Tablice i funkcje (w czasach zarazy, oczywiście)

Wojciech Myszka

Streszczenie

Do przemyślenia na kolejny tydzień

Spis treści

Wprowadzenie

Mając gotowy (i poprawnie działający) program, możemy postarać się wyodrębnić zasadniczą jego część i zaprogramować ją jako funkcję.

Dziś chciałem o dwu sprawach napisać:

1. Jak przerobić kawałek algorytmu na funkcję.
2. Jak to jest gdy chcemy aby parametrem funkcji była tablica albo inna funkcja.

Algorytm B

Posłużę się wersją opisaną na stronie: Algorytm B dla bystrzaków

```
#include <stdio.h>
int main(int argc, char **argv)
{
    int u, v, w, k;
    u = 258;
    v = 640;
// krok1:
    k = 0;
    while ( u % 2 == 0 && v % 2 == 0 )
    {
        u = u / 2;
        v = v / 2;
        k = k + 1;
    }
// krok2:
    if ( u % 2 == 1 )
    {
        t = -v;
        goto krok4;
    }
    else
        t = u;
krok3:
    t = t / 2;
krok4:
    if ( t % 2 == 0 )
        goto krok3;
// krok5:
    if ( t > 0 )
        u = t;
    else
        v = -t;
// krok6:
    t = u - v;
    if ( t != 0 )
        goto krok3;
    int wynik = 1;
    while ( k > 0 )
    {
        wynik = wynik * u;
        k = k - 1;
    }
    printf("NWD wynosi %d\n", u * wynik);
    return 0;
```

```
}
    while ( u % 2 == 0 && v % 2 == 0 )
    {
        u = u / 2;
        v = v / 2;
        k = k + 1;
    }
// krok2:
    if ( u % 2 == 1 )
    {
        t = -v;
        goto krok4;
    }
    else
        t = u;
krok3:
    t = t / 2;
krok4:
    if ( t % 2 == 0 )
        goto krok3;
// krok5:
    if ( t > 0 )
        u = t;
    else
        v = -t;
// krok6:
    t = u - v;
    if ( t != 0 )
        goto krok3;
    int wynik = 1;
    while ( k > 0 )
    {
        wynik = wynik * u;
        k = k - 1;
    }
    printf("NWD wynosi %d\n", u * wynik);
    return 0;
}
```

Żeby sobie ułatwić, komentarzami zaznaczyłem poszczególne kroki algorytmu.

Gdzie zaczyna się kod algorytmu?

To co chcemy zamknąć jako osobną funkcję „zaczyna się" tam gdzie zaznaczyłem **krok1**. A gdzie się kończy? Prosta odpowiedź jest taka, że tam gdzie jest drukowanie wyniku. Ale to nie jest dobra odpowiedź!

Nie interesuje nas funkcja, która na podstawie przedstawionych danych podaje wynik na ekran! My chcemy ten wynik wykorzystać do dalszych obliczeń. Czasami tylko upewnić się, że NWD[1] jest równe 1 (liczby są *wzajemnie pierwsze*), lub wykorzystać zebraną wiedzę do jakichś działań (uprościć ułamek).

Możemy więc powiedzieć, że Algorytm B kończy się tam gdzie jest już wyliczony `wynik`.

Jakie są dane do algorytmu?[2]

Danymi są zmienne `u` i `v` (typu całkowitego).

Czy funkcja powinna sprawdzać ich poprawność? Pytanie ma charakter filozoficzny. Z formalnego punktu widzenia `u` i `v` powinny być większe od zera.

Z drugiej strony zadanie możemy uogólnić. NWD dla dowolnej liczby różnej od zera i zera jest ta liczba.

Jaki powinien być wynik działania programu gdy obie wartości (`u` i `v`) równe są zeru?

Najprostsza odpowiedź może być taka: Program napisze błąd i skończy pracę. Ale znowu: nie interesują nas komunikaty pojawiające się na ekranie tylko informacja zwracana przez funkcję temu, kto ją wywołał...

Umówmy się, że w tym przypadku (oba argumenty mają wartość zero) funkcja zwraca zero.

Co jest wynikiem algorytmu

Zmienna `wynik` (typu `int`).

