C - 5 - Funkcje

5.1 Wbudowane funkcje z biblioteki math.h

Otwórz (np. w Notatniku) plik:

C:\Program Files\Microsoft Visual Studio\VC98\Include\math.h

Obejrzyj go dokładnie, ale go nie zmieniaj! Mniej więcej w ¼ długości pliku znajdują się prototypy funkcji matematycznych:

```
int      abs(int);                  - wartość absolutna dla liczb całkowitych
double   acos(double);              - arccos
double   asin(double);              - arcsin
double   atan(double);              - arctg
double   atan2(double, double);     - przyjmuje wartość x oraz y, oblicza arctg(x/y)
double   cos(double);               - cosinus
double   cosh(double);              - hiperboliczny cosinus
double   exp(double);               - e^x
double   fabs(double);              - wartość absolutna dla liczb zmiennoprzecinkowych
double   fmod(double, double);      - wartość modulo dla liczb zmiennoprzecinkowych (dla całkowitych: %)
long     labs(long);                - wartość absolutna dla liczb typu long
double   log(double);               - logarytm naturalny
double   log10(double);             - logarytm dziesiętny
double   pow(double, double);       - potęga
double   sin(double);               - sinus
double   sinh(double);              - hiperboliczny sinus
double   tan(double);               - tangens
double   tanh(double);              - hiperboliczny tangens
double   sqrt(double);              - pierwiastek
```

Typ po lewej oznacza zwracaną wartość, typ (lub typy) w nawiasach oznacza przyjmowany parametr/parametry.

Zadanie 0. Korzystając z funkcji **asin** wyznacz i wyświetl wartość liczby π oraz π/2.

Poniższy kod oