Typy danych

Przemysław Gawroński
D-10, p. 234

Wykład 1

12 października 2020

Outline

1. Literatura
2. Programowanie proceduralne
3. Podstawowe typy danych
4. Stałe

Literatura

1. Stephen Prata - Język C: szkoła programowania
2. K.N. King – C Programming. A Modern Approach
3. Jens Gustedt - Modern C
4. Ben Klemens - 21st Century C: C Tips from the New School

Literatura

1. Stephen Prata - Język C: szkoła programowania
2. K.N. King – C Programming. A Modern Approach
3. Jens Gustedt - Modern C
4. Ben Klemens - 21st Century C: C Tips from the New School
5. Stephen G. Kochan - Język C. Kompendium wiedzy
6. Herbert Schildt - Programowanie C
7. Brian W. Kernighan, Dennis M. Ritchie - Język ANSI C
8. Richard M. Reese - Understanding and Using C Pointers
9. Kenneth A. Reek - Język C: wskaźniki: vademecum profesjonalisty

Literatura

1. Stephen Prata - Język C: szkoła programowania
2. K.N. King – C Programming. A Modern Approach
3. Jens Gustedt - Modern C
4. Ben Klemens - 21st Century C: C Tips from the New School
5. Stephen G. Kochan - Język C. Kompendium wiedzy
6. Herbert Schildt - Programowanie C
7. Brian W. Kernighan, Dennis M. Ritchie - Język ANSI C
8. Richard M. Reese - Understanding and Using C Pointers
9. Kenneth A. Reek - Język C: wskaźniki: vademecum profesjonalisty
10. Kyle Loudon - Algorytmy w C
11. Frantisek Franek - Pamięć w językach C i C++

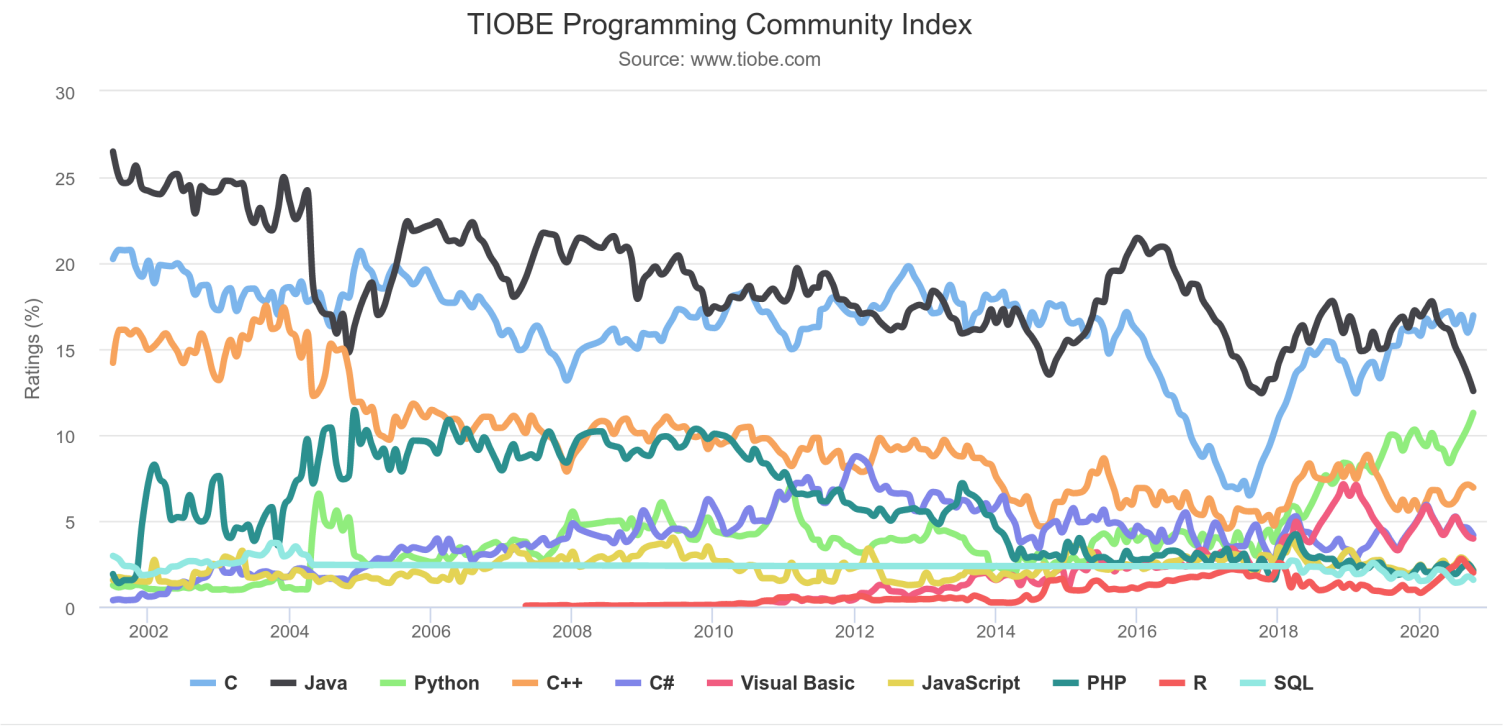
Literatura

1. Stephen Prata - Język C: szkoła programowania
2. K.N. King – C Programming. A Modern Approach
3. Jens Gustedt - Modern C
4. Ben Klemens - 21st Century C: C Tips from the New School
5. Stephen G. Kochan - Język C. Kompendium wiedzy
6. Herbert Schildt - Programowanie C
7. Brian W. Kernighan, Dennis M. Ritchie - Język ANSI C
8. Richard M. Reese - Understanding and Using C Pointers
9. Kenneth A. Reek - Język C: wskaźniki: vademecum profesjonalisty
10. Kyle Loudon - Algorytmy w C
11. Frantisek Franek - Pamięć w językach C i C++
12. Steve Summit – Programowanie w języku C. FAQ
13. Peter van der Linden - Expert C Programming. Deep C Secrets

Literatura

1. Stephen Prata - Język C: szkoła programowania
2. K.N. King – C Programming. A Modern Approach
3. Jens Gustedt - Modern C
4. Ben Klemens - 21st Century C: C Tips from the New School
5. Stephen G. Kochan - Język C. Kompendium wiedzy
6. Herbert Schildt - Programowanie C
7. Brian W. Kernighan, Dennis M. Ritchie - Język ANSI C
8. Richard M. Reese - Understanding and Using C Pointers
9. Kenneth A. Reek - Język C: wskaźniki: vademecum profesjonalisty
10. Kyle Loudon - Algorytmy w C
11. Frantisek Franek - Pamięć w językach C i C++
12. Steve Summit – Programowanie w języku C. FAQ
13. Peter van der Linden - Expert C Programming. Deep C Secrets
14. http://stackoverflow.com/questions/tagged/c

TIOBE INDEX - http://www.tiobe.com



- Programming Language of the Year
  - 2019 - C
  - 2017 - C
  - 2008 - C
- The award is given to the programming language that has the highest rise in ratings in a year.

Czym jest programowanie?

Czym jest programowanie?

- Programowanie - rozwiązywanie problemów za pomocą komputera.

Czym jest programowanie?

- Programowanie - rozwiązywanie problemów za pomocą komputera.
- Programowanie proceduralne: modularność, biblioteki funkcji, zmienne lokalne, zmienne globalne, przekazywanie parametrów.

Czym jest programowanie?

- Programowanie - rozwiązywanie problemów za pomocą komputera.
- Programowanie proceduralne: modularność, biblioteki funkcji, zmienne lokalne, zmienne globalne, przekazywanie parametrów.
- Programowanie proceduralne w języku C – zarządzanie pamięcią:

Czym jest programowanie?

- Programowanie - rozwiązywanie problemów za pomocą komputera.
- Programowanie proceduralne: modularność, biblioteki funkcji, zmienne lokalne, zmienne globalne, przekazywanie parametrów.
- Programowanie proceduralne w języku C – zarządzanie pamięcią:
  - Alokacja pamięci (zmienne):

Czym jest programowanie?

- Programowanie - rozwiązywanie problemów za pomocą komputera.
- Programowanie proceduralne: modularność, biblioteki funkcji, zmienne lokalne, zmienne globalne, przekazywanie parametrów.
- Programowanie proceduralne w języku C – zarządzanie pamięcią:
  - Alokacja pamięci (zmienne):
    - automatyczna/statyczna (typ, nazwa zmiennej, zakres widzialności),

Czym jest programowanie?

- Programowanie - rozwiązywanie problemów za pomocą komputera.
- Programowanie proceduralne: modularność, biblioteki funkcji, zmienne lokalne, zmienne globalne, przekazywanie parametrów.
- Programowanie proceduralne w języku C – zarządzanie pamięcią:
  - Alokacja pamięci (zmienne):
    - automatyczna/statyczna (typ, nazwa zmiennej, zakres widzialności),
    - dynamiczna (malloc, realloc)

Czym jest programowanie?

- Programowanie - rozwiązywanie problemów za pomocą komputera.
- Programowanie proceduralne: modularność, biblioteki funkcji, zmienne lokalne, zmienne globalne, przekazywanie parametrów.
- Programowanie proceduralne w języku C – zarządzanie pamięcią:
  - Alokacja pamięci (zmienne):
    - automatyczna/statyczna (typ, nazwa zmiennej, zakres widzialności),
    - dynamiczna (malloc, realloc)
  - Operacje na zmiennych, modyfikacja wartości zmiennych, zmiana stanu programu:

Czym jest programowanie?

- Programowanie - rozwiązywanie problemów za pomocą komputera.
- Programowanie proceduralne: modularność, biblioteki funkcji, zmienne lokalne, zmienne globalne, przekazywanie parametrów.
- Programowanie proceduralne w języku C – zarządzanie pamięcią:
  - Alokacja pamięci (zmienne):
    - automatyczna/statyczna (typ, nazwa zmiennej, zakres widzialności),
    - dynamiczna (malloc, realloc)
  - Operacje na zmiennych, modyfikacja wartości zmiennych, zmiana stanu programu:
    - operatory,

Czym jest programowanie?

- Programowanie - rozwiązywanie problemów za pomocą komputera.
- Programowanie proceduralne: modularność, biblioteki funkcji, zmienne lokalne, zmienne globalne, przekazywanie parametrów.
- Programowanie proceduralne w języku C – zarządzanie pamięcią:
  - Alokacja pamięci (zmienne):
    - automatyczna/statyczna (typ, nazwa zmiennej, zakres widzialności),
    - dynamiczna (malloc, realloc)
  - Operacje na zmiennych, modyfikacja wartości zmiennych, zmiana stanu programu:
    - operatory,
    - funkcje (przekazywanie i zwracanie zmiennych).

Czym jest programowanie?

- Programowanie - rozwiązywanie problemów za pomocą komputera.
- Programowanie proceduralne: modularność, biblioteki funkcji, zmienne lokalne, zmienne globalne, przekazywanie parametrów.
- Programowanie proceduralne w języku C – zarządzanie pamięcią:
  - Alokacja pamięci (zmienne):
    - automatyczna/statyczna (typ, nazwa zmiennej, zakres widzialności),
    - dynamiczna (malloc, realloc)
  - Operacje na zmiennych, modyfikacja wartości zmiennych, zmiana stanu programu:
    - operatory,
    - funkcje (przekazywanie i zwracanie zmiennych).
  - Dealokacja pamięci - automatyczna, bądź za pomocą funkcji free.

Podstawowe typy danych

- W C89 zdefiniowanych jest 5 podstawowych typów danych:

Podstawowe typy danych

- W C89 zdefiniowanych jest 5 podstawowych typów danych:
  1. brak wartości - **void**,

Podstawowe typy danych

- W C89 zdefiniowanych jest 5 podstawowych typów danych:
  1. brak wartości - **void**,
  2. znak - **char**

Podstawowe typy danych

- W C89 zdefiniowanych jest 5 podstawowych typów danych:
  1. brak wartości - **void**,
  2. znak - **char**
     You can pronounce the C keyword char in at least three ways: like the English words char, care, or car (or maybe even character); the choice is arbitrary.

Podstawowe typy danych

- W C89 zdefiniowanych jest 5 podstawowych typów danych:
  1. brak wartości - **void**,
  2. znak - **char**
     You can pronounce the C keyword char in at least three ways: like the English words char, care, or car (or maybe even character); the choice is arbitrary.
  3. liczba całkowita - **int**,

Podstawowe typy danych

- W C89 zdefiniowanych jest 5 podstawowych typów danych:
  1. brak wartości - **void**,
  2. znak - **char**
     You can pronounce the C keyword char in at least three ways: like the English words char, care, or car (or maybe even character); the choice is arbitrary.
  3. liczba całkowita - **int**,
  4. liczba zmiennoprzecinkowa - **float**,

Podstawowe typy danych

- W C89 zdefiniowanych jest 5 podstawowych typów danych:
  1. brak wartości - **void**,
  2. znak - **char**
     You can pronounce the C keyword char in at least three ways: like the English words char, care, or car (or maybe even character); the choice is arbitrary.
  3. liczba całkowita - **int**,
  4. liczba zmiennoprzecinkowa - **float**,
  5. liczba zmiennoprzecinkowa podwójnej precyzji - **double**.

Modyfikatory typów

- Typy podstawowe stanowią bazę do budowania typów złożonych i typów zmodyfikowanych.

Modyfikatory typów

- Typy podstawowe stanowią bazę do budowania typów złożonych i typów zmodyfikowanych.
- Rozmiar oraz zakres zmieniają się w zależności od procesora oraz kompilatora. Nie można przyjmować żadnych założeń odnośnie rozmiaru typów, jeżeli chcemy zapewnić przenośność programu między środowiskami wykonania.