Funkcja

Teraz możemy zacząć pisać funkcję:

```
int nwd(int u, int v)
{
    int k, wynik;
    if ( u * v == 0 )   // gdy jedna lub obie równe zero
    {
        if ( u != 0 )
            wynik = u;
```

[1] Największy Wspólny Dzielnik.

[2] Zdecydować trzeba, które zmienne będą argumentami i jaki jest ich typ.

```
        else if ( v != 0 )
            wynik = v;
        else
            wynik = 0;
    }
    else
    {
        k = 0;
// krok1:
        while ( u % 2 == 0 && v % 2 == 0 )
        {
            u = u / 2;
            v = v / 2;
            k = k + 1;
        }
// krok2:
        if ( u % 2 == 1 )
        {
            t = -v;
            goto krok4;
        }
        else
            t = u;
krok3:
        t = t / 2;
krok4:
        if ( t % 2 == 0 )
            goto krok3;
// krok5:
        if ( t > 0 )
            u = t;
        else
            v = -t;
// krok6:
        t = u - v;
        if ( t != 0 )
            goto krok3;
        wynik = 1;
        while ( k > 0 )
        {
            wynik = wynik * u;
            k = k - 1;
        }
    }
```

```
    return wynik;
}
```

Natomiast program korzystający z tej funkcji może wygladać tak:

```
#include <stdio.h>
int main(int argc, char **argv)
{
    int a, b, x;
    a = 123;
    b = 256;
    x = nwd(a, b);
    if ( x > 0 )
        printf("NWD liczb %d i %d wynosi %d.\n", a, b, n);
    else
        printf("Zle dane!\n");
    return 0;
}
```

Metoda Połowienia

Skorzystam z algorytmu przedstawionego tu: Metoda połowienia dla bystrzaków

Funkcja nasza nazywać się będzie `polowienie()` i będzie zwracała wartość typu `double` będącą wartością miejsca zerowego. Nie będzie sprawdzała poprawności danych początkowych (to znaczy tego czy na początku $f(a) * f(b) < 0$).

Danymi wejściowymi funkcji będą wartości `A` i `B`. stałe `delta` i `epsilon` ustalone są na „sztywno”.

Funkcja może wyglądać jakoś tak:

```
double polowienie(double A, double B)
{
    double C;
    double epsilon = 1.e-4, delta = 1.e-4;
pocz:
    C = ( A + B ) / 2.;
    if ( fabs( sin(C) ) < delta )
    {
        return C;
    }
    else if ( sin(A) * sin(C) > 0 )
        A = C;
    else
        B = C;
    if ( fabs(A - B) > epsilon )
```

```
        goto pocz;
    return ( A + B ) / 2.;
}
```

Polecenie `return` tuż przed końcem funkcji zostało tak zaprogramowane, żeby „wykonać jeszcze jeden krok" połowienia. Przedział A–B jest już wystarczająco krótki, zwracamy informację o jego środku.

Wadą tak przedstawionej funkcji jest to, że **nie jest ona uniwersalna**: szuka miejsca zerowego funkcji sinus. Żeby zrobić funkcję uniwersalną potrzebne jest jeszcze coś.

Funkcja jako parametr funkcji

Do tej pory (Algorytm B) parametrami funkcji były zwykłe zmienne. A wyszukiwaniu binarnym parametrem była tablica jednowymiarowa. Tu parametrem będzie funkcja...

Jest to właściwie bardzo proste. Trzeba sobie tylko przypomnieć jak wygląda **prototyp** funkcji.

```
double polowienie (double A, double B, double f(double))
```

Trzecim parametrem funkcji jest dowolna funkcja o jednym argumencie typu `double` zwracająca wartości typu `double`. Wszędzie teraz trzeba zamienić `sin()` na `f()`. I gotowe.

Wywołanie funkcji będzie mogło wyglądać tak:

```
wynik = polowienie(2., 5., sin);
```

(szukamy miejsca zerowego funkcji `sin()` w przedziale $(2., 5.)$).

