Typy złożone

Typ wyliczeniowy, pola bitowe i unie.

Ostatnia aktualizacja: 2 lutego 2026

Typy podstawowe

- Typy całkowite:
  - `char`
  - `short`
  - `int`
  - `long`
- Typy zmiennopozycyjne
  - `float`
  - `double`
- Modyfikatory : `unsigned`, `signed`
- Typ wskaźnikowy

Typy złożone

- Tablice
- Struktury `struct`
- Unie `union`
- Pola bitowe
- Typ wyliczeniowy `enum`
- Wskaźniki na funkcje i wskaźniki na typy złożone

Deklaracja

```
enum nazwa_typu
{
    element1,
    element2,
    ...
};
```

Przykład

```
enum kolory { czerwony, niebieski, zielony, czarny };

int main()
{
    enum kolory n;
    n = 0;
    if (n == czerwony) printf("To kolor czerwony\n");
}
```

- Typ wyliczeniowy `enum` to liczba całkowita (prawie to samo co `int`)
- Operacje arytmetyczne takie same jak na typie całkowitym

```
enum kolory n=zielony;
n++;
n=n+100;
```

- Poprawia czytelność kodu
- Może być zastąpiony przez odpowiednie dyrektywy, np.:

```
#define ZIELONY 1
#define CZERWONY 2
int kolor = ZIELONY;
```

Przykład:

```c
#include <stdio.h>

enum kolory { zielony, niebieski, czerwony, czarny };

int main()
{
    enum kolory n;

    for(n=zielony; n<=10; n++)
        printf("%d\n", n);

    return 0;
}
```

```
0
1
2
3
```

☞ enum1.c

Przykład:

```c
#include <stdio.h>

enum kolory { zielony=3, niebieski=7, czerwony, czarny };

int main()
{
    enum kolory n;

    printf("sizeof = %d\n", sizeof(n));

    for(n=zielony; n<=czarny; n++)
        printf("%d\n", n);

    return 0;
}
```

```
sizeof = 4
3
4
5
6
7
8
9
```

☞ enum2.c

- Polecenie `typedef` pozwala nadać własne nazwy dowolnym typom
- Deklaracja typu wygląda identycznie jak deklaracja zmiennej, należy tylko dodać słowo `typedef` na początku

```
typedef float liczba;
typedef float punkt[4];
typedef enum { false, true } bool;
typedef float *wskaznik;

int main()
{
   liczba x = 3.14;
   punkt y;
   bool czy_koniec = false;
   wskaznik w = &x;
}
```

- Polecenie `typedef` pozwala uniknąć powtarzania słowa `struct` lub `enum`

```
struct student
{
    char nazwisko[100];
    int indeks;
};

typedef struct student student_s;

void wypisz_student(student_s s)
{
    printf("Nazwisko: %s\n", s.nazwisko);
}
```

Operator -> umożliwia dostęp do pola struktury za pomocą wskaźnika.
`wskaznik->pole` jest równoważne z `(*wskaznik).pole`

```c
#include <stdio.h>

typedef enum { M, K } plec;

typedef struct
{
    char nazwisko[100];
    int indeks;
    plec p;
} student;

void wypisz_studenta(const student *s)
{
    printf("Nazwisko: %s\n", s->nazwisko);
    printf("Indeks  : %d\n", (*s).indeks);
    printf("Plec    : %s\n", s->p == M ? "M" : "K");
}

int main()
{
    student s = {"Zenon Adamski", 123456, M};
    wypisz_studenta(&s);
    return 0;
}
```

☞ struct2.c

- Unie, podobnie jak struktury, przechowują w polach zmienne dowolnego typu
- W odróżnieniu od struktur, wszystkie zmienne umieszczone są pod tym samym adresem (przechowywana jest pojedyncza wartość).
- Modyfikacja jednego pola zmienia wartośc pozostałych.

```
1 union ufloat
2 {
3     float f;
4     unsigned int i;
5     unsigned char c[4];
6 };
```

☞ union1.c

```c
#include <stdio.h>

union liczba
{
   float f;
   unsigned int i;
   unsigned char c;
};

int main()
{
  union liczba x;

  x.f = 3.14;

  printf("sizeof = %d\n", sizeof(union liczba));

  printf("adres pola f = %p\n", &x.f);
  printf("adres pola i = %p\n", &x.i);
  printf("adres pola c = %p\n", &x.c);

  printf("wartosc pola f = %f\n", x.f);
  printf("wartosc pola i = %x\n", x.i);
  printf("wartosc pola c = %x\n", x.c);

}
```

☞ union2.c

- Pola bitowe to pola struktury o określonej liczbie bitów
- Pozwalają zoptymalizować zużycie pamięci w reprezentacji zmiennych całkowitych
- Wykorzystywane m. in. przy obsłudze urządzeń zewnętrznych przez porty

```
struct data
{
   unsigned int dzien    :  5 ;
   unsigned int miesiac  :  4 ;
   unsigned int rok      : 11 ;
};
```

```c
#include<stdio.h>

typedef struct
{
   unsigned int dzien   :  5 ;
   unsigned int miesiac :  4 ;
   unsigned int rok     : 11 ;
} data;

int main()
{
   data dzis = { 18, 16, 2022 };

   printf("Data: %u-%u-%u\n",
          dzis.dzien, dzis.miesiac, dzis.rok);

   printf("sizeof(data)=%ld\n", sizeof(data));

   return 0;
```

☞ bitfield.c

Pola bitowe pozwalają poprawić czytelnośc kodu zawierającego operacje na bitach, jednak działają wolniej niż operatory bitowe.

```
typedef struct
{
    int prawo_odczytu: 1;
    int prawo_zapisu : 1;
    int prawo_wykonania : 1;

} atrybuty_pliku;

int main()
{
    atrybuty_pliku plik;

    if ( plik.prawo_odczytu )
    {
        /*  ...  */
    }
}
```

- deklarowanie własnych typów poleceniem `typedef`
- struktury, unie i pola bitowe
- typ wyliczeniowy `enum`