

Wstęp do programowania w języku C

Wykład 4 - Operacje na bitach, definiowanie typów

26 października 2016

Tablice jako parametry i argumenty funkcji

- Argumentu są przekazywane do funkcji *przez wartość*.
- Oznacza to kopiowanie wartości argumentu do odpowiedniego parametru funkcji, który jest traktowany jak lokalna zmienna z zadaną wartością początkową.
- Ale tablice mogą być duże, a ich kopiowanie - nieefektywne.
- Dlatego w przypadku tablic przekazywany jest adres (wskaźnik na) początku tablicy.
- Słowo `const` przed deklaracją parametru tablicowego zabrania funkcji modyfikacji tablicy.

Operatory bitowe języka C

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.	

Odczytywanie i ustawianie wskazanego bitu w liczbie

ZADANIE: Napisać trzy funkcje, z których:

- pierwsza sprawdza, czy k-ty bit liczby typu `unsigned int` jest jedyneką;
- druga ustawia k-ty bit liczby na jeden;
- trzecia ustawia k-ty bit liczby na zero.

UWAGA: Bit numer zero to najmniej znaczący bit liczby.

bity.c - implementacja

```
#include <stdbool.h>
#include <limits.h>

const int maks_numer_bitu = (CHAR_BIT*sizeof(unsigned int)) - 1;

bool bit_jedynka(unsigned int liczba, int k)
{
    if (k < 0 || k > maks_numer_bitu)
        return false;
    return liczba & (1 << k);
}

unsigned int ustaw_bit_na_1(unsigned int liczba, int k)
{
    if (0 <= k && k <= maks_numer_bitu)
        liczba = liczba | (1 << k);
    return liczba;
}

unsigned int ustaw_bit_na_0(unsigned int liczba, int k)
{
    if (0 <= k && k <= maks_numer_bitu)
        liczba = liczba & ~(1 << k);
    return liczba;
}
```

Implementacja nowego typu danych - ZBIOR

ZADANIE: Zaimplementować w C nowy typ danych: zbiór liczb naturalnych o elementach nie większych niż pewna stała `MAX_ELEM` wraz z operacjami:

- 1 suma, przekrój i różnica zbiorów;
- 2 dopełnienie zbiorów;
- 3 usuwania wszystkich elementów i sprawdzania, czy zbiór jest pusty;
- 4 dodawania i usuwania pojedynczego elementu i sprawdzania, czy liczba należy do zbioru.

zbiory.h

```
// Plik naglowkowy: zbiory.h
//      Bitowa implementacja operacji na zbiorach
#include <limits.h>
#include <stdbool.h>

#define MAX_ELEM 10000000UL
#define BITOW_W_INT (CHAR_BIT*sizeof(unsigned int))
#define ROZMIAR (MAX_ELEM/BITOW_W_INT+1)

typedef unsigned long int Element; // po zmianie sprawdzić format drukowania
typedef unsigned int Zbior[ROZMIAR];

/***** PROTOTYPY FUNKCJI *****/

void suma_z(const Zbior z1, const Zbior z2, Zbior wynik);
void przekroj_z(const Zbior z1, const Zbior z2, Zbior wynik);
void roznica_z(const Zbior z1, const Zbior z2, Zbior wynik);
void dopelnienie_z(Zbior z);
bool czy_pusty_z(const Zbior z);
void wyczyszc_z(Zbior z);
void dodaj_e(Element e, Zbior z);
void usun_e(Element e, Zbior z);
bool element_z(Element e, const Zbior z);
```

zbiory.c I

```
#include "zbiory.h"

void suma_z(const Zbior z1,const Zbior z2,Zbior wynik)
{
    for (unsigned int i=0; i < ROZMIAR; ++i)
        wynik[i]=z1[i] | z2[i];
}

void przekroj_z(const Zbior z1,const Zbior z2,Zbior wynik)
{
    for (unsigned int i=0; i < ROZMIAR; ++i)
        wynik[i]=z1[i] & z2[i];
}

void roznica_z(const Zbior z1,const Zbior z2,Zbior wynik)
{
    for (unsigned int i=0; i < ROZMIAR; ++i)
        wynik[i]=z1[i] & ~z2[i];
}

void dopelnienie_z(Zbior z)
{
    for (unsigned int i=0; i < ROZMIAR; ++i)
        z[i]=~z[i];
}
```


