

Funkcje

Pewne zestawy operacji, zależne od zmiennych, możemy zebrać w grupki (funkcje) i wywoływać jak `circle` i `line`. Przykład z poprzedniego paragrafu możemy zamknąć w funkcji:

```
void obrazek(int h, int r)
{
    line( 10, 0, 0, h);
    line( 10, 0, 2*r, h);
    circle( 10+r, h, r);
}
```

Pierwsza linia deklaruje funkcję, która jest zależna od dwóch parametrów: `h,r`. Taką funkcję, możemy wywołać dla przykładu tak: `obrazek(100,50);`. Spowoduje to wykonanie powyższych trzech operacji przy $h = 100$ i $r = 50$.

****Pamiętaj:** Nową funkcję napisz przed funkcją `main`**

W funkcji `main` wywołujemy funkcję `obrazek`, tak jak `circle` czy `line`:

```
void main()
{
    graphics(200,200);
    obrazek(100,50);
    wait();
}
```

Ćwiczenia

Napisz i wywołaj dowolne dwie z poniższych funkcji: - `prostokat(x,y,a,b)` — Narysuj prostokąt o bokach `a` i `b` i środku w (x, y) - `kwadrat(x,y,r)` — Narysuj kwadrat o boku $2r$ i wpisane koło o promieniu r . - `ludzik(x,y,h)` — Narysuj ludzika wysokości h i środku głowy w (x, y) - `olimpiada(x,y)` — Narysuj koła olimpijskie o środku w (x, y) - `* okno(a)` — Używając funkcji do rysowania prostokąta narysuj okno o boku a .

Trochę więcej szczegółów

Omówmy pewne rzeczy trochę dokładniej.

Typy

W C i C++ *musimy* deklarować zmienne, tzn. powiedzieć, jakich będziemy używać zmiennych i jakich one będą typów. Deklaracje piszemy 'typ zmienna1,zmienna2,...;'. Najważniejsze typy to: - `int` — Liczba całkowita (32-bitowa, od -2^{31} do 2^{31}) - `float` — Liczba zmiennie-przecinkowa. Może opisywać ułamki dziesiętne z ok. 7 cyframi znaczącymi (32-bity) - `double` — Liczba zmiennie-przecinkowa. Ma 16 cyfr znaczących (64-bity)

****Pamiętaj:** Jeśli używasz liczb rzeczywistych (a nie całkowitych), używaj typu `double**`.

Pierwszym przykładem niech będzie:

```
double a;
a = 0;
while (a < 2*3.14)
{
    circle(a * 100, sin(a) * 100 + 100, 3);
    a = a + 0.001;
}
```

Ten program narysuj wykres sinusa przeskalowany o 100, za pomocą kółek o promieniu 3.

Ćwiczenia

Używając analogicznej pętli, wykonaj dowolne dwa z poniższych zadań. - Narysuj wykres a^2 . - Narysuj punkty o współrzędnych $x = 100 \sin a + 100$ i $y = 100 \cos a + 100$. - Narysuj punkty o współrzędnych $x = 100 \sin a \cos 4a + 100$ i $y = 100 \cos a \cos 4a + 100$. - Narysuj punkty o współrzędnych $x = 100r \sin a + 100$ i $y = 100r \cos a + 100$, gdzie $r = \frac{\cos a + 2}{3}$ (niech r będzie kolejną zmienną).

Typy — pułapki

Ważne, by pamiętać, że liczby bez przecinka dziesiętnego, są uważane za całkowite, tzn. wykonywane są na nich działania jak dla liczb całkowitych. Dlatego $1/4$ da jako wynik 0! Bo wynik 0.25 zostanie obcięty do liczby całkowitej. Żeby tego uniknąć, możemy napisać $1.0/4$ lub jeszcze lepiej $1.0/4.0$. Możemy także bezpośrednio ‘rzutować’ zmienne z `int` na `double` pisząc: `(double) zmienna`.

