "\"" | "\?" | "\a" | "\b"
                    | "\f" | "\n" | "\r" | "\t" | "\v"
    | "\" (( cyfra-osemkowa [ cyfra-osemkowa ] [ cyfra-osemkowa ] )
          | ciag-cyfr-szesnastkowych ) .

cyfra-osemkowa = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" .
```

Kategorie składniowe C – literały cd.

```
literal-zmiennopozycyjny =
          [ znak-liczby ] liczba-rzeczywista-bez-znaku .

liczba-rzeczywista-bez-znaku =
  ( ciag-cyfr "." [ ciag-cyfr ] [ ("e"|"E") mnoznik-skalujacy ]
  | "."  ciag-cyfr  [ ("e"|"E") mnoznik-skalujacy ]
  | ciag-cyfr ("e"|"E") mnoznik-skalujacy ) [ "f" | "F" | "l" | "L" ] .

mnoznik-skalujacy = [ znak-liczby ] ciag-cyfr .
```

```
literal_znakowy = "'" ( znak-bez-\ | sekwencja-specjalna ) "'" .

znak-bez-\ =  dowolny znak za wyjątkiem znaku „\” .

sekwencja-specjalna = "\\" | "\'" | "\"" | "\?" | "\a" | "\b"
                    | "\f" | "\n" | "\r" | "\t" | "\v"
    | "\" (( cyfra-osemkowa [ cyfra-osemkowa ] [ cyfra-osemkowa ] )
          | ciag-cyfr-szesnastkowych ) .

cyfra-osemkowa = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" .
```

Literał znakowy jest liczbą całkowitą.

Kategorie składniowe C – literały cd.

`literal-tekstowy` – analogiczny do kategorii `literal-znakowy`, tyle że złożony z zera lub więcej znaków i ograniczony znakami cudzysłowu (”) w miejsce apostrofów (’).

Przykłady:

```
"Jestem napisem"
```

Kategorie składniowe C – literały cd.

`literal-tekstowy` – analogiczny do kategorii `literal-znakowy`, tyle że złożony z zera lub więcej znaków i ograniczony znakami cudzysłowu (") w miejsce apostrofów (').

Przykłady:

```
"Jestem napisem"
""      /* napis pusty */
```

Kategorie składniowe C – literały cd.

`literal-tekstowy` – analogiczny do kategorii `literal-znakowy`, tyle że złożony z zera lub więcej znaków i ograniczony znakami cudzysłowu (") w miejsce apostrofów (').

Przykłady:

```
"Jestem napisem"
""      /* napis pusty */
"a"     /* napis zawierajacy jeden znak */
```

Kategorie składniowe C – literały cd.

`literal-tekstowy` – analogiczny do kategorii `literal-znakowy`, tyle że złożony z zera lub więcej znaków i ograniczony znakami cudzysłowu (") w miejsce apostrofów (').

Przykłady:

```
"Jestem napisem"
""      /* napis pusty */
"a"     /* napis zawierajacy jeden znak */
'a'     /* literal znakowy */
```

Kategorie składniowe C – literały cd.

`literal-tekstowy` – analogiczny do kategorii `literal-znakowy`, tyle że złożony z zera lub więcej znaków i ograniczony znakami cudzysłowu (") w miejsce apostrofów (').

Przykłady:

```
"Jestem napisem"
""      /* napis pusty */
"a"     /* napis zawierajacy jeden znak */
'a'     /* literal znakowy */
```

`"Oto jestem" " swiecie"` jest równoznaczne z `"Oto jestem swiecie"`

Kategorie składniowe C – literały cd.

`literal-tekstowy` – analogiczny do kategorii `literal-znakowy`, tyle że złożony z zera lub więcej znaków i ograniczony znakami cudzysłowu (") w miejsce apostrofów (').

Przykłady:

```
"Jestem napisem"
""      /* napis pusty */
"a"     /* napis zawierajacy jeden znak */
'a'     /* literal znakowy */
```

`"Oto jestem" " swiecie"` jest równoznaczne z `"Oto jestem swiecie"`

Reprezentacja:

Literał tekstowy jest tablicą, której elementami są znaki, zawierającą na końcu dodatkowy element `\0`.

Typy i rozmiary danych

- Podstawowe typy danych (arytmetyczne)
  - typy całkowite
    - `char` — typ znakowy (jeden bajt)

Typy i rozmiary danych

- Podstawowe typy danych (arytmetyczne)
  - typy całkowite
    - `char` — typ znakowy (jeden bajt)
    - `int` — typ stałopozycyjny (co najmniej 16 bitów)

Typy i rozmiary danych

- Podstawowe typy danych (arytmetyczne)
  - typy całkowite
    - `char` — typ znakowy (jeden bajt)
    - `int` — typ stałopozycyjny (co najmniej 16 bitów)
  - typy zmiennopozycyjne
    - `float` — pojedynczej precyzji

Typy i rozmiary danych

- Podstawowe typy danych (arytmetyczne)
  - typy całkowite
    - `char` — typ znakowy (jeden bajt)
    - `int` — typ stałopozycyjny (co najmniej 16 bitów)
  - typy zmiennopozycyjne
    - `float` — pojedynczej precyzji
    - `double` — podwójnej precyzji

Typy i rozmiary danych

- Podstawowe typy danych (arytmetyczne)
  - ⋆ typy całkowite
    - ∗ `char` — typ znakowy (jeden bajt)
    - ∗ `int` — typ stałopozycyjny (co najmniej 16 bitów)
  - ⋆ typy zmiennopozycyjne
    - ∗ `float` — pojedynczej precyzji
    - ∗ `double` — podwójnej precyzji
- Kwalifikatory typów podstawowych
  - ⋆ `short`, `long` — odnoszą się do obiektów stałopozycyjnych
  - ⋆ `signed`, `unsigned` — odnoszą się do obiektów całkowitych
  - ⋆ `long` — odnosi się do obiektów `double`

Typy i rozmiary danych

- Podstawowe typy danych (arytmetyczne)
  - ⋆ typy całkowite
    - ∗ `char` — typ znakowy (jeden bajt)
    - ∗ `int` — typ stałopozycyjny (co najmniej 16 bitów)
  - ⋆ typy zmiennopozycyjne
    - ∗ `float` — pojedynczej precyzji
    - ∗ `double` — podwójnej precyzji
- Kwalifikatory typów podstawowych
  - ⋆ `short`, `long` — odnoszą się do obiektów stałopozycyjnych
  - ⋆ `signed`, `unsigned` — odnoszą się do obiektów całkowitych
  - ⋆ `long` — odnosi się do obiektów `double`
- Podstawowe typy pochodne
  - ⋆ tablice obiektów

Typy i rozmiary danych

- Podstawowe typy danych (arytmetyczne)
  - ⋆ typy całkowite
    - ∗ `char` — typ znakowy (jeden bajt)
    - ∗ `int` — typ stałopozycyjny (co najmniej 16 bitów)
  - ⋆ typy zmiennopozycyjne
    - ∗ `float` — pojedynczej precyzji
    - ∗ `double` — podwójnej precyzji
- Kwalifikatory typów podstawowych
  - ⋆ `short`, `long` — odnoszą się do obiektów stałopozycyjnych
  - ⋆ `signed`, `unsigned` — odnoszą się do obiektów całkowitych
  - ⋆ `long` — odnosi się do obiektów `double`
- Podstawowe typy pochodne
  - ⋆ tablice obiektów
  - ⋆ funkcje, których rezultatami są obiekty

Typy i rozmiary danych

- Podstawowe typy danych (arytmetyczne)
  - typy całkowite
    - `char` — typ znakowy (jeden bajt)
    - `int` — typ stałopozycyjny (co najmniej 16 bitów)
  - typy zmiennopozycyjne
    - `float` — pojedynczej precyzji
    - `double` — podwójnej precyzji
- Kwalifikatory typów podstawowych
  - `short`, `long` — odnoszą się do obiektów stałopozycyjnych
  - `signed`, `unsigned` — odnoszą się do obiektów całkowitych
  - `long` — odnosi się do obiektów `double`
- Podstawowe typy pochodne
  - tablice obiektów
  - funkcje, których rezultatami są obiekty
  - wskazania na obiekty
  - struktury i unie obiektów

Typy i rozmiary danych cd.

- Relacje między wielkościami typów całkowitych
  - ⋆ `short` i `int` są co najmniej 16-bitowe
  - ⋆ `long` jest co najmniej 32-bitowy

Typy i rozmiary danych cd.

- Relacje między wielkościami typów całkowitych
  - `short` i `int` są co najmniej 16-bitowe
  - `long` jest co najmniej 32-bitowy
  - `short` nie może być dłuższy niż `int`
  - `int` nie może być dłuższy niż `long`

Typy i rozmiary danych cd.

- Relacje między wielkościami typów całkowitych
  - `short` i `int` są co najmniej 16-bitowe
  - `long` jest co najmniej 32-bitowy
  - `short` nie może być dłuższy niż `int`
  - `int` nie może być dłuższy niż `long`
  - Kwalifikatory `signed`, `unsigned` nie zmieniają rozmiaru typu a jedynie zakres wartości

Typy i rozmiary danych cd.

- Relacje między wielkościami typów całkowitych
  - `short` i `int` są co najmniej 16-bitowe
  - `long` jest co najmniej 32-bitowy
  - `short` nie może być dłuższy niż `int`
  - `int` nie może być dłuższy niż `long`
  - Kwalifikatory `signed`, `unsigned` nie zmieniają rozmiaru typu a jedynie zakres wartości
- Stałe określające dla danego komputera i kompilatora odpowiednie rozmiary zdefiniowane są w standardowych plikach nagłówkowych `limits.h` i `float.h`, przykładowo:

Typy i rozmiary danych cd.

- Relacje między wielkościami typów całkowitych
  - `short` i `int` są co najmniej 16-bitowe
  - `long` jest co najmniej 32-bitowy
  - `short` nie może być dłuższy niż `int`
  - `int` nie może być dłuższy niż `long`
  - Kwalifikatory `signed`, `unsigned` nie zmieniają rozmiaru typu a jedynie zakres wartości
- Stałe określające dla danego komputera i kompilatora odpowiednie rozmiary zdefiniowane są w standardowych plikach nagłówkowych `limits.h` i `float.h`, przykładowo:
  - ```
    #define SHRT_MIN    (-32768)  /* min value of a "short int" */
    #define SHRT_MAX    32767     /* max value of a "short int" */
    #define LONG_MAX    9223372036854775807L
    ```
    ```

Typy i rozmiary danych cd.