zbiory.c II

```
bool czy_pusty_z(const Zbior z)
{
    for (unsigned int i=0; i < ROZMIAR; ++i)
        if (z[i] != 0)
            return false;
    return true;
}

void wyczyszc_z(Zbior z)
{
    for (unsigned int i=0; i < ROZMIAR; ++i)
        z[i]=0;
}

void dodaj_e(Element e,Zbior z)
{
    if (e <= MAX_ELEM)
        z[e/BITOW_W_INT] |= (1 << e % BITOW_W_INT);
}

void usun_e(Element e,Zbior z)
{
    if (e <= MAX_ELEM)
        z[e/BITOW_W_INT] &= ~(1 << e % BITOW_W_INT);
}
```

zbiory.c III

```
bool element_z(Element e,const Zbior z)
{
    return e <= MAX_ELEM ? z[e/BITOW_W_INT] & (1 << e % BITOW_W_INT) : false;
}
```

Sito Erastotenesa - użycie typu ZBIOR I

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

Element isqrt(Element n)
{
    Element i,kwadrat=1,np=3;

    if (n <= 3) return 1;
    for (i=1; kwadrat <= n-np; ++i,kwadrat+=np,np+=2);
    return i;
}

static Zbior sito;

int main(void)
{
    /* znajdowanie liczb pierwszych metoda sita Eratostenesa */
    int c=0;;
    Element pierwiastek=isqrt(MAX_ELEM);

    wyczyszc_z(sito); dopelnienie_z(sito); usun_e(1,sito);
    for (Element i=2; i <= MAX_ELEM; ++i)
        if (element_z(i,sito)) {
            printf(++c % 8 == 0 ? "%10lu\n" : "%10lu  ",i);
            if (i <= pierwiastek)
                for (Element j=i*i; j <= MAX_ELEM; j+=i)
                    if (element_z(j,sito))
```

Sito Erastotenesa - użycie typu ZBIOR II

```
        usun_e(j,sito);  
    }  
    putchar('\n');  
    return 0;  
}
```

Typy stałych całkowitych

Przyrostek	Stała dziesiętna	Stała ósemkowa lub szesnastkowa
brak	int	int
	long int	unsigned int
	long long int	long int
		unsigned long int
		long long int
		unsigned long long int
u lub U	unsigned int	unsigned int
	unsigned long int	unsigned long int
	unsigned long long int	unsigned long long int
l lub L	long int	long int
	long long int	unsigned long int
		long long int
		unsigned long long int
u lub U oraz l lub L	unsigned long int	unsigned long int
	unsigned long long int	unsigned long long int
ll lub LL	long long int	long long int
		unsigned long long int
u lub U oraz ll lub LL	unsigned long long int	unsigned long long int

Typy stałych zmiennopozycyjnych

- Stała zmiennopozycyjna musi zawierać część ułamkową (z kropką dziesiętną) lub wykładnik (zaczynający się od `E` lub `e`).
- Stała zmiennopozycyjna może się kończyć jednym ze znaków `f F l L`. Jeśli się nie kończy żadnym z tych znaków, to jest typu `double`.
- Stała zmiennopozycyjna kończąca się jednym ze znaków `f F` jest typu `float`.
- Stała zmiennopozycyjna kończąca się jednym ze znaków `l L` jest typu `long double`.

Konwersje typów w obliczeniach arytmetycznych

- Promocja typów całkowitych.
- Przekształcenia pomiędzy typami całkowitymi.
- Przekształcenia pomiędzy typami zmiennopozycyjnymi.
- Przekształcenia pomiędzy typami całkowitymi a zmiennopozycyjnymi.