Wojciech Myszka

Laboratorium 4: Funkcje. Metoda połowienia

2021-03-28 08:47:02 +0200

1. Metoda połowienia

1.1. Wprowadzenie

1.1.1. Liczby zmiennoprzecinkowe

Dotychczas wykonywane zadania operowały na liczbach całkowitych. Dziś pierwsze zajęcia gdzie skorzystamy z bardziej zaawansowanych typów liczbowych — zmiennoprzecinkowych.

Typy takie są dwa (no, właściwie trzy):

1. **float** — podstawowy, trzydziestodwubitowy o niezbyt wielkim zakresie i niezbyt wielkiej precyzji.
2. **double** — sześćdziesięcioczterobitowy.
3. **long double** o długości 128 bitów (raczej rzadko stosowany).

Pewne pojęcie o błędach wprowadzanych podczas długotrwałych (i wrednych) obliczeń pokazuje poniższy program[1]:

```
#include <stdio.h>
int
main ()
{
    float s;
    double d;
    long double e;
    int i;
    s = d = e = 0.5;
    for ( i = 1;  i <= 100; i++ )
    {
```

[1] W programie są celowo umieszczone błędy!

```
        s = 3.8F * s * ( 1.F - s );
        d = 3.8 * d * ( 1. - d );
        e = 3.8L * e * ( 1.L - e );
        if ( i % 10 == 0 )
            printf ("%5i %11.5f %11.5lf %11.5Lf" \
                    "%11.2lf %11.2Lf \n", \
                    i, s, d, e, \
                    ( (double) s - d ) / d * 100, \
                    ( (long double) s - e ) / e * 100);
    }
    return 0;
}
```

(**Uwaga:** W powyższym listingu zwracam uwagę na jedyną instrukcję drukowania. Jest ona długa i żeby lepiej mieściła się na stronie podzieliłem ją na krótsze fragmenty. Na końcu każdej linii występuję znak \, który informuje, że poniżej zawartość jest kontynuowana.)

Program wylicza kolejne wartości ciągu:

$$x_{n+1} = ax_n(1 - x_n) \tag{1}$$

w pojedynczej, podwójnej i rozszerzonej precyzji, drukuje co dziesiąty wynik dodatkowo podając procentowy błąd wartości wyliczonej w pojedynczej precyzji w porównaniu do wyniki w uzyskanego z precyzji wyższej.

Zwracam uwagę na konsekwentne stosowanie przyrostków określających typ stałej (3.8 vs 3.8F vs 3.8L) oraz operatora rzutowania (konwersji:
((**double**) s − d) / d ∗ 100).

To co nalezy zapamiętać, to to, że liczby zmiennoprzecinkowe to zupełnie co innego niż liczby rzeczywiste znane z analizy matematycznej!

1.1.2. Funkcje (standardowe)

Drugą „nowością" są funkcje. Bez funkcji nie ma języka C. Natomiast samo pojęcie funkcji w językach programowania zostało zapożyczone z matematyki. Bardzo wiele praktycznych obliczeń wymaga korzystania, na przykład, z wartości funkcji trygonometrycznych $\sin(x)$ czy $\log(x)$.

Nie zastanawiając się jaki jest sposób (algorytm) wyliczania wartości $\cos(x)$ śmiało wpisujemy takie wyrażenie zakładając, że prowadzący obliczenia znajdzie jakąś metodę żeby tę wartość wyliczyć.

Podobne jest w przypadku języków programowania. Do dyspozycji mamy różne funkcje, w tym trygonometryczne. Nie musimy się martwić o to w jaki sposób są wyliczane. Ktoś za nas je już zaprogramował i umieścił w „bibliotece" (matematycznej) funkcji.

Aby móc z funkcji trygonometrycznych (i innych matematycznych) skorzystać trzeba wykonać pewne „magiczne" czynności:

1. Po pierwsze trzeba umieścić wiersz o postaci

   ```
   #include <math.h>
   ```

   gdzieś obok (przed albo po) #**include** <stdio.h>. Ponieważ, podobnie jak każda zmienna przed pierwszym „użyciem" musi być zadeklarowana — zadeklarowane muszą być również funkcje. Linia ta wczytuje plik zawierający deklaracje podstawowych funkcji matematycznych.
   Ale to nie wszystko.
2. Wszystkie funkcje dodatkowe przechowywane są w bibliotekach. Musimy komputerowi powiedzieć, aby podczas budowania naszego programu zajrzał do właściwych bibliotek. Z menu wybieramy Projekt → Właściwości. W otwartym oknie klikamy w zakładkę „Buduj" i w okienku opisanym Build/Buduj do istniejącej tam zawartości (`gcc -Wall -o "%e" "%f"`) na końcu dodajemy `␣-lm` i mamy: `gcc␣-Wall␣-o␣"%e"␣"%f"␣-lm` Na koniec klikamy OK.