- Relacje między wielkościami typów całkowitych
  - `short` i `int` są co najmniej 16-bitowe
  - `long` jest co najmniej 32-bitowy
  - `short` nie może być dłuższy niż `int`
  - `int` nie może być dłuższy niż `long`
  - Kwalifikatory `signed`, `unsigned` nie zmieniają rozmiaru typu a jedynie zakres wartości
- Stałe określające dla danego komputera i kompilatora odpowiednie rozmiary zdefiniowane są w standardowych plikach nagłówkowych `limits.h` i `float.h`, przykładowo:
  - ```
    #define SHRT_MIN    (-32768)  /* min value of a "short int" */
    #define SHRT_MAX    32767     /* max value of a "short int" */
    #define LONG_MAX    9223372036854775807L
    #define DBL_MIN     2.22507385850720138309023E-308
    #define DBL_MAX     1.79769313486231570814527E+308
    ```
    ```

Zmienne

Zmienne posiadają:

a) nazwę, która musi być poprawnym identyfikatorem i różnić się od słów kluczowych języka C,

Zmienne

Zmienne posiadają:

a) nazwę, która musi być poprawnym identyfikatorem i różnić się od słów kluczowych języka C,

b) typ, który określa, jakie informacje będą przechowywane w zmiennej; nazwa i typ zmiennej są wymienione w jej deklaracji,

Zmienne

Zmienne posiadają:

a) nazwę, która musi być poprawnym identyfikatorem i różnić się od słów kluczowych języka C,

b) typ, który określa, jakie informacje będą przechowywane w zmiennej; nazwa i typ zmiennej są wymienione w jej deklaracji,

c) aktualną wartość,

Zmienne

Zmienne posiadają:

a) nazwę, która musi być poprawnym identyfikatorem i różnić się od słów kluczowych języka C,

b) typ, który określa, jakie informacje będą przechowywane w zmiennej; nazwa i typ zmiennej są wymienione w jej deklaracji,

c) aktualną wartość,

d) alokację, która jest miejscem w pamięci, gdzie ma być przechowywana wartość zmiennej,

Zmienne

Zmienne posiadają:

a) nazwę, która musi być poprawnym identyfikatorem i różnić się od słów kluczowych języka C,

b) typ, który określa, jakie informacje będą przechowywane w zmiennej; nazwa i typ zmiennej są wymienione w jej deklaracji,

c) aktualną wartość,

d) alokację, która jest miejscem w pamięci, gdzie ma być przechowywana wartość zmiennej,

e) zakres, który jest miejscem w programie, gdzie można odwoływać się do zmiennej,

Zmienne

Zmienne posiadają:

a) nazwę, która musi być poprawnym identyfikatorem i różnić się od słów kluczowych języka C,

b) typ, który określa, jakie informacje będą przechowywane w zmiennej; nazwa i typ zmiennej są wymienione w jej deklaracji,

c) aktualną wartość,

d) alokację, która jest miejscem w pamięci, gdzie ma być przechowywana wartość zmiennej,

e) zakres, który jest miejscem w programie, gdzie można odwoływać się do zmiennej,

f) czas trwania, to jest czas, w jakim mogą wystąpić odwołania do zmiennej.

Zmienne

Zmienne posiadają:

a) nazwę, która musi być poprawnym identyfikatorem i różnić się od słów kluczowych języka C,

b) typ, który określa, jakie informacje będą przechowywane w zmiennej; nazwa i typ zmiennej są wymienione w jej deklaracji,

c) aktualną wartość,

d) alokację, która jest miejscem w pamięci, gdzie ma być przechowywana wartość zmiennej,

e) zakres, który jest miejscem w programie, gdzie można odwoływać się do zmiennej,

f) czas trwania, to jest czas, w jakim mogą wystąpić odwołania do zmiennej.

---

a), **b)** — określone treścią programu,
c), **d)** — ustalone chwilowo w trakcie wykonywania programu,
e), **f)** — określone w C kwalifikatorami.

Klasy zmiennych

W języku C zdefiniowane są 4 klasy zmiennych:

- *zmienne automatyczne* — tworzone podczas wykonywania prologu bloku, w ktorym zostały zadeklarowane i usuwane podczas wykonywania jego epilogu,

Klasy zmiennych

W języku C zdefiniowane są 4 klasy zmiennych:

- *zmienne automatyczne* — tworzone podczas wykonywania prologu bloku, w ktorym zostały zadeklarowane i usuwane podczas wykonywania jego epilogu,
- *zmienne statyczne* —tworzone podczas wykonywania prologu programu i usuwane podczas wykonywania jego epilogu,

Klasy zmiennych

W języku C zdefiniowane są 4 klasy zmiennych:

- *zmienne automatyczne* — tworzone podczas wykonywania prologu bloku, w ktorym zostały zadeklarowane i usuwane podczas wykonywania jego epilogu,
- *zmienne statyczne* —tworzone podczas wykonywania prologu programu i usuwane podczas wykonywania jego epilogu,
- *zmienne rejestrowe* — jak automatyczne, tyle że umieszczane w szybkich rejestrach procesora,

Klasy zmiennych

W języku C zdefiniowane są 4 klasy zmiennych:

- *zmienne automatyczne* — tworzone podczas wykonywania prologu bloku, w ktorym zostały zadeklarowane i usuwane podczas wykonywania jego epilogu,
- *zmienne statyczne* —tworzone podczas wykonywania prologu programu i usuwane podczas wykonywania jego epilogu,
- *zmienne rejestrowe* — jak automatyczne, tyle że umieszczane w szybkich rejestrach procesora,
- *zmienne zewnętrzne* — jak statyczne, tyle że deklarowane wielokrotnie w różnych blokach i sekcjach programu.

Deklaracja zmiennych

```
deklaracja =
   specyfikator-deklaracji [ inicjowana-lista-deklaratorow ] ";" .
```

Deklaracja zmiennych

```
deklaracja =
   specyfikator-deklaracji [ inicjowana-lista-deklaratorow ] ";" .

specyfikator-deklaracji =
      specyfikator-klasy-pamieci [ specyfikator-deklaracji ]
    | specyfikator-typu [ specyfikator-deklaracji ]
    | kwalifikator-typu [ specyfikator-deklaracji ] .
```

Deklaracja zmiennych

```
deklaracja =
   specyfikator-deklaracji [ inicjowana-lista-deklaratorow ] ";" .

specyfikator-deklaracji =
      specyfikator-klasy-pamieci [ specyfikator-deklaracji ]
    | specyfikator-typu [ specyfikator-deklaracji ]
    | kwalifikator-typu [ specyfikator-deklaracji ] .

specyfikator-klasy-pamieci = "auto"   | "static"  | "register"
                           | "extern" | "typedef" .
```

Deklaracja zmiennych

```
deklaracja =
   specyfikator-deklaracji [ inicjowana-lista-deklaratorow ] ";" .

specyfikator-deklaracji =
      specyfikator-klasy-pamieci [ specyfikator-deklaracji ]
    | specyfikator-typu [ specyfikator-deklaracji ]
    | kwalifikator-typu [ specyfikator-deklaracji ] .

specyfikator-klasy-pamieci = "auto"   | "static"  | "register"
                           | "extern" | "typedef" .

specyfikator-typu = "void" | "char" | "short" | "int" | "long"
                  | "float" | "double" | "signed" | "unsigned" | ...
```

Deklaracja zmiennych

```
deklaracja =
   specyfikator-deklaracji [ inicjowana-lista-deklaratorow ] ";" .

specyfikator-deklaracji =
      specyfikator-klasy-pamieci [ specyfikator-deklaracji ]
    | specyfikator-typu [ specyfikator-deklaracji ]
    | kwalifikator-typu [ specyfikator-deklaracji ] .

specyfikator-klasy-pamieci = "auto"   | "static"  | "register"
                           | "extern" | "typedef" .

specyfikator-typu = "void" | "char" | "short" | "int" | "long"
                  | "float" | "double" | "signed" | "unsigned" | ...

inicjowana-lista-deklaratorow =
    inicjowany-deklarator { "," inicjowany deklarator } .
```

Deklaracja zmiennych

```
deklaracja =
   specyfikator-deklaracji [ inicjowana-lista-deklaratorow ] ";" .

specyfikator-deklaracji =
      specyfikator-klasy-pamieci [ specyfikator-deklaracji ]
    | specyfikator-typu [ specyfikator-deklaracji ]
    | kwalifikator-typu [ specyfikator-deklaracji ] .

specyfikator-klasy-pamieci = "auto"   | "static"  | "register"
                           | "extern" | "typedef" .

specyfikator-typu = "void" | "char" | "short" | "int" | "long"
                  | "float" | "double" | "signed" | "unsigned" | ...

inicjowana-lista-deklaratorow =
    inicjowany-deklarator { "," inicjowany deklarator } .

inicjowany-deklarator = identyfikator { "=" wyrazenie-przypisania }
                      | ...
```

Deklaracja zmiennych – przykłady

```
int delta;
```

Deklaracja zmiennych – przykłady

```
int delta;
int a, b, c;
```

Deklaracja zmiennych – przykłady

```
int delta;
int a, b, c;
int wyr = 3;
int zm1 = 1, zm2 = 2, zm3 = 3;
```


Deklaracja zmiennych – przykłady

```
int delta;
int a, b, c;
int wyr = 3;
int zm1 = 1, zm2 = 2, zm3 = 3;
unsigned int bez_znaku;
```

Deklaracja zmiennych – przykłady

```
int delta;
int a, b, c;
int wyr = 3;
int zm1 = 1, zm2 = 2, zm3 = 3;
unsigned int bez_znaku;
char znak1 = 'J';
char znak2 = 74;
```

Deklaracja zmiennych – przykłady

```
int delta;
int a, b, c;
int wyr = 3;
int zm1 = 1, zm2 = 2, zm3 = 3;
unsigned int bez_znaku;
char znak1 = 'J';
char znak2 = 74;
char esc = '\\';
int limit = MAX + 1;   /* przy #define MAX 10 */
```

Deklaracja zmiennych – przykłady

```
int delta;
int a, b, c;
int wyr = 3;
int zm1 = 1, zm2 = 2, zm3 = 3;
unsigned int bez_znaku;
char znak1 = 'J';
char znak2 = 74;
char esc = '\\';
int limit = MAX + 1;   /* przy #define MAX 10 */
```

Porównaj

```
int x = 1, y = 2;
int x = y = 1;
```

Stałe

Wszystkie elementy, których wartość może zostać wyliczona na etapie kompilacji programu są stałymi. W programie mogą one występować jako

- stałe jawne

Stałe

Wszystkie elementy, których wartość może zostać wyliczona na etapie kompilacji programu są stałymi. W programie mogą one występować jako

- stałe jawne

  200

Stałe

Wszystkie elementy, których wartość może zostać wyliczona na etapie kompilacji programu są stałymi. W programie mogą one występować jako

- stałe jawne

