

Wstęp do programowania w języku C

Marek Piotrów - Wykład 3
Operatory języka C. Funkcje i moduły.

18 października 2016

- Tablica jest podstawową złożoną strukturą danych służącą do przechowywania ciągu wartości tego samego typu. Tablice definiujemy podając *<typ>* *<identyfikator tablicy>* [*<liczba elementów>*], np. `int tab[12];`
- Do elementu tablicy odwołujemy się za pomocą indeksu, który może być dowolnym wyrażeniem, np. `tab[3] + tab[n-3]` . Początkowy element tablicy ma indeks 0.
- Takie tablice są jednowymiarowe. Tablicę dwuwymiarową (prostokątną) można zdefiniować jako tablicę tablic jednowymiarowych, czyli np. `float m[15][20];` oznacza tablicę składającą się z 15 wierszy po 20 wartości typu `float` w każdym.
- Tablice można inicjalizować przy ich definiowaniu, np. `int tab[6] = {2, 3, 5};`

Priorytety i łączność operatorów

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.	

Potęgowanie w C - nie ma operatora potęgowania

- Jest funkcja x^y w standardowej bibliotece matematycznej:
`double pow(double x, double y)`
- Są też funkcje `powf` dla typu `float` oraz `powl` dla typu `long double`, np. `float powf(float x, float y)`.
- **Jak napisać taką funkcję dla typu całkowitego?**
- **Jakiej efektywnej metody potęgowania użyć?**
- Najprościej: $a^{53} = a \cdot a \cdot a \cdot \dots \cdot a$ - 52 mnożenia;
- lub binarnie $a^{53} = a \cdot a^4 \cdot (a^4)^4 \cdot ((a^4)^4)^2$ - tylko 9 mnożeń.

Przykład 1 - prototypy funkcji

```
#include <stdio.h>
/* testowanie algorytmu szybkiego potegowania - wersja 1 */

long long int potega(int podstawa, int wykladnik);

int main(void)
{
    int i;

    for (i=0; i <= 15; ++i)
        printf("2^%2d = %10lld,   (-3)^%2d = %10lld\n", i, potega(2,i), i, potega(-3,i));
    return 0;
}

/* funkcja potega podnosi podstawe do potegi wykladnik, wykladnik >= 0 */
long long int potega(int podstawa, int wykladnik)
{
    long long int wynik, pot;

    wynik=1; pot=podstawa;
    for (int i=wykladnik; i > 0; i=i/2) {
        if (i%2 == 1) wynik=wynik*pot;
        pot=pot*pot;
    }
    return wynik;
}
```

Przykład 2 - przekazywanie parametrów

```
#include <stdio.h>
/* testowanie algorytmu szybkiego potegowania - wersja 2 */

long long int potega(int podstawa, int wykladnik);

int main(void)
{
    for (int i=0; i <= 15; ++i)
        printf("2^%2d = %10lld,   (-3)^%2d = %10lld\n",i,potega(2,i),i,potega(-3,i));
    return 0;
}

/* funkcja potega podnosi podstawe do potegi wykladnik, wykladnik >= 0 */
long long int potega(int podstawa, int wykladnik)
{
    long long int wynik = 1;

    for (long long int pot=podstawa ; wykladnik > 0; wykladnik=wykladnik/2) {
        if (wykladnik % 2 == 1) wynik=wynik*pot;
        pot=pot*pot;
    }
    return wynik;
}
```

Przykład 3 - deklaracje w starym stylu

```
#include <stdio.h>
/* testowanie algorytmu szybkiego potegowania - wersja w starym stylu  *
 * (styl ten trzeba znac ze wzgledow historycznych - nie nalezy go uzywac) */

long long int potega();

int main()
{
    int i;

    for (i=0; i <= 15; ++i)
        printf("2^%2d = %10lld,   (-3)^%2d = %10lld\n",i,potega(2,i),i,potega(-3,i));
    return 0;
}

/* funkcja potega podnosi podstawa do potegi wykladnik, wykladnik >= 0 */
long long int potega(podstawa,wykladnik)
{
    int podstawa,wykladnik;
    long long int wynik = 1;

    for (long long int pot=podstawa; wykladnik > 0; wykladnik=wykladnik/2) {
        if (wykladnik % 2 == 1) wynik*=pot;
        pot=pot*pot;
    }
    return wynik;
}
```