Wyszukiwanie binarne

Tu omówię algorytm ze strony: Wyszukiwanie binarne dla bystrzaków.

```
#include <stdio.h>
int main(int argc, char **argv)
{
    int t[] = {
        1, 10, 123, 200, 1000, 2000
    };
    int n = sizeof( t ) / sizeof( int );
    printf("n=%d\n", n);
    int x = -1;
    int a, b, c;
    a = 0;
    b = n - 1;
poczatek:
    c = ( a + b ) / 2;
    if ( t[c] == x )
```

```
        goto koniec;
    else if ( t[c] > x )
        b = c - 1;
    else
        a = c + 1;
    if ( a <= b )
        goto poczatek;
    printf("nie ma\n");
    return 1;
koniec:
    printf("wartosc jest w tablicy na miejscu %d\n", c);
    return 0;
}
```

Analiza przebiega podobnie jak w poprzednich przypadkach. Początek algorytmu jest tuż przed linią kodu oznaczoną etykietą `poczatek`.

Zastanówmy się co będzie **parametrami** funkcji:

- na pewno tablica z danymi `t` (i to jest najważniejsza cześć tego wykładu);
- jej rozmiar `n`;
- szukana wartość `x`.

Funkcja powinna **zwrócić informację**, na którym miejscu tablicy znajduje się szukany element lub informację, że elementu nie ma. W pierwszym przypadku będzie to liczba z zakresu od 0 do `n-1` (włącznie); w drugim — -1.

„Najtrudniejsza" część to zadeklarowanie tablicy jako parametru funkcji. Wszystko wyjaśni się gdy zaczniemy dyskutować wskaźniki.

Funkcja `binarne()`

Niech nasza funkcja nazywa się `binarne()`, a jej parametrami będą zmienna całkowita `x`, zmienna całkowita `n` i tablica `t` (na potrzeby tego przykładu `x` i `t` będą całkowite).

Do tej pory deklaracje parametrów funkcji niewiele różniły się od deklaracji zmiennych. Może uda się i teraz:

```
int binarne(int x, int n, int t[n])
{}
```

Zadeklarowałem tablicę `t` tak samo jak deklaruje się tablicę o zmiennej długości. Robiąc to trzeba zagwarantować, że długość tablicy `n` jest zadeklarowana wcześniej (czyli na lewo od `t[n]`).

Okaże się później (ale to powinno być w pewnym sensie oczywiste), że nie trzeba podawać długości tablicy w deklaracji[3] (i tak nie ma ona żadnego znaczeni), czyli:

[3]Natomiast **koniecznie** jednym z parametrów **musi** być `n` czyli rozmiar tablicy. Z różnych względów (o czym będzie mowa później) nie można zmierzyć rozmiaru tego parametru (`sizeof()`) wewnątrz funkcji.

```
int binarne(int x, int n, int t[])
{}
```

trzeba jednak zasygnalizować, że `t` to tablica (o czym świadczy para nawiasów kwadratowych po nazwie, tak jak w deklaracji tablicy gdy nie ma długości, ale jest inicjator).

Zmiennymi używanymi przez funkcję będą

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

Wyrzucamy z funkcji `printf()` i ostatecznie otrzymujemy:

```
int binarne(int x, int n, int t[n])
{
    int a, b, c;
    a = 0;
    b = n - 1;
poczatek:
    c = ( a + b ) / 2;
    if ( t[c] == x )
        goto koniec;
    else if ( t[c] > x )
        b = c - 1;
    else
        a = c + 1;
    if ( a <= b )
        goto poczatek;
    return -1;
koniec:
    return c;
}
```

Funkcja główna będzie mogła wyglądać jakoś tak:

```
#include <stdio.h>
int main(int argc, char **argv)
{
    int t[] = {
        1, 10, 123, 200, 1000, 2000
    };
    int n = sizeof( t ) / sizeof( int );
    printf("n=%d\n", n);
    int x = -1;
    int wynik = binarne(x, n, t)
```

Zwracam uwagę, że w wywołaniu funkcji `binarne` podajemy tylko i wyłącznie **nazwę** tablicy. Tak na prawdę to nić innego nie pasuje. Wpisanie `t[`α`]` (gdzie α to jakaś stałą, zmienna lub

wyrażenie typu `int`) nie ma sensu bo wartość elementu tablicy o indeksie α. A `t[]` nie ma najmniejszego sensu[4].

I zakończenie programu:

```
    if (wynik > 0)
        printf("Szukana wartość %d znajduje się na miejscu %d w tablicy\n", x, wynik);
    else
        printf("Szukana wartość %d nie zostałą znaleziona w tablicy\n", x);
    return 0;
}
```

Plik w formacie...

...PDF jest również dostępny.

[4]Może pojawić się jedynie w deklaracji tablicy, dla której nie podajemy długości.