1.1.3. Funkcje (użytkownika)

Korzystanie z funkcji nie jest wcale skomplikowane. Zwracam uwagę, że najprostszy program w C:

```
#include <stdio.h>

int main(int argc, char **argv)
{
        printf("Ala ma kota");
        return 0;
}
```

składa się z jednej funkcji o nazwie main. Przy czym, użytkownik nigdy jej nie używa (wywołuje). Używa jej system operacyjny: aby uruchomić program wywołuje funkcję main.

Stworzenie nowej funkcji nie jest specjalnie skomplikowane. Trzeba:

1. Wybrać dla niej nazwę.
2. Zdecydować ile parametrów wejściowych będzie miała funkcja (funkcja ilu zmiennych). Tu podejmujemy również decyzję o ich typie.
3. Zdecydować jakiego typu wartości zwracać będzie funkcja.
4. Opracować algorytm działania funkcji (jej program).

Nasza funkcja będzie się nazywała „mój sinus" (my_sin). Będzie miała jedną zmienną x typu **double**. Zwracać będzie wartości typu **double**. A jej algorytm będzie dosyć prosty:

$$\text{my\_sin}(x) = \sin(x/180 * \pi) \tag{2}$$

czyli argument jest konwertowany ze stopni na radiany i dla tej wartości wyliczana jest wartość standardowej funkcji sinus.

```
double my_sin(double x)
{
    return sin(x / 180. * 3.14);
}
```

Z funkcji można skorzystać w następujący sposób:

```
    x = 90.
    printf("sin(%f)=%f\n", x, my_sin(x));
```

Polecenie **return** odgrywa bardzo ważną rolę: mówi, która z wyliczonych w funkcji wartości[2] ma być użyta gdy wyrażenie my_sin(x) pojawi się gdzieś po prawej strony znaku równości. Właśnie ta, która pojawi się jako argument plecenia **return**.

Kompletny program[3] wygląda tak:

```
#include <math.h>
#include <stdio.h>
double my_sin(double x)
{
    return sin(x / 180. * 3.14);
}

int main(int argc, char **argv)
{
    double x;
    x = 90.;
    printf( "sin(%f)=%f\n", x, my_sin(x) );
    return 0;
}
```

Ponieważ każdy obiekt (zmienna, funkcja,...) musi być zadeklarowana przed pierwszym użyciem — funkcja my_sin() zadeklarowana jest **przed** funkcją main()!

Jeżeli funkcja wywołuje podczas obliczeń samą siebie — nazywa się rekurencyjną.

1.2. Metoda połowienia: postawienie problemu

1. Zadanie jest proste. Mamy funkcję $f(x)$ ciągłą i taką, że na końcach pewnego przedziału $[A, B]$ $f(A)f(B) < 0$. Zatem, funkcja ta zmienia znak w przedziale

[2] No dobra! Nasza funkcja wylicza tylko jedną. Ale i tak!

[3] Można z niego skorzystać, ale zawiera błędy!

$[A, B]$ (co najmniej raz) ma zatem (co najmniej jedno) miejsce zerowe w tym przedziale.
2. Przedział $[A, B]$ dzielimy na pół (wyznaczając odpowiednio punkt C).
3. Odrzucamy ten z przedziałów $[A, C]$, $[C, B]$ w którym funkcja nie zmienia znaku (to znaczy ma ten sam znak na końcach przedziału).
4. Postępowanie prowadzimy tak długo, aż długość przedziału $[A, B]$ będzie mniejsza od zadanej liczby ε.

Uwaga: Obliczenia najprościej wykonać dla funkcji $\sin()$[4] wybierając $0 < A < 3$ i $3{,}5 < B < 6$ albo albo my_sin() (wybierając wartości z zakresu, na przykład $0 < A < 30$ i $100 < B < 240$).

Powyższe zadanie można również zaprogramować korzystając z rekurencji!

Zadanie do zrobienia

— Zaprogramować metodę połowienia.
— Zaprogramować metodę połowienia wykorzystując funkcje (minimum dwie: pierwsza to funkcja, której miejsca zerowego szukamy, druga — metoda połowienia).
— Zaprogramować zadanie metodą rekurencyjną.

2. Wersja PDF tego dokumentu...

...pod adresem.

Wersja: 59 z drobnymi modyfikacjami! data ostatniej modyfikacji 2021-03-28 08:47:02 +0200

[4] Ale pomysły ambitniejsze będą mile widziane!