Modyfikatory typów

- Typy podstawowe stanowią bazę do budowania typów złożonych i typów zmodyfikowanych.
- Rozmiar oraz zakres zmieniają się w zależności od procesora oraz kompilatora. Nie można przyjmować żadnych założeń odnośnie rozmiaru typów, jeżeli chcemy zapewnić przenośność programu między środowiskami wykonania.
- Standard języka C definiuje jedynie minimalny zakres typu, a nie rozmiar w bajtach.

Modyfikatory typów

- Typy podstawowe stanowią bazę do budowania typów złożonych i typów zmodyfikowanych.
- Rozmiar oraz zakres zmieniają się w zależności od procesora oraz kompilatora. Nie można przyjmować żadnych założeń odnośnie rozmiaru typów, jeżeli chcemy zapewnić przenośność programu między środowiskami wykonania.
- Standard języka C definiuje jedynie minimalny zakres typu, a nie rozmiar w bajtach.
- Modyfikatory typów zmieniają znaczenie typu podstawowego, w celu lepszego dostosowania go do rozwiązywanego problemu.

Modyfikatory typów

- Typy podstawowe stanowią bazę do budowania typów złożonych i typów zmodyfikowanych.
- Rozmiar oraz zakres zmieniają się w zależności od procesora oraz kompilatora. Nie można przyjmować żadnych założeń odnośnie rozmiaru typów, jeżeli chcemy zapewnić przenośność programu między środowiskami wykonania.
- Standard języka C definiuje jedynie minimalny zakres typu, a nie rozmiar w bajtach.
- Modyfikatory typów zmieniają znaczenie typu podstawowego, w celu lepszego dostosowania go do rozwiązywanego problemu.
- Modyfikatory typu **int** – **signed, unsigned, long, short**.

Modyfikatory typów

- Typy podstawowe stanowią bazę do budowania typów złożonych i typów zmodyfikowanych.
- Rozmiar oraz zakres zmieniają się w zależności od procesora oraz kompilatora. Nie można przyjmować żadnych założeń odnośnie rozmiaru typów, jeżeli chcemy zapewnić przenośność programu między środowiskami wykonania.
- Standard języka C definiuje jedynie minimalny zakres typu, a nie rozmiar w bajtach.
- Modyfikatory typów zmieniają znaczenie typu podstawowego, w celu lepszego dostosowania go do rozwiązywanego problemu.
- Modyfikatory typu **int** – **signed, unsigned, long, short**.
- Modyfikatory typu **char** – **signed, unsigned**.

Modyfikatory typów

- Typy podstawowe stanowią bazę do budowania typów złożonych i typów zmodyfikowanych.
- Rozmiar oraz zakres zmieniają się w zależności od procesora oraz kompilatora. Nie można przyjmować żadnych założeń odnośnie rozmiaru typów, jeżeli chcemy zapewnić przenośność programu między środowiskami wykonania.
- Standard języka C definiuje jedynie minimalny zakres typu, a nie rozmiar w bajtach.
- Modyfikatory typów zmieniają znaczenie typu podstawowego, w celu lepszego dostosowania go do rozwiązywanego problemu.
- Modyfikatory typu **int** – **signed, unsigned, long, short**.
- Modyfikatory typu **char** – **signed, unsigned**.
- Modyfikator **long** stosujemy z typem **double** (oraz w standardzie C99 z typem **long int**).

Typy całkowite - <limits.h>(gcc 9.2.1, 64bit)

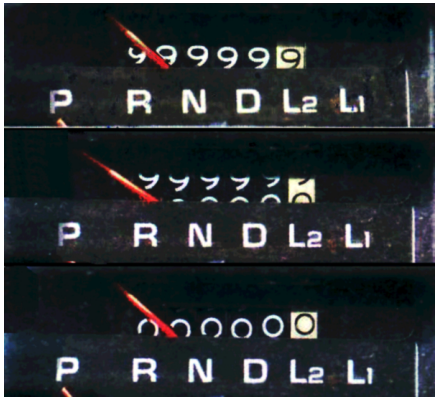
Typ	Typowy rozmiar w bitach	Minimalny zakres
char	8	$-128 \div 127$
unsigned char	8	$0 \div 255$
signed char	8	$-128 \div 127$
short int	16	$-32768 \div 32767$
signed short int	16	$-32768 \div 32767$
unsigned short int	16	$0 \div 65535$
int	32	$-2147483648 \div 2147483647$
signed int	32	$-2147483648 \div 2147483647$
unsigned int	32	$0 \div 4294967295$

Typy całkowite - <limits.h>(gcc 9.2.1, 64bit)

Typ	Typowy rozmiar w bitach	Minimalny zakres
long int	64	−9223372036854775808÷ 9223372036854775807
unsigned long int	64	0÷ 18446744073709551615
signed long int	64	−9223372036854775808÷ 9223372036854775807