```
200
'a'
"a"
"albo tez wiecej literek aaaaa"
```

Stałe

Wszystkie elementy, których wartość może zostać wyliczona na etapie kompilacji programu są stałymi. W programie mogą one występować jako

- stałe jawne

```
200
'a'
"a"
"albo tez wiecej literek aaaaa"
```

- stałe symboliczne

```
#define LOW  0      /* dolna granica temperatur */
#define UP   300    /* gorna granica temperatur */
#define STEP 20     /* rozmiar kroku */
```

Stałe

Wszystkie elementy, których wartość może zostać wyliczona na etapie kompilacji programu są stałymi. W programie mogą one występować jako

- stałe jawne

```
200
'a'
"a"
"albo tez wiecej literek aaaaa"
```

- stałe symboliczne

```
#define LOW  0      /* dolna granica temperatur */
#define UP   300    /* gorna granica temperatur */
#define STEP 20     /* rozmiar kroku */
#define NAME "Ala"  /* szanownej malzonki */
```

Stałe

Wszystkie elementy, których wartość może zostać wyliczona na etapie kompilacji programu są stałymi. W programie mogą one występować jako

- stałe jawne

```
200
'a'
"a"
"albo tez wiecej literek aaaaa"
```

- stałe symboliczne

```
#define LOW  0      /* dolna granica temperatur */
#define UP   300    /* gorna granica temperatur */
#define STEP 20     /* rozmiar kroku */
#define NAME "Ala"  /* szanownej malzonki */
#define BELL '\x7'  /* ASCII: znak alarmu */
                /* NAZWY SYMBOLICZNE ZWYCZAJOWO ZAPISUJEMY */
                /* WIELKIMI LITERAMI ALFABETU              */
```

Wyrażenia

- stałe, zmienne, wywołania funkcji, wyrażenia operatorowe

Wyrażenia

- stałe, zmienne, wywołania funkcji, wyrażenia operatorowe
- `3.14`, `PI`, `delta`, `sqrt(delta)`, `3+4`, `(-b-sqrt(delta))/(2*a)`

Wyrażenia

- stałe, zmienne, wywołania funkcji, wyrażenia operatorowe
- `3.14`, `PI`, `delta`, `sqrt(delta)`, `3+4`, `(-b-sqrt(delta))/(2*a)`

Reguły wyliczania wyrażeń

a) stałe jawne: ich wartość jest im równa,

Wyrażenia

- stałe, zmienne, wywołania funkcji, wyrażenia operatorowe
- `3.14`, `PI`, `delta`, `sqrt(delta)`, `3+4`, `(-b-sqrt(delta))/(2*a)`

Reguły wyliczania wyrażeń

a) **stałe jawne:** ich wartość jest im równa,

b) **stałe symboliczne i zmienne:** ich wartość jest im przypisana, przy czym dla zmiennych może ulegać zmianie w trakcie pracy programu,

Wyrażenia

- stałe, zmienne, wywołania funkcji, wyrażenia operatorowe
- `3.14`, `PI`, `delta`, `sqrt(delta)`, `3+4`, `(-b-sqrt(delta))/(2*a)`

Reguły wyliczania wyrażeń

a) **stałe jawne:** ich wartość jest im równa,

b) **stałe symboliczne i zmienne:** ich wartość jest im przypisana, przy czym dla zmiennych może ulegać zmianie w trakcie pracy programu,

c) **wywołania funkcji i wyrażenia operatorowe:** wpierw wyliczane są wartości argumentów (które same są wyrażeniami), a następnie operator lub funkcja wylicza swoją wartość.

Wyrażenia

- stałe, zmienne, wywołania funkcji, wyrażenia operatorowe
- `3.14`, `PI`, `delta`, `sqrt(delta)`, `3+4`, `(-b-sqrt(delta))/(2*a)`

Reguły wyliczania wyrażeń

a) **stałe jawne:** ich wartość jest im równa,

b) **stałe symboliczne i zmienne:** ich wartość jest im przypisana, przy czym dla zmiennych może ulegać zmianie w trakcie pracy programu,

c) **wywołania funkcji i wyrażenia operatorowe:** wpierw wyliczane są wartości argumentów (które same są wyrażeniami), a następnie operator lub funkcja wylicza swoją wartość.

- Skąd wiemy jak je konstruować?
- Jak rozstrzygać które z nich są poprawne?
- A które maja sens?

Operatory

- Operatory podstawowe
  - przypisania — =

Operatory

- Operatory podstawowe
  - przypisania — =
  - porównania — ==

Operatory

- Operatory podstawowe
  - przypisania — `=`
  - porównania — `==`
- operatory arytmetyczne
  - dodawania — `+`
  - odejmowania — `-`
  - mnożenia — `*`
  - dzielenia — `/`, `%`

Operatory

- Operatory podstawowe
  - przypisania — `=`
  - porównania — `==`
- operatory arytmetyczne
  - dodawania — `+`
  - odejmowania — `-`
  - mnożenia — `*`
  - dzielenia — `/`, `%`
- operatory logiczne
  - alternatywy — `||`
  - koniunkcji — `&&`
  - negacji — `!`

Operatory

- Operatory podstawowe
  - przypisania — `=`
  - porównania — `==`
- operatory arytmetyczne
  - dodawania — `+`
  - odejmowania — `-`
  - mnożenia — `*`
  - dzielenia — `/`, `%`
- operatory logiczne
  - alternatywy — `||`
  - koniunkcji — `&&`
  - negacji — `!`
- pozostałe operatory przypisania — `+=`, `-=`, `*=`, `/=`, `%=`

Operatory

- Operatory podstawowe
  - przypisania — `=`
  - porównania — `==`
- operatory arytmetyczne
  - dodawania — `+`
  - odejmowania — `-`
  - mnożenia — `*`
  - dzielenia — `/`, `%`
- operatory logiczne
  - alternatywy — `||`
  - koniunkcji — `&&`
  - negacji — `!`
- pozostałe operatory przypisania — `+=`, `-=`, `*=`, `/=`, `%=`
- pozostałe operatory relacyjne — `<`, `<=`, `!=`, `>=`, `>`

Operatory — priorytety

Operatory	Łączność
`() [] -> .`	lewostronna
`! ~ ++ -- + - * & (`*typ*`) sizeof`	prawostronna
`* / %`	lewostronna
`+ -`	lewostronna
`<< >>`	lewostronna
`< <= > >=`	lewostronna
`== !=`	lewostronna
`&`	lewostronna
`^`	lewostronna
`|`	lewostronna
`&&`	lewostronna
`||`	lewostronna
`?:`	prawostronna
`= += -= *= /= %= ^= |= <<= >>=`	prawostronna
`,`	lewostronna

Jednoargumentowe operatory `+`, `-`, `*` oraz `&` mają priorytet wyższy niż ich odpowiedniki dwuargumentowe. Nie określa się kolejności wyliczania wartości argumentów operatora.

Operatory — kolejność wyliczania argumentów

- Priorytety i kierunek wiązania operatorów są w pełni określone.

Operatory — kolejność wyliczania argumentów

- Priorytety i kierunek wiązania operatorów są w pełni określone.
- Ale kolejność wyliczania operandów operatora jest dowolna!

```
wyn = fun1() + fun2();    /* ktora najpierw? fun1 czy fun2? */
```

Operatory — kolejność wyliczania argumentów

- Priorytety i kierunek wiązania operatorów są w pełni określone.
- Ale kolejność wyliczania operandów operatora jest dowolna!

```
wyn = fun1() + fun2();    /* ktora najpierw? fun1 czy fun2? */
```

- Poza kilkoma wyjątkami :)