****Pamiętaj:** Wszędzie, gdzie robisz obliczenia, używaj `double**`. Unikaj mieszania liczb całkowitych i zmiennie-przecinkowych. Nigdy nie pisz ułamków jako $1/3$

Ćwiczenia

Przeanalizuj (i przetestuj) wynik tego programu. Które linie nie dadzą pożądanego efektu?

```
double a;
a = 0;
while (a < 2)
{
    circle(a * 100, sin( a * 3.14 ) * 100 + 100, 3);
    circle(a * 100, sin( a * (314 / 100) ) * 100 + 100, 3);
    circle(a * 100, sin( (a * 314) / 100 ) * 100 + 100, 3);
    a = a + 0.001;
}
```

Funkcje po raz drugi

Zestawy operacji, które powtarzamy w programie wielokrotnie, możemy zamknąć w funkcjach. Taka funkcja „połyka” parametry i coś z nimi robi. Dla przykładu:

```
void kreski(int n, double r)
{
    int i;
    i = 0;
    while (i < n)
    {
```

```
        line(i, 0, i, r * i);
        i = i + 1;
    }
}
```

W pierwszej linii mówimy: - jak nazywa się funkcja — **kreski** - jakie ma parametry — `n` typu `int` i `r` typu `double` - jakiego typu zwraca wartość — w naszym wypadku `void` oznacza, że nic nie zwraca

Gdy gdziekolwiek w funkcji `main` użyjemy wywołania `kreski(20, 0.4)`; jako efekt działania funkcji otrzymamy 20 pionowych kresek o długości od 0 do $0.4 \cdot 19$ (dlaczego 19 a nie 20?).

Taką funkcję możemy wykonać wielokrotnie dla różnych wartości `n` i `r`:

```
void main()
{
    graphics(200,200);

    kreski( 10, 1.000);
    kreski( 20, 0.500);
    kreski( 30, 0.333);
    kreski( 40, 0.250);

    wait();
}
```

Ćwiczenia

Napisz i wywołaj dwie spośród niżej wymienionych funkcji. - Funkcję, która narysuje ludzika wysokości `h` i środka głowy w `(x, y)`. - Funkcję, która w pętli narysuje tłum (używając poprzedniej funkcji). - Funkcję, która narysuje `n` kółek w punkcie `(x,y)` o coraz większych promieniach. - * Funkcję, która narysuje wielokąt foremny o `n` bokach.

Instrukcja warunkowa

Kolejnym blokiem składowym programowania, jest instrukcja warunkowa. Sprawdza ona warunek i wykonuje pewną część kodu, tylko gdy warunek jest

spełniony.

```
x = 2.0;
if ( x > 0 ) {
    y = sqrt(x);
} else {
    y = 0;
}
```

Instrukcja ta sprawdza czy $x > 0$ i jeśli jest to prawdą, to wstawia \sqrt{x} do zmiennej y . Gdy warunek nie jest spełniony, wykonywana jest część po **else**, więc wstawiane jest 0 do y . W ten sposób możemy zabezpieczyć się na przykład przed niemożliwymi obliczeniami, albo uzależnić działanie programu od jakis wartości.

Zobaczmy prosty przykład:

```
double a;
a = 0;
while (a < 2*3.14)
{
    if (a < 2) {
        circle(sin(a) * 100 + 100, cos(a) * 100 + 100, 5);
    } else {
        circle(sin(a) * 100 + 100, cos(a) * 100 + 100, 10);
    }
    a = a + 0.001;
}
```

Gdyby nie instrukcja **if**, ten program narysował by koło z małych kółek. Teraz, gdy kąt a przekroczy 2 radiany zmieni promień kółeczka z 5 na 10

Ćwiczenia

Napisz program który: - Dla parametru w rysuje wykres $x^2 - w$, przeskalowany o 100 w obu kierunkach i przesunięty na środek (patrz poprzedni przykład). - Wyrysuje większe kółka w miejscach przecięcia wykresu z osią x (jeżeli przecina). - Zmodyfikuj program by działał dla dowolnych a , b , c i funkcji $ax^2 + bx + c$.