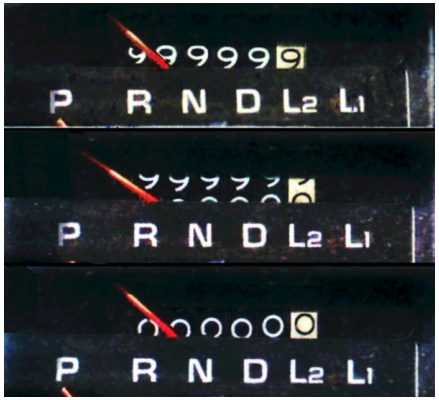
Typ (C99 i C11)	Typowy rozmiar w bitach	Minimalny zakres
long long int	64	−9223372036854775808÷ 9223372036854775807
unsigned long long int	64	0÷ 18446744073709551615
signed long long int	64	−9223372036854775808÷ 9223372036854775807

"Przekręcanie licznika"



"Przekręcanie licznika"

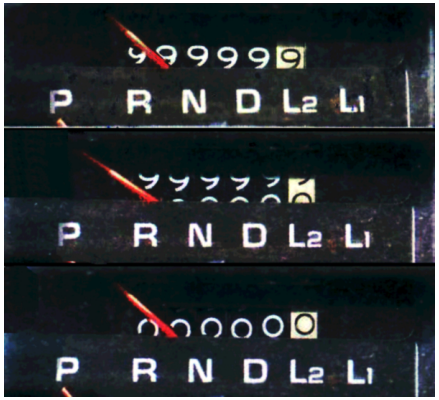
```
short int x = 32767;
printf("%hd\n", x); // 32767
x = x + 1;
printf("%hd\n", x); //-32768
```



"Przekręcanie licznika"

```
short int x = 32767;
printf("%hd\n", x); // 32767
x = x + 1;
printf("%hd\n", x); //-32768

unsigned int y=0;
printf("%u\n", y); // 0
y = y - 1;
printf("%u\n", y); // 4294967295
```



Typy rzeczywiste - <float.h>(gcc 9.2.1, 64bit)

- Liczba rzeczywista x jest przechowywana w pamięci zgodnie z wzorem (IEEE 754) $x = (-1)^s mantysa \cdot 2^{wykladnik - bias}$

Typy rzeczywiste - <float.h>(gcc 9.2.1, 64bit)

- Liczba rzeczywista x jest przechowywana w pamięci zgodnie z wzorem (IEEE 754) $x = (-1)^s mantysa \cdot 2^{wykladnik - bias}$
- **double** - 64 bity (8 bajtów):

Typy rzeczywiste - <float.h>(gcc 9.2.1, 64bit)

- Liczba rzeczywista x jest przechowywana w pamięci zgodnie z wzorem (IEEE 754) $x = (-1)^s mantysa \cdot 2^{wykladnik-bias}$
- **double** - 64 bity (8 bajtów):
  - **s** - 1 bit znaku (+ or -),

Typy rzeczywiste - <float.h>(gcc 9.2.1, 64bit)

- Liczba rzeczywista x jest przechowywana w pamięci zgodnie z wzorem (IEEE 754) $x = (-1)^s mantysa \cdot 2^{wykladnik-bias}$
- **double** - 64 bity (8 bajtów):
  - **s** - 1 bit znaku (+ or -),
  - **wykładnik** - 11 bitów,

Typy rzeczywiste - <float.h>(gcc 9.2.1, 64bit)

- Liczba rzeczywista x jest przechowywana w pamięci zgodnie z wzorem (IEEE 754) $x = (-1)^s mantysa \cdot 2^{wykladnik-bias}$
- **double** - 64 bity (8 bajtów):
  - **s** - 1 bit znaku (+ or -),
  - **wykładnik** - 11 bitów,
  - **mantysa** - 52 bitów,

Typy rzeczywiste - <float.h>(gcc 9.2.1, 64bit)

- Liczba rzeczywista x jest przechowywana w pamięci zgodnie z wzorem (IEEE 754) $x = (-1)^s mantysa \cdot 2^{wykladnik-bias}$
- **double** - 64 bity (8 bajtów):
  - **s** - 1 bit znaku (+ or -),
  - **wykładnik** - 11 bitów,
  - **mantysa** - 52 bitów,
  - **bias** - pozwala by wykładnik był zawsze dodatni.

Typy rzeczywiste - <float.h>(gcc 9.2.1, 64bit)

- Liczba rzeczywista x jest przechowywana w pamięci zgodnie z wzorem (IEEE 754) $x = (-1)^s mantysa \cdot 2^{wykladnik-bias}$
- **double** - 64 bity (8 bajtów):
  - **s** - 1 bit znaku (+ or -),
  - **wykładnik** - 11 bitów,
  - **mantysa** - 52 bitów,
  - **bias** - pozwala by wykładnik był zawsze dodatni.