```
wyn = fun1() || fun2(); /* to samo z && ale juz nie z | i & */
```

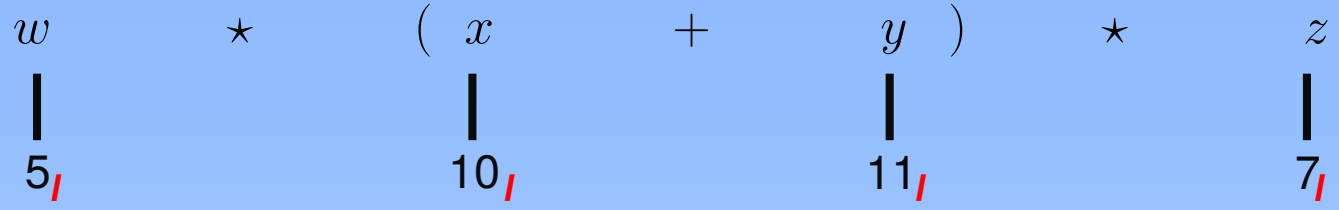
„Operator `||` gwarantuje obliczanie wartości operandów od lewej do prawej: najpierw obliczana jest wartość pierwszego operandu, łącznie z operacjami zmieniającymi wartości obiektów; jeżeli jego wartość jest różna od 0, wartość całego wyrażenia to 1. Tylko w innych przypadkach obliczana jest wartość operandu po prawej stronie i jeżeli jest różna od 0, całe wyrażenie ma wartość 1. W pozostałych przypadkach wyrażenie ma wartość 0.”

(dokumentacja wszystkich standardów języka C)

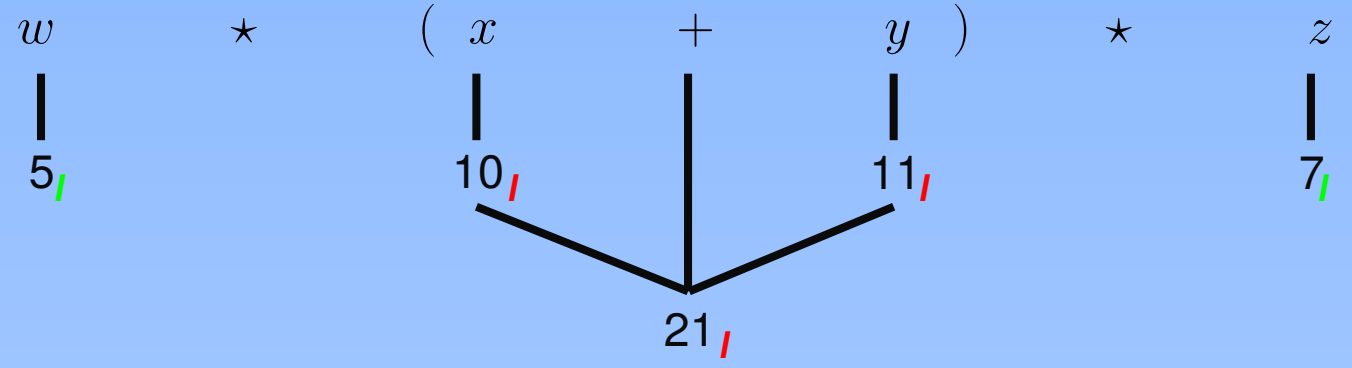
Drzewa wyliczania wartości wyrażeń

$$w \quad \star \quad (\quad x \quad + \quad y \quad) \quad \star \quad z$$

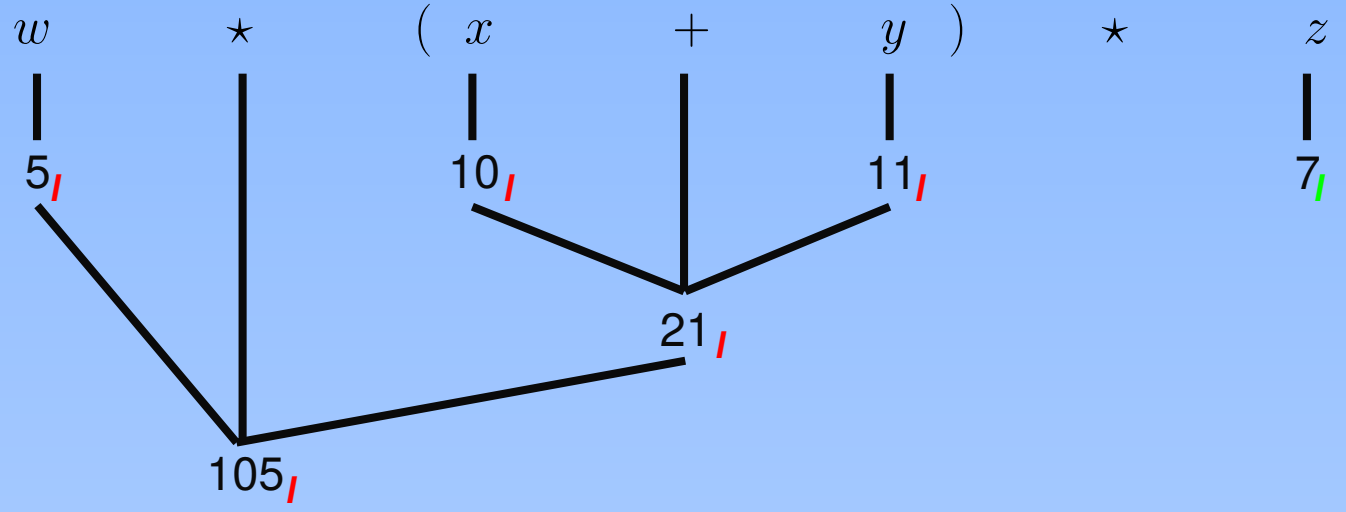
Drzewa wyliczania wartości wyrażeń



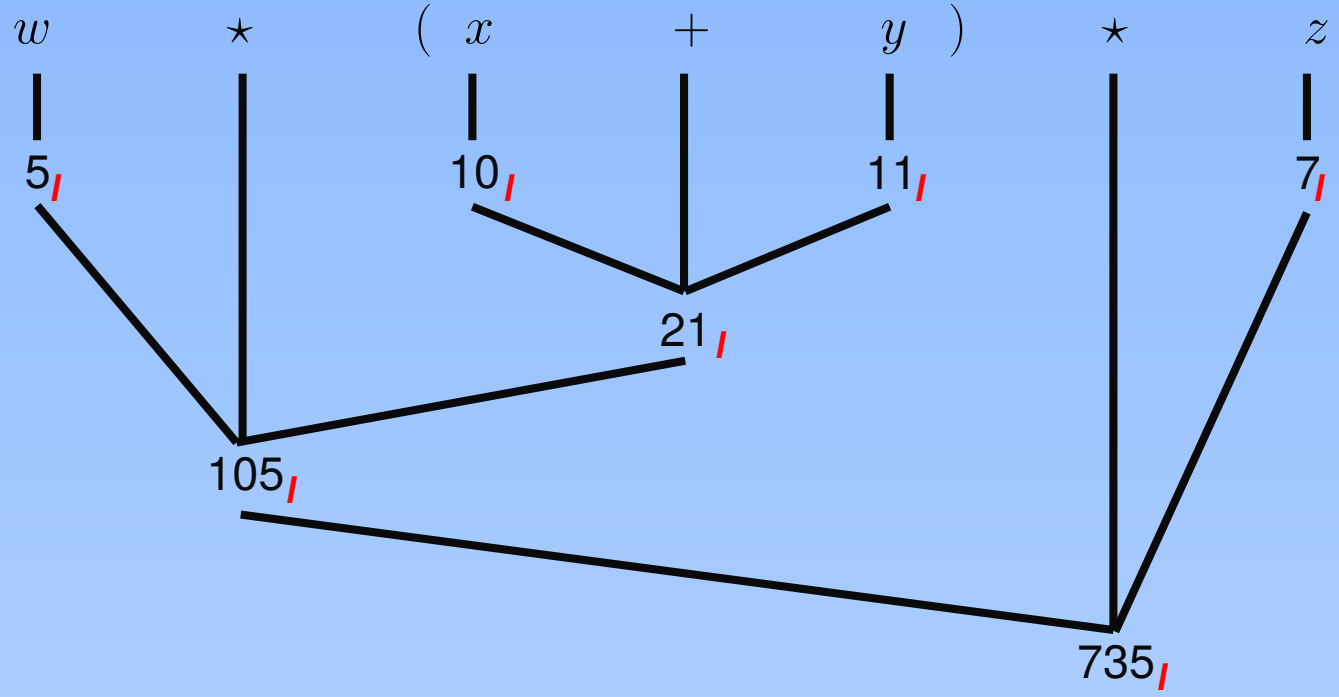
Drzewa wyliczania wartości wyrażeń



Drzewa wyliczania wartości wyrażeń