Przykład 4 - funkcja rekurencyjna

```
#include <stdio.h>
/* testowanie algorytmu szybkiego potegowania - wersja 1 */

long long int potega(long int podstawa, int wykladnik);

int main(void)
{
    for (int i=0; i <= 15; ++i)
        printf("2^%2d = %10lld,   (-3)^%2d = %10lld\n", i, potega(2,i), i, potega(-3,i));
    return 0;
}

/* funkcja potega podnosi wartosc a do potegi n, n >= 0 */
long long int potega(long int a, int n)
{
    if (n == 0) return 1;
    else
        if (n == 1) return a;
    else
        return n%2 == 0 ? potega(a*a, n/2) : a*potega(a*a, n/2);
}
```


Przykład 5 - Wypisywanie najdłuższego wiersza

ZADANIE: Napisz program (filtr) w języku C, który czyta tekst ze standardowego wejścia i wypisuje na standardowym wyjściu najdłuższy wiersz z wejścia. Można założyć, że maksymalna długość wiersza nie przekracza stałej $MAX = 1000$.

Naturalne pod-zadania:

- 1 Czytanie wiersza do tablicy znaków i obliczanie jego długości.
- 2 Kopiowanie wiersza (ciągu znaków) z jednej tablicy do drugiej.

Przykład 5 - moduł 1

```
#include <stdio.h>
#define MAX 1002 /* maksymalna dlugosc wiersza + 2 znaki na koniec wiersza i zerow */

// Przykład ten pokazuje jak moduly moga sie komunikowac przez
// przekazywanie argumentow do funkcji i korzystanie ze zwracanych wartosci.
// Jest to zalecany sposob takiej komunikacji.

int czytaj_wiersz(char wiersz[],int max);
void kopiuj(char cel[],char zrodlo[]);

int main(void)
{
    int dl,maxdl; /* dlugosc aktualnego wiersza, maksymalna dlugosc */
    char wiersz[MAX],maxwiersz[MAX]; /* aktualny wiersz, najdluzszy wiersz */

    maxdl=0;
    while ((dl=czytaj_wiersz(wiersz,MAX)) > 0)
        if (dl > maxdl) {
            maxdl=dl; kopiuj(maxwiersz,wiersz);
        }
    if (maxdl > 0) printf("%s",maxwiersz);
    return 0;
}
```

Przykład 5 - moduł 2

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

```
/* funkcja czytaj_wiersz: czyta wiersz znakow z wejscia lacznie z '\n',  
 * zwraca dlugosc wiersza lub 0 jesli jest to koniec danych */
```

```
int czytaj_wiersz(char wiersz[],int max)  
{  
    int c,i;  
  
    for (i=0; i < max-1 && (c=getchar()) != EOF; ++i)  
        if ((wiersz[i]=c) == '\n') {  
            ++i; break;  
        }  
    wiersz[i]='\0';  
    return i;  
}
```

```
/* funkcja kopiuj: kopiuje ciag znakow zakonczony znakiem '\0'  
 * z tablicy zt do tablicy dot */
```

```
void kopiuj(char dot[],char zt[])  
{  
    for (int i=0; (dot[i]=zt[i]) != '\0'; ++i) ;  
}
```

Przykład 6 - moduł 1

```
#include <stdio.h>
#define MAX 1002 /* maksymalna dlugosc wiersza + 2 znaki na koniec wiersza i zero */

// Przykład ten pokazuje jak moduły mogą się komunikować przez
// globalne struktury danych (tablice) - stosuje się je rzadko

int czytaj_wiersz(void);
void kopiuj(void);

int maxdl; /* maksymalna znaleziona dlugosc wiersza */
char wiersz[MAX]; /* aktualny wiersz */
char maxwiersz[MAX]; /* najdluzszy wiersz */

int main(void)
{
    int dl; /* dlugosc aktualnego wiersza */
    extern int maxdl;
    extern char maxwiersz[];

    maxdl=0;
    while ((dl=czytaj_wiersz()) > 0)
        if (dl > maxdl) {
            maxdl=dl; kopiuj();
        }
    if (maxdl > 0) printf("%s",maxwiersz);
    return 0;
}
```

Przykład 6 - moduł 2

```
#include <stdio.h>
#define MAX 1000 /* maksymalna dlugosc wiersza */

/* funkcja czytaj_wiersz: czyta wiersz znakow z wejscia lacznie z '\n' */
int czytaj_wiersz(void)
{
    int c,i;
    extern char wiersz[];

    for (i=0; i < MAX-1 && (c=getchar()) != EOF; ++i)
        if ((wiersz[i]=c) == '\n') {
            ++i; break;
        }
    wiersz[i]='\0';
    return i;
}

/* funkcja kopiuj: kopiuje ciag znakow zakonczony znakiem '\0' */
void kopiuj(void)
{
    extern char wiersz[],maxwiersz[];

    for (int i=0; (maxwiersz[i]=wiersz[i]) != '\0'; ++i) ;
}
```