Typ	Liczba bitów	Precyzja Cyfry znaczące/bity	Wykładnik min/max/bity
float	32	6 cyfr/23 bity	-128/+127/8
double	64	15 cyfr/52 bity	-1022/+1023/11
long double	128	18 cyfr/112 bity	-16382/16383/15

Stałe

- Stałe to ustalone wartości, których program nie może zmienić; mogą należeć do jednego z podstawowych typów.

Stałe

- Stałe to ustalone wartości, których program nie może zmienić; mogą należeć do jednego z podstawowych typów.
- Sposób reprezentacji stałej zależy od jej typu.

Stałe

- Stałe to ustalone wartości, których program nie może zmienić; mogą należeć do jednego z podstawowych typów.
- Sposób reprezentacji stałej zależy od jej typu.
- Stałe nazywane też są wartościami natychmiastowymi lub literałami.

Stałe

- Stałe to ustalone wartości, których program nie może zmienić; mogą należeć do jednego z podstawowych typów.
- Sposób reprezentacji stałej zależy od jej typu.
- Stałe nazywane też są wartościami natychmiastowymi lub literałami.
- Stałe znakowe: 'a', '?'.

Stałe

- Stałe to ustalone wartości, których program nie może zmienić; mogą należeć do jednego z podstawowych typów.
- Sposób reprezentacji stałej zależy od jej typu.
- Stałe nazywane też są wartościami natychmiastowymi lub literałami.
- Stałe znakowe: 'a', '?'.
- Stałe całkowite: 123, 3242.

Stałe

- Stałe to ustalone wartości, których program nie może zmienić; mogą należeć do jednego z podstawowych typów.
- Sposób reprezentacji stałej zależy od jej typu.
- Stałe nazywane też są wartościami natychmiastowymi lub literałami.
- Stałe znakowe: 'a', '?'.
- Stałe całkowite: 123, 3242.
- Stałe zmiennoprzecinkowe: 123.3242, $123.16e-11$.

Stałe

- Stałe to ustalone wartości, których program nie może zmienić; mogą należeć do jednego z podstawowych typów.
- Sposób reprezentacji stałej zależy od jej typu.
- Stałe nazywane też są wartościami natychmiastowymi lub literałami.
- Stałe znakowe: 'a', '?'.
- Stałe całkowite: 123, 3242.
- Stałe zmiennoprzecinkowe: 123.3242, $123.16e-11$.
- Standardowo kompilator dopasowuje stałą liczbową do najmniejszego typu danych.

Stałe

- Stałe to ustalone wartości, których program nie może zmienić; mogą należeć do jednego z podstawowych typów.
- Sposób reprezentacji stałej zależy od jej typu.
- Stałe nazywane też są wartościami natychmiastowymi lub literałami.
- Stałe znakowe: 'a', '?'.
- Stałe całkowite: 123, 3242.
- Stałe zmiennoprzecinkowe: 123.3242, $123.16e-11$.
- Standardowo kompilator dopasowuje stałą liczbową do najmniejszego typu danych.
- Dla 16-bitowego typu całkowitego 10 będzie typu **int**, ale 103000 będzie typem **long int**.

Stałe

- Stałe to ustalone wartości, których program nie może zmienić; mogą należeć do jednego z podstawowych typów.
- Sposób reprezentacji stałej zależy od jej typu.
- Stałe nazywane też są wartościami natychmiastowymi lub literałami.
- Stałe znakowe: 'a', '?'.
- Stałe całkowite: 123, 3242.
- Stałe zmiennoprzecinkowe: 123.3242, $123.16e-11$.
- Standardowo kompilator dopasowuje stałą liczbową do najmniejszego typu danych.
- Dla 16-bitowego typu całkowitego 10 będzie typu **int**, ale 103000 będzie typem **long int**.
- Stałe zmiennoprzecinkowe zawsze są typu **double**.

Stałe

- Aby dokładnie określić typ stałej należy korzystać z przyrostka.

Stałe

- Aby dokładnie określić typ stałej należy korzystać z przyrostka.
- Stałe zmiennoprzecinkowe:
  - F lub f – stała typu **float**
  - L lub l - stała typu **long double**

Stałe

- Aby dokładnie określić typ stałej należy korzystać z przyrostka.
- Stałe zmiennoprzecinkowe:
  - F lub f – stała typu **float**
  - L lub l - stała typu **long double**
- Stałe całkowite:

Stałe

- Aby dokładnie określić typ stałej należy korzystać z przyrostka.
- Stałe zmiennoprzecinkowe:
  - F lub f – stała typu **float**
  - L lub l - stała typu **long double**
- Stałe całkowite:
  - U lub u – stała typu **unsigned int**
  - L lub l - stała typu **long int**