Drzewa wyliczania wartości wyrażeń



Drzewa wyliczania wartości wyrażeń

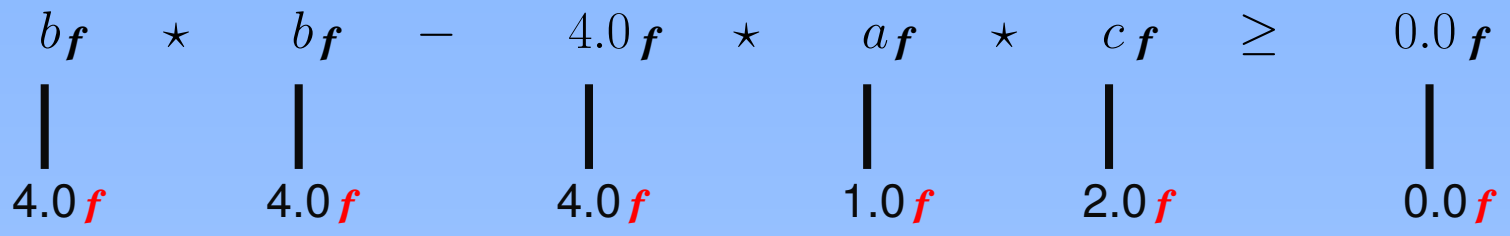
$$b_{\,f} \quad \star \quad b_{\,f} \quad - \quad 4.0_{\,f} \quad \star \quad a_{\,f} \quad \star \quad c_{\,f} \quad \geq \quad 0.0_{\,f}$$

(a=1.0)

(b=4.0)

(c=2.0)

Drzewa wyliczania wartości wyrażeń

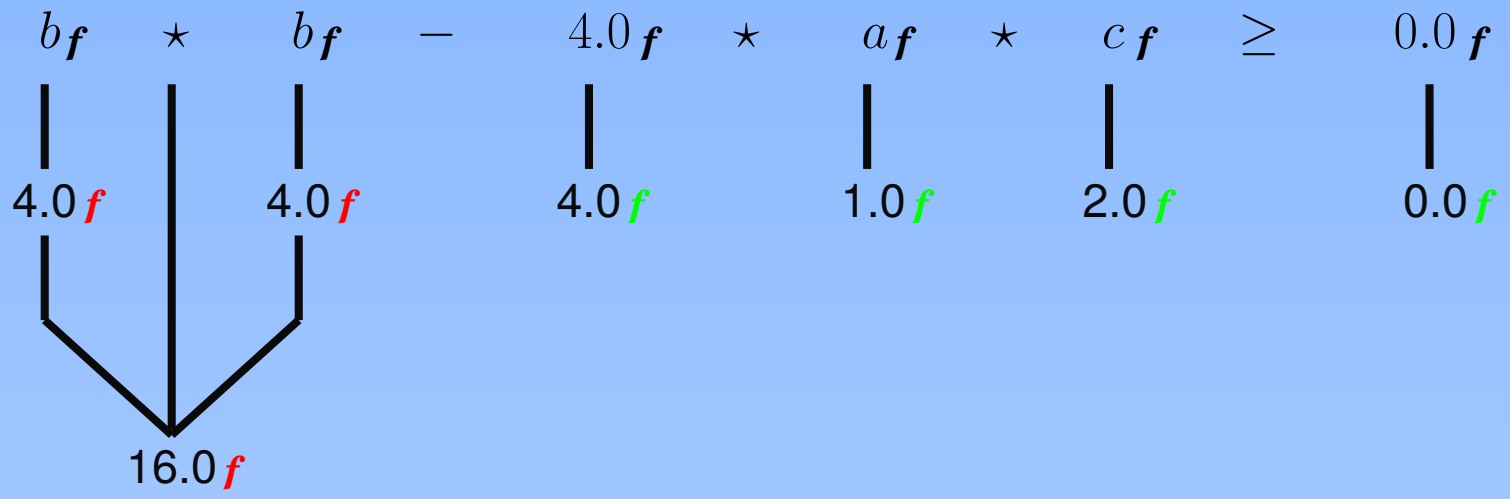


(a=1.0)

(b=4.0)

(c=2.0)

Drzewa wyliczania wartości wyrażeń

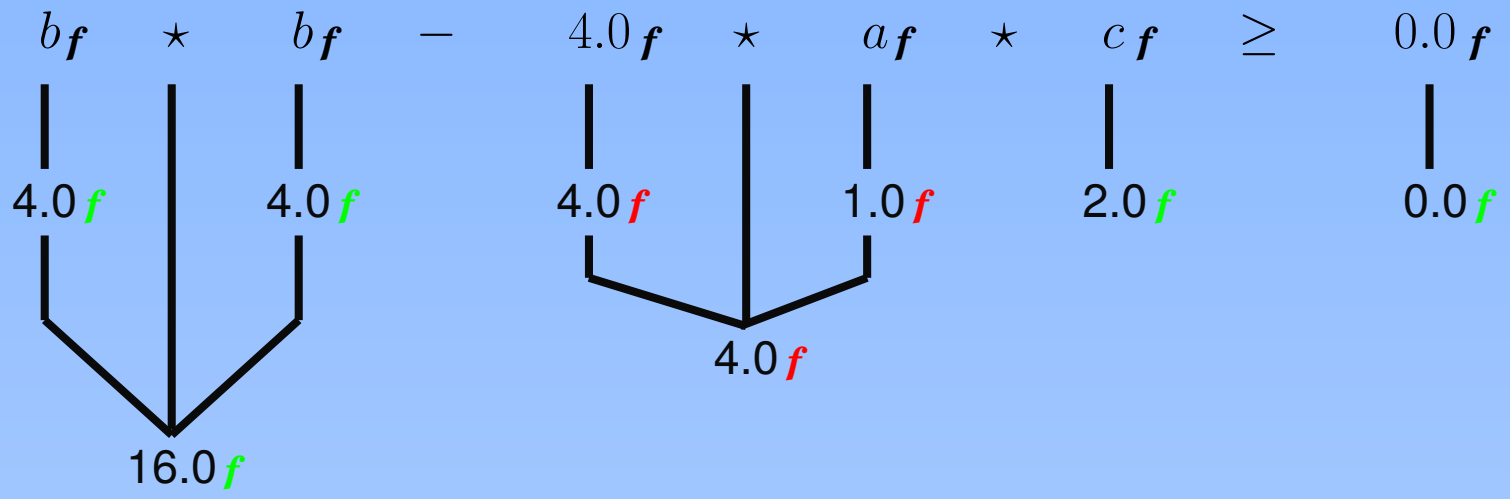


(a=1.0)

(b=4.0)

(c=2.0)

Drzewa wyliczania wartości wyrażeń

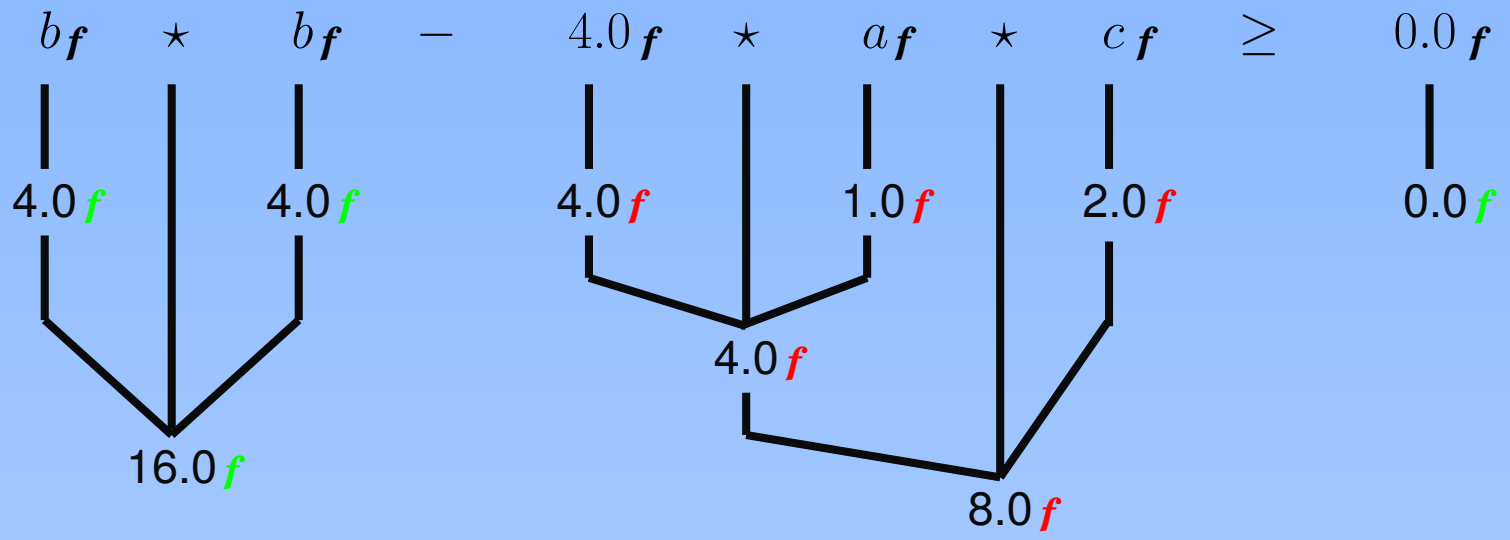


(a=1.0)

(b=4.0)

(c=2.0)

Drzewa wyliczania wartości wyrażeń

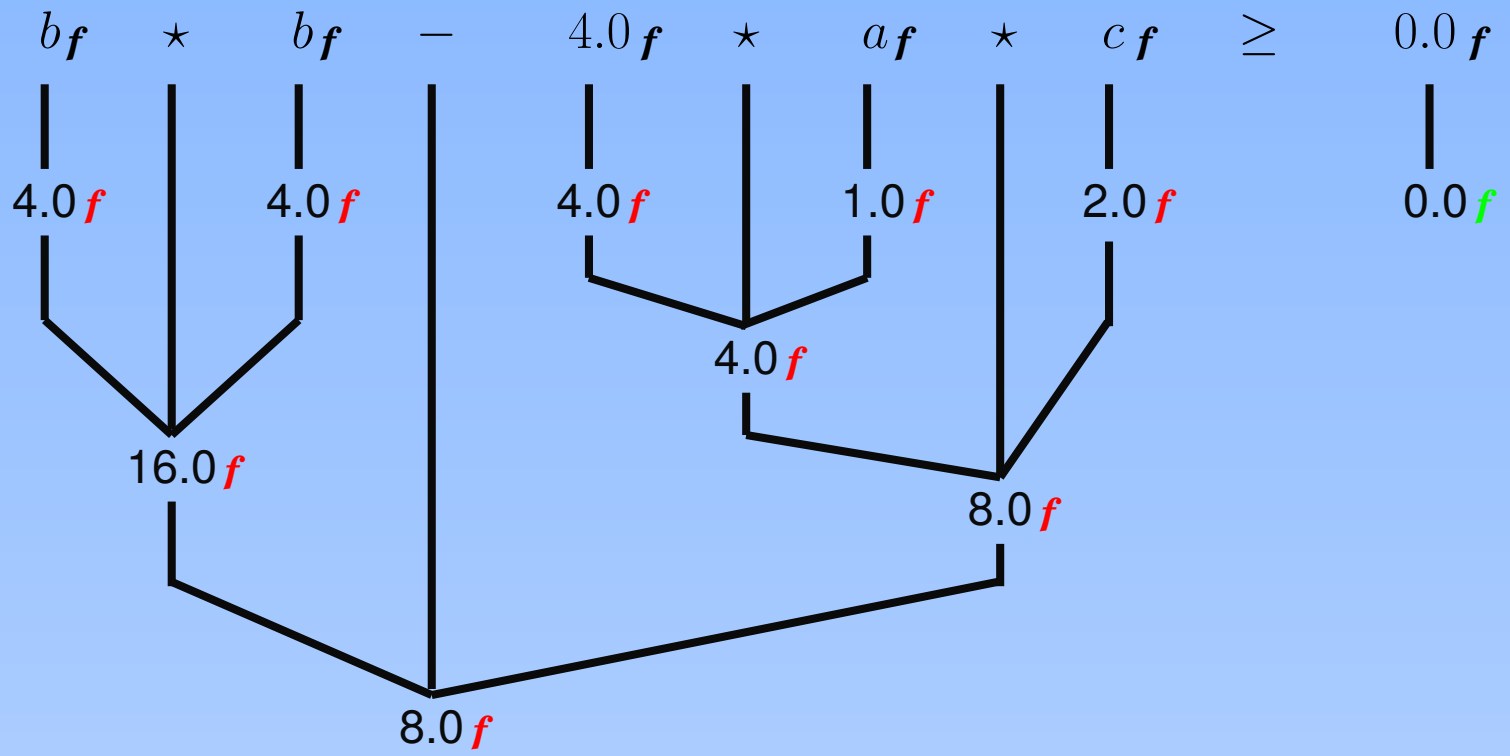


(a=1.0)

(b=4.0)

(c=2.0)

Drzewa wyliczania wartości wyrażeń

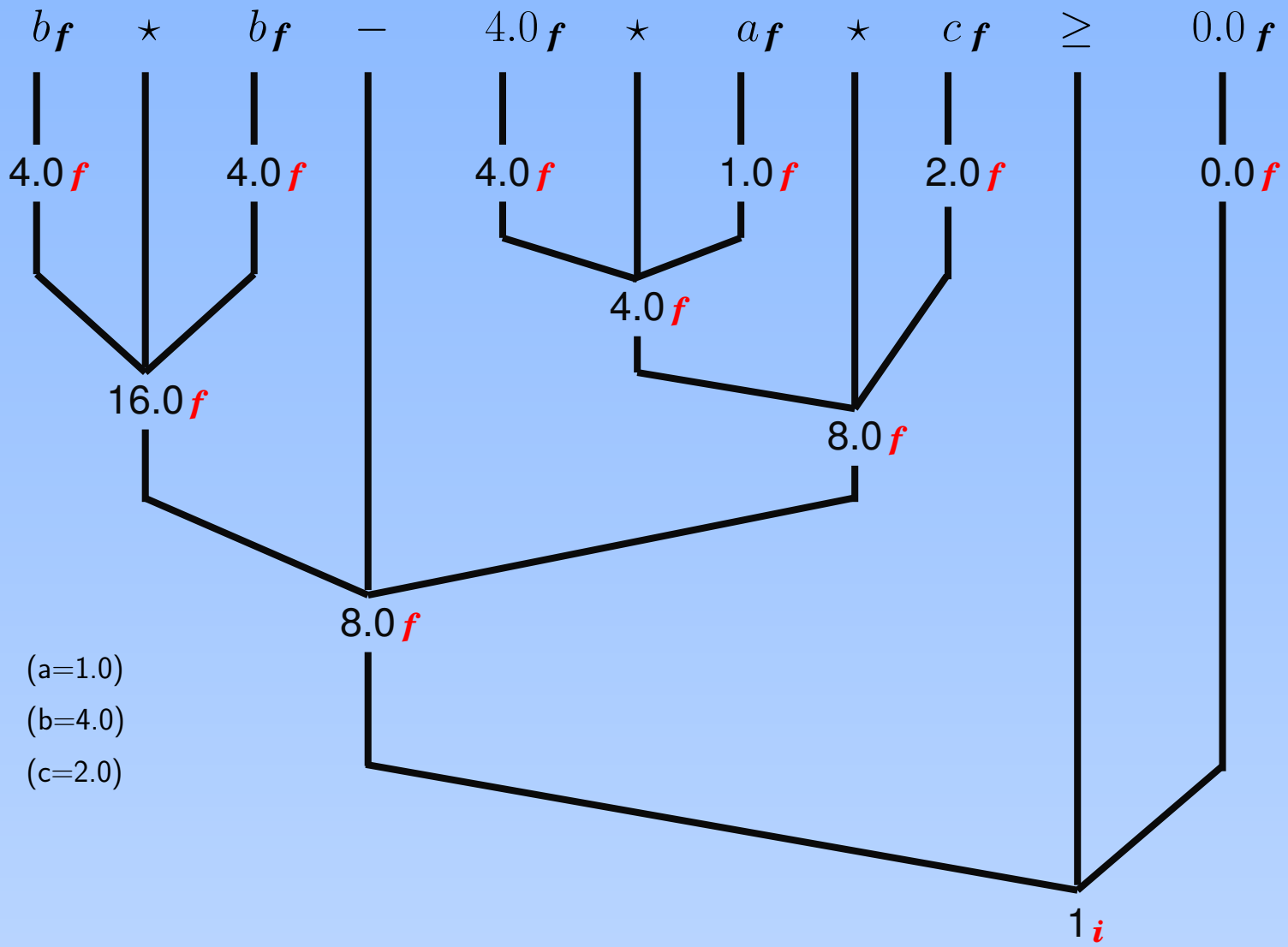


(a=1.0)

(b=4.0)

(c=2.0)

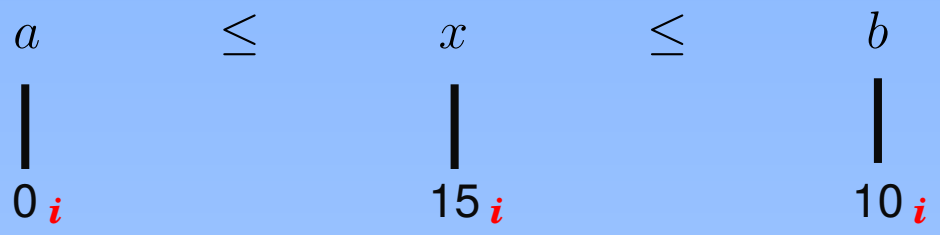
Drzewa wyliczania wartości wyrażeń



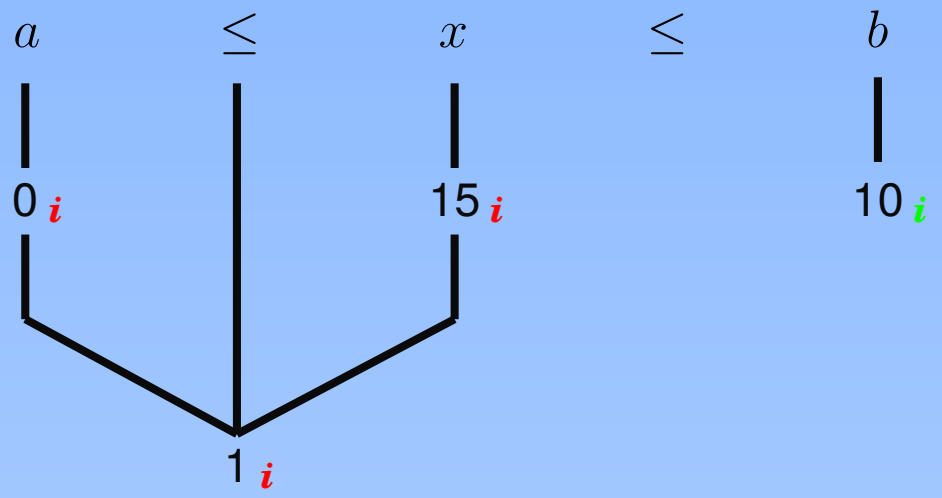
Drzewa wyliczania wartości wyrażeń