Stałe

- Aby dokładnie określić typ stałej należy korzystać z przyrostka.
- Stałe zmiennoprzecinkowe:
  - F lub f – stała typu **float**
  - L lub l - stała typu **long double**
- Stałe całkowite:
  - U lub u – stała typu **unsigned int**
  - L lub l - stała typu **long int**
- Przykłady:

Stałe

- Aby dokładnie określić typ stałej należy korzystać z przyrostka.
- Stałe zmiennoprzecinkowe:
  - F lub f – stała typu **float**
  - L lub l - stała typu **long double**
- Stałe całkowite:
  - U lub u – stała typu **unsigned int**
  - L lub l - stała typu **long int**
- Przykłady:
  - 1, 123, -234 – stałe typu **int**

Stałe

- Aby dokładnie określić typ stałej należy korzystać z przyrostka.
- Stałe zmiennoprzecinkowe:
  - F lub f – stała typu **float**
  - L lub l - stała typu **long double**
- Stałe całkowite:
  - U lub u – stała typu **unsigned int**
  - L lub l - stała typu **long int**
- Przykłady:
  - 1, 123, -234 – stałe typu **int**
  - 35000L, -34L – stałe typu **long int**

Stałe

- Aby dokładnie określić typ stałej należy korzystać z przyrostka.
- Stałe zmiennoprzecinkowe:
  - F lub f – stała typu **float**
  - L lub l - stała typu **long double**
- Stałe całkowite:
  - U lub u – stała typu **unsigned int**
  - L lub l - stała typu **long int**
- Przykłady:
  - 1, 123, -234 – stałe typu **int**
  - 35000L, -34L – stałe typu **long int**
  - 10000U, 987u – stałe typu **unsigned int**

Stałe

- Aby dokładnie określić typ stałej należy korzystać z przyrostka.
- Stałe zmiennoprzecinkowe:
  - F lub f – stała typu **float**
  - L lub l - stała typu **long double**
- Stałe całkowite:
  - U lub u – stała typu **unsigned int**
  - L lub l - stała typu **long int**
- Przykłady:
  - 1, 123, -234 – stałe typu **int**
  - 35000L, -34L – stałe typu **long int**
  - 10000U, 987u – stałe typu **unsigned int**
  - 123.23F, 4.34e-2f – stałe typu **float**

Stałe

- Aby dokładnie określić typ stałej należy korzystać z przyrostka.
- Stałe zmiennoprzecinkowe:
  - F lub f – stała typu **float**
  - L lub l - stała typu **long double**
- Stałe całkowite:
  - U lub u – stała typu **unsigned int**
  - L lub l - stała typu **long int**
- Przykłady:
  - 1, 123, -234 – stałe typu **int**
  - 35000L, -34L – stałe typu **long int**
  - 10000U, 987u – stałe typu **unsigned int**
  - 123.23F, 4.34e-2f – stałe typu **float**
  - 123.23, 1.0, -0.89898 – stałe typu **double**

Stałe

- Aby dokładnie określić typ stałej należy korzystać z przyrostka.
- Stałe zmiennoprzecinkowe:
  - F lub f – stała typu **float**
  - L lub l - stała typu **long double**
- Stałe całkowite:
  - U lub u – stała typu **unsigned int**
  - L lub l - stała typu **long int**
- Przykłady:
  - 1, 123, -234 – stałe typu **int**
  - 35000L, -34L – stałe typu **long int**
  - 10000U, 987u – stałe typu **unsigned int**
  - 123.23F, 4.34e-2f – stałe typu **float**
  - 123.23, 1.0, -0.89898 – stałe typu **double**
  - 123.23L, 1000.12L – stałe typu **long double**

Stałe

- W C można używać liczb zapisanych w systemach ósemkowym lub szesnastkowym:

Stałe

- W C można używać liczb zapisanych w systemach ósemkowym lub szesnastkowym:
  - int hex = 0x80; $//80_{16} = 128_{10}$

Stałe

- W C można używać liczb zapisanych w systemach ósemkowym lub szesnastkowym:
  - int hex = 0x80; $//80_{16} = 128_{10}$
  - int oct = 012; $//12_8 = 10_{10}$

Stałe

- W C można używać liczb zapisanych w systemach ósemkowym lub szesnastkowym:
  - int hex = 0x80; $//80_{16} = 128_{10}$
  - int oct = 012; $//12_8 = 10_{10}$
- Stała tekstowa to zbiór znaków otoczonych znakami cudzysłowu - "zbiór znaków i cyfra 2 w otoczeniu".

Stałe

- W C można używać liczb zapisanych w systemach ósemkowym lub szesnastkowym:
  - int hex = 0x80; $//80_{16} = 128_{10}$
  - int oct = 012; $//12_8 = 10_{10}$
- Stała tekstowa to zbiór znaków otoczonych znakami cudzysłowu - "zbiór znaków i cyfra 2 w otoczeniu".
- Uwaga:

Stałe

- W C można używać liczb zapisanych w systemach ósemkowym lub szesnastkowym:
  - int hex = 0x80; $//80_{16} = 128_{10}$
  - int oct = 012; $//12_8 = 10_{10}$
- Stała tekstowa to zbiór znaków otoczonych znakami cudzysłowu - "zbiór znaków i cyfra 2 w otoczeniu".
- Uwaga:
  - a – zmienna,

Stałe

- W C można używać liczb zapisanych w systemach ósemkowym lub szesnastkowym:
  - int hex = 0x80; $//80_{16} = 128_{10}$
  - int oct = 012; $//12_8 = 10_{10}$
- Stała tekstowa to zbiór znaków otoczonych znakami cudzysłowu - "zbiór znaków i cyfra 2 w otoczeniu".
- Uwaga:
  - a – zmienna,
  - 'a' – znak,

Stałe

- W C można używać liczb zapisanych w systemach ósemkowym lub szesnastkowym:
  - int hex = 0x80; $//80_{16} = 128_{10}$
  - int oct = 012; $//12_8 = 10_{10}$
- Stała tekstowa to zbiór znaków otoczonych znakami cudzysłowu - "zbiór znaków i cyfra 2 w otoczeniu".
- Uwaga:
  - a – zmienna,
  - 'a' – znak,
  - "a" – stała tekstowa, tablica znaków zakończona znakiem końca łańcucha.

Sekwencje specjalne

Sekwencje specjalne to znaki sterujące, których nie można wpisywać bezpośrednio z klawiatury.

```
\b - backspace
\f - nowa strona
\n - nowy wiersz
\r - powrót karetki
\t - poziomy tabulator
\" - znak cudzysłowu
\' - apostrof
\\ - lewy ukośnik
\v - pionowy tabulator
\a - dzwonek
\? - znak zapytania
\N - stała ósemkowa (N oznacza stała zapisana ósemkowo)
\xN - stała szesnastkowa
```

Biblioteka wejścia/wyjścia, funkcja **printf()**

```
//plik wypiszStale.c
#include <stdio.h>
int main(void) {
  printf("Znak␣to␣spacja.\n");
  printf("char:␣%c␣\n", 'a');
  printf("char(kod␣ASCII):␣%d␣\n", 'a');
  printf("string:␣%s␣\n\n", "abc");
  printf("int:␣%d␣\n", 178);
  printf("long␣int:␣%ld␣\n", 2147483648L);
  printf("unsigned␣int:␣%u␣\n", 1234U);
  printf("unsigned␣long␣int:␣%lu␣\n\n",
   18446744073709551615UL);
  printf("float:␣%f\n", 3.14F);
  printf("double:␣%lf\n", 3.14);
  printf("long␣double:␣%Lf\n", 3.14L);
  return 0;
}
```

Biblioteka wejścia/wyjścia, funkcja **printf()**

- Kompilacja i konsolidacja: **gcc wypiszStale.c**

Biblioteka wejścia/wyjścia, funkcja **printf()**

- Kompilacja i konsolidacja: **gcc wypiszStale.c**
- Wykonanie: **./a.out**

Biblioteka wejścia/wyjścia, funkcja **printf()**

- Kompilacja i konsolidacja: **gcc wypiszStale.c**
- Wykonanie: **./a.out**

```
char: a
char(kod ASCII): 97
string: abc

int: 178
long int: 2147483648
unsigned int: 1234
unsigned long int: 18446744073709551615

float: 3.140000
double: 3.140000
long double: 3.140000
```

Biblioteka wejścia/wyjścia, funkcja **printf()**

- Kompilacja i konsolidacja: **gcc wypiszStale.c**
- Wykonanie: **./a.out**

```
char: a
char(kod ASCII): 97
string: abc

int: 178
long int: 2147483648
unsigned int: 1234
unsigned long int: 18446744073709551615

float: 3.140000
double: 3.140000
long double: 3.140000
```

- **gcc wypiszStale.c -Wall**

Biblioteka wejścia/wyjścia, funkcja **printf()**

- Kompilacja i konsolidacja: **gcc wypiszStale.c**
- Wykonanie: **./a.out**

```
char: a
char(kod ASCII): 97
string: abc

int: 178
long int: 2147483648
unsigned int: 1234
unsigned long int: 18446744073709551615

float: 3.140000
double: 3.140000
long double: 3.140000
```

- **gcc wypiszStale.c -Wall**
- **gcc wypiszStale.c -o wypiszStale**
- **./wypiszStale**