$$a \qquad \leq \qquad x \qquad \leq \qquad b$$

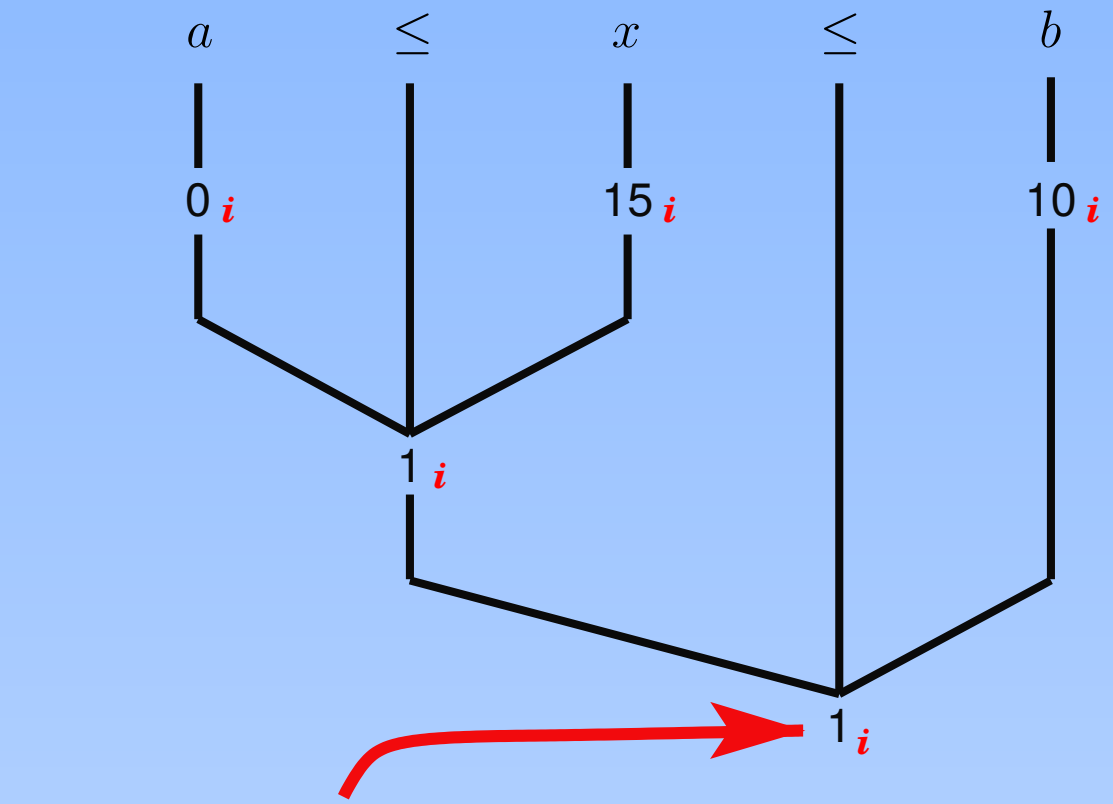
Drzewa wyliczania wartości wyrażeń



Drzewa wyliczania wartości wyrażeń



Drzewa wyliczania wartości wyrażeń



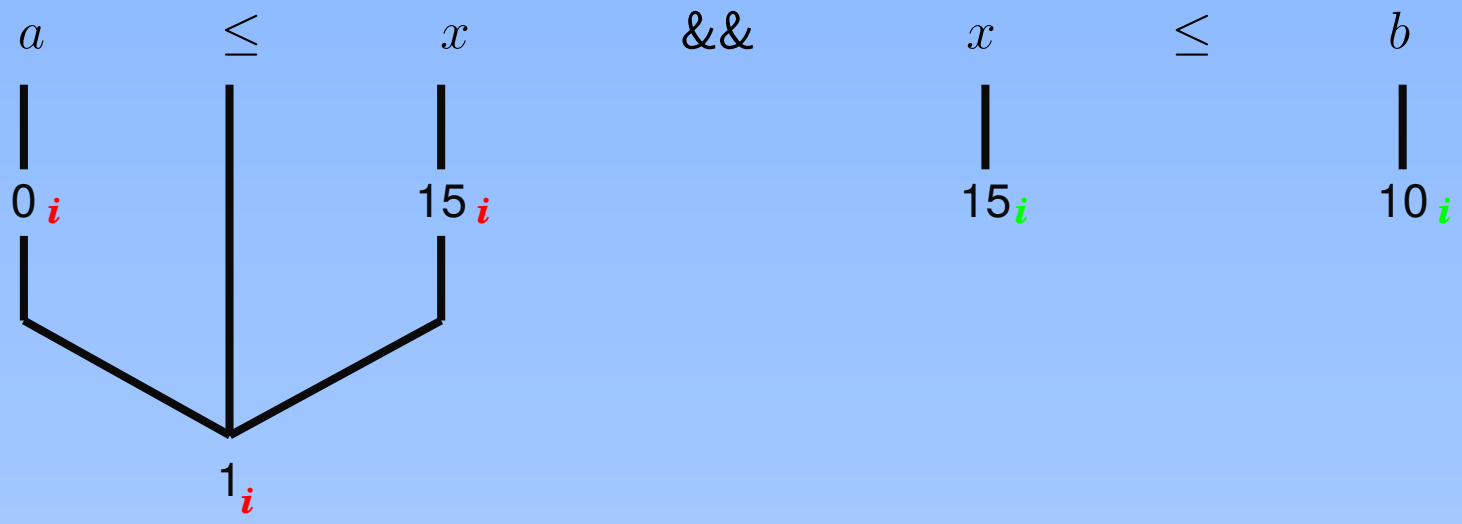
Niekoniecznie tak jak chcieliśmy :(

Drzewa wyliczania wartości wyrażeń

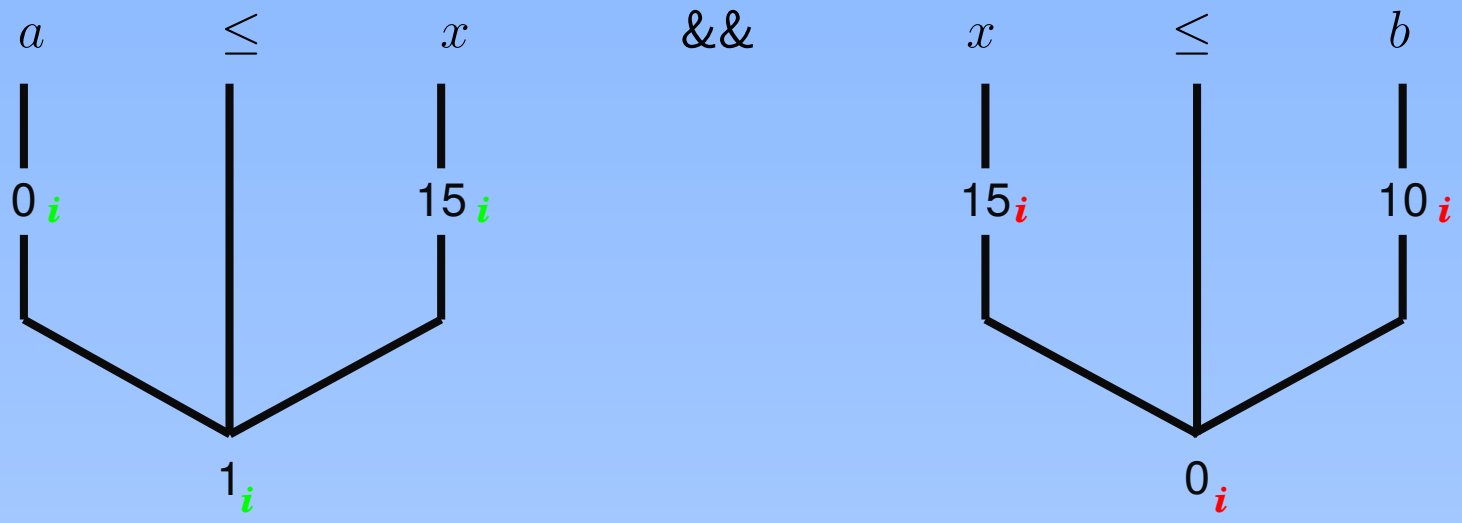
$$a \quad \leq \quad x \quad \&\& \quad x \quad \leq \quad b$$

Drzewa wyliczania wartości wyrażeń

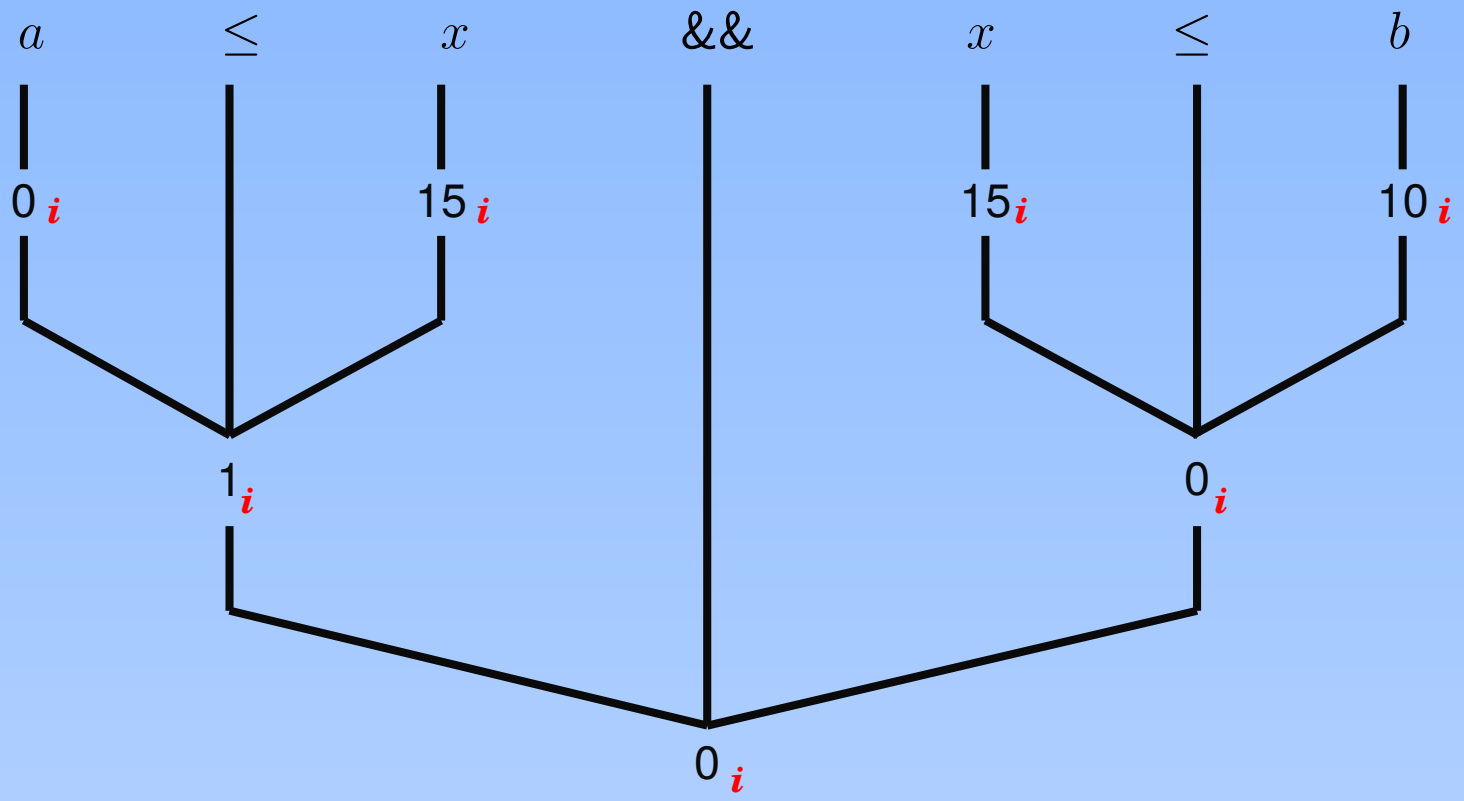
Drzewa wyliczania wartości wyrażeń



Drzewa wyliczania wartości wyrażeń



Drzewa wyliczania wartości wyrażeń



Teraz lepiej :)

Instrukcje

Instrukcja — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego. W języku C wyrażenie staję się instrukcją jeśli jest zakończone średnikiem.

Instrukcje

Instrukcja — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego. W języku C wyrażenie staję się instrukcją jeśli jest zakończone średnikiem.

Przykłady instrukcji

```
x = 0;
```

Instrukcje

Instrukcja — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego. W języku C wyrażenie staję się instrukcją jeśli jest zakończone średnikiem.

Przykłady instrukcji

```
x = 0;
i++;
```

Instrukcje

Instrukcja — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego. W języku C wyrażenie staję się instrukcją jeśli jest zakończone średnikiem.

Przykłady instrukcji

```
x = 0;
i++;
printf("Oto jest instrukcja");
```

Instrukcje

Instrukcja — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego. W języku C wyrażenie staję się instrukcją jeśli jest zakończone średnikiem.

Przykłady instrukcji

```
x = 0;
i++;
printf("Oto jest instrukcja");
if (wynik == 3)
  printf("Wynik poprawny\n");
else
{
  wynik = 7;
  printf("Wynik niepoprawny\n");
}
```

Instrukcje

Instrukcja — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego. W języku C wyrażenie staję się instrukcją jeśli jest zakończone średnikiem.

Przykłady instrukcji

```
x = 0;
i++;
printf("Oto jest instrukcja");
if (wynik == 3)
  printf("Wynik poprawny\n");
else
{
  wynik = 7;
  printf("Wynik niepoprawny\n");
}
```

Instrukcje

Instrukcja — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego. W języku C wyrażenie staję się instrukcją jeśli jest zakończone średnikiem.

Przykłady instrukcji

```
x = 0;
i++;
printf("Oto jest instrukcja");
if (wynik == 3)
  printf("Wynik poprawny\n");
else
{
  wynik = 7;
  printf("Wynik niepoprawny\n");
}
```

Instrukcje

Instrukcja — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego. W języku C wyrażenie staję się instrukcją jeśli jest zakończone średnikiem.

Przykłady instrukcji

```
x = 0;
i++;
printf("Oto jest instrukcja");
if (wynik == 3)
  printf("Wynik poprawny\n");
else
{
  wynik = 7;
  printf("Wynik niepoprawny\n");
}
```

Instrukcje

Instrukcja — konstrukcja języka opisująca akcje, które mają być wykonane podczas procesu obliczeniowego. W języku C wyrażenie staję się instrukcją jeśli jest zakończone średnikiem.

Przykłady instrukcji

```
x = 0;
i++;
printf("Oto jest instrukcja");
if (wynik == 3)
  printf("Wynik poprawny\n");
else
{
  wynik = 7;
  printf("Wynik niepoprawny\n");
}
```

Instrukcja złożona

```
instrukcja-zlozona = "{" { deklaracja } { instrukcja } "}" .
```

Instrukcja złożona

```
instrukcja-zlozona = "{" { deklaracja } { instrukcja } "}" .
```

Instrukcja złożona, czyli *blok*, tworzy ujęty w nawiasy klamrowe ciąg deklaracji i instrukcji równoważny składniowo pojedynczej instrukcji.

Instrukcja złożona

```
instrukcja-zlozona = "{" { deklaracja } { instrukcja } "}" .
```

Instrukcja złożona, czyli *blok*, tworzy ujęty w nawiasy klamrowe ciąg deklaracji i instrukcji równoważny składniowo pojedynczej instrukcji.

```
int main() {
  int a, b;

  printf("Program wskazuje ... Podaj wartosci ...");
  scanf("%d%d", &a, &b);
  if (a < b)
  {
    printf("Mniejsza wartoscia jest %d\n",a);
    printf("Czy nie jest to zadziwiajace\n");
  }
  else
  {
    printf("Mniejsza wartoscia jest %d\n",b);
    printf("Coz poczac:(\n");
  }
}
```

Instrukcja złożona

```
instrukcja-zlozona = "{" { deklaracja } { instrukcja } "}" .
```

Instrukcja złożona, czyli *blok*, tworzy ujęty w nawiasy klamrowe ciąg deklaracji i instrukcji równoważny składniowo pojedynczej instrukcji.

```
int main() {
  int a, b;

  printf("Program wskazuje ... Podaj wartosci ...");
  scanf("%d%d", &a, &b);
  if (a < b)
  {
    printf("Mniejsza wartoscia jest %d\n",a);
    printf("Czy nie jest to zadziwiajace\n");
  }
  else
  {
    printf("Mniejsza wartoscia jest %d\n",b);
    printf("Coz poczac:(\n");
  }
}
```

Instrukcja złożona

```
instrukcja-zlozona = "{" { deklaracja } { instrukcja } "}" .
```

Instrukcja złożona, czyli *blok*, tworzy ujęty w nawiasy klamrowe ciąg deklaracji i instrukcji równoważny składniowo pojedynczej instrukcji.

```
int main() {
  int a, b;

  printf("Program wskazuje ... Podaj wartosci ...");
  scanf("%d%d", &a, &b);
  if (a < b)
  {
    printf("Mniejsza wartoscia jest %d\n",a);
    printf("Czy nie jest to zadziwiajace\n");
  }
  else
  {
    printf("Mniejsza wartoscia jest %d\n",b);
    printf("Coz poczac:(\n");
  }
}
```

Instrukcja złożona

```
instrukcja-zlozona = "{" { deklaracja } { instrukcja } "}" .
```

Instrukcja złożona, czyli *blok*, tworzy ujęty w nawiasy klamrowe ciąg deklaracji i instrukcji równoważny składniowo pojedynczej instrukcji.

```
int main() {
  int a, b;

  printf("Program wskazuje ... Podaj wartosci ...");
  scanf("%d%d", &a, &b);
  if (a < b)
  {
    printf("Mniejsza wartoscia jest %d\n",a);
    printf("Czy nie jest to zadziwiajace\n");
  }
  else
  {
    printf("Mniejsza wartoscia jest %d\n",b);
    printf("Coz poczac:(\n");
  }
}
```

Instrukcja warunkowa `if-else`

```
instrukcja-if = "if (" wyrazenie ")" instrukcja
              | "if (" wyrazenie ")" instrukcja
                "else" instrukcja .
```

Instrukcja warunkowa `if-else`

```
instrukcja-if = "if (" wyrazenie ")" instrukcja
              | "if (" wyrazenie ")" instrukcja
                "else" instrukcja .
```

Porządkowanie liczb

```
if (x > y)
{
  wieksza = x;
  mniejsza = y;
}
else
{
  wieksza = y;
  mniejsza = x;
}
```

Instrukcja warunkowa `if-else`

```
instrukcja-if = "if (" wyrazenie ")" instrukcja
              | "if (" wyrazenie ")" instrukcja
                "else" instrukcja .
```

Porządkowanie liczb

```
if (x > y)
{
  wieksza = x;
  mniejsza = y;
}
else
{
  wieksza = y;
  mniejsza = x;
}
```

Można też tak ;-)

```
if(x>y){wieksza=x;mniejsza=y;}else
{wieksza=y;mniejsza=x;}
```

Czytelność kodu

Poprawnie skonstruowany kod w ANSII C?!

```
int putchar(int c);
int main(t,_,a)
char *a;
{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86, 0, a+1 )+a)):1,t<_?main(t+1, _, a ):3,main ( -94, -27+t, a
)&&t == 2 ?_<13 ?main ( 2, _+1, "%s %d %d\n" ):9:16:t<0?t<-72?main(_,
t,"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+\
,/+#n+,/#;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l q#'+d'K#!/\
+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# ){n\
l]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#\
n'wk nw' iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;\
#'rdq#w! nr'/ ') }+}{rl#'{n' ')# }'+}##(!!/")
:t<-50?_==*a ?putchar(a[31]):main(-65,_,a+1):main((*a == '/')+t,_,a\
+1 ):0<t?main ( 2, 2 , "%s"):*a=='/'||main(0,main(-61,*a, "!ek;dc \
i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);}
```

http://www.kcir.pwr.edu.pl/~mucha/PProg/Pomoce/mystery.c

Instrukcja pętli `while`

```
instrukcja-while = "while (" wyrazenie ")" instrukcja .
```

Instrukcja pętli `while`

```
instrukcja-while = "while (" wyrazenie ")" instrukcja .
```

Sprawdzanie czy liczba `N` jest pierwsza

```
czy_pierwsza = 1;
dzielnik = 2;
while (dzielnik < N)
{
  if (N % dzielnik == 0)
    czy_pierwsza = 0;
  dzielnik = dzielnik + 1;
}
```

Instrukcja pętli `while`

```
instrukcja-while = "while (" wyrazenie ")" instrukcja .
```

Sprawdzanie czy liczba `N` jest pierwsza

```
czy_pierwsza = 1;           /* Lepiej zdefiniowac stala TRUE */
dzielnik = 2;          /* i wtedy przypisac pierwsza = TRUE; */
while (dzielnik < N)
{
  if (N % dzielnik == 0)
    czy_pierwsza = 0;                 /* A tutaj uzyc FALSE */
  dzielnik = dzielnik + 1;
}
```

Instrukcja pętli `while`

```
instrukcja-while = "while (" wyrazenie ")" instrukcja .
```

Sprawdzanie czy liczba `N` jest pierwsza

```
czy_pierwsza = 1;           /* Lepiej zdefiniowac stala TRUE */
dzielnik = 2;          /* i wtedy przypisac pierwsza = TRUE; */
while (dzielnik < N)
{
  if (N % dzielnik == 0)     /* Mozna tez (!(N % dzielnik)) */
    czy_pierwsza = 0;                  /* A tutaj uzyc FALSE */
  dzielnik = dzielnik + 1;
}
```

Instrukcja pętli `while`

```
instrukcja-while = "while (" wyrazenie ")" instrukcja .
```

Sprawdzanie czy liczba `N` jest pierwsza

```
czy_pierwsza = 1;              /* Lepiej zdefiniowac stala TRUE */
dzielnik = 2;            /* i wtedy przypisac pierwsza = TRUE; */
while (dzielnik < N)
{
  if (N % dzielnik == 0)     /* Mozna tez (!(N % dzielnik)) */
    czy_pierwsza = 0;                  /* A tutaj uzyc FALSE */
  dzielnik = dzielnik + 1;       /* Mozna tez dzielnik += 1; */
}                                      /* a takze dzielnik++; */
```

Struktury danych + Algorytm = Program

Podsumowanie

- Zagadnienia podstawowe

1. Do czego służy notacja MBNF?
2. Czy notacja MBNF może zostać wykorzystana do zapisania programu telewizyjnego?
3. Czy napis `.0-0` spełnia podaną na wykładzie definicję liczby rzeczywistej bez znaku?
4. Wymień podstawowe typy danych w języku C.
5. Czy `123abc_` jest poprawnym identyfikatorem w języku C?
6. Opisz sposób deklaracji stałych symblicznych w języku C.
7. Opisz sposób deklaracji zmiennych w języku C.
8. Czy `int _2razy` jest prawidłową nazwą zmiennej?
9. Czy `int 123` jest prawidłową nazwą zmiennej?
10. Czy `int .moja` jest prawidłową nazwą zmiennej?
11. Wymień rodzaje operatorów języka C i podaj ich przykłady.
12. Czy poprawne jest wyrażenie `(74 + 68) * 5 - 7 > 5 * 3`? Jeśli tak, jaka jest jego wartość?
13. W jakiej kolejności wykonane zostaną działania w poniższym wyrażeniu?
    `k+= ++i + --j %3 == 1 && 2 || 0`
14. Czy wyrażenie `x=((x++) * (--x))` jest poprawne składniowo? Jaką będzie miało wartość dla `x=5`?

15. Jaka funkcja w języku C służy do wprowadzenia danych do programu?
16. Jak wygląda składnia instrukcji warunkowej `if`?
17. Jak wygląda składnia instrukcji pętli `while`?
18. Czy można wstawić komentarz w instrukcji warunkowej pomiędzy słowem `if` a wyrażeniem?

• Zagadnienia rozszerzające

1. Zapisz w notacji MBNF definicję adresu pocztowego (Zwrot grzecznościowy; Imię Nazwisko; Ulica nr domu/nr mieszkania (opcjonalny); kod pocztowy Miasto; Kraj).
2. Znajdź w dokumentacji standardu języka C ISO/IEC 9899 definicję literału stałopozycyjnego (integer constant) i porównaj ją z tą przedstawioną na wykładzie.
3. Jakie są standardy nadawania nazw zmiennym?
4. Jakie są wielkości podstawowych typów zmiennych (w bajtach) na posiadanym komputerze domowym, a jakie na diablo?
5. W jaki sposób (binarnie) reprezentowane są liczby całkowite ze znakiem i bez znaku (signed i unsigned)? A jak liczby zmiennoprzecinkowe (float)?
6. Sprawdź, zmienne z których klas zmiennych przedstawionych na wykładzie są inicjowane automatycznie przy ich tworzeniu. Jakimi wartościami inicjowane są zmienne typu `int` a jakimi `char`?
7. W jaki sposób można wywnioskować priorytety operatorów ze składni języka C?
8. Jak będzie wyglądało drzewo wyliczania wartości wyrażeń dla przypadku z punktu 13

z Zagadnień podstawowych przy `i=2`, `j=3`, `k=4`? Jakie będą wartości zmiennych `i`, `j` i `k` po wyliczeniu wyrażenia?

• Zadania

1. Zdefiniuj za pomocą diagramu składni `literal-zmiennopozycyjny`. Spróbuj zdefiniować w ten sposób inne elementy składni języka C.
2. Przy pomocy drzewa wyliczania wartości oblicz wartość wyrażenia `4b^2/(27ab)+c/a-3a/(b+c)` dla `a=4`, `b=9`, `c=3`.
   Powtórz zadanie dla wyrażenia z pominiętymi wszystkimi nawiasami. (Wartości końcowe wyrażeń: 1, 436). Jaka największa wartość pojawi się jako wynik pośredni w obu przypadkach?
3. Napisz program, który sprawdzi, czy wczytana liczba całkowita jest większa od zera, a następnie czy jest ona równa 7. Na ile różnych sposobów można skonstruować instrukcje warunkowe w takim programie?
4. Napisz program, który wczyta 3 liczby całkowite, a następnie wyświetli wartość największej z nich.
5. Napisz program, który wczyta 4 liczby całkowite i sprawdzi ile jest wśród nich powtórzeń (dwójek, trójek lub czwórek).
6. Napisz program wczytujący z klawiatury 3 liczby określające długości odcinków i sprawdzający, czy da się z nich zbudować trójkąt.
7. Napisz program wczytujący z klawiatury 4 liczby określające długości odcinków i spraw-

dzający, czy da się z dowolnych trzech z nich zbudować trójkąt.

8. Napisz program klasyfikujący wzrost osób. Program ma za zadanie pytać o wzrost osoby w cm, a następnie wyświetlać jeden z komunikatów: `Niski`, `Sredni`, `Wysoki` według zasady: `Niski<150cm<Sredni<180cm<Wysoki`.
9. Napisz program wczytujący z klawiatury dwie daty kalendarzowe (zapisywane za pomocą trzech liczb całkowitych: dzień, miesiąc, rok) i sprawdzający czy pierwsza data jest wcześniejsza od drugiej.
10. Liczbami Duffy'ego nazywamy liczby naturalne większe od 1 które nie są liczbami pierwszymi i których suma dzielników właściwych nie jest podzielna przez żaden z dzielników właściwych różnych od 1. Napisz program, który dla danej liczby naturalnej `n > 3` znajdzie wszystkie liczby Duffy'ego nie większe od `n`.

Indeks

- Reguły wyliczania
- Przykłady

- Operatory
  - Priorytety
- Instrukcje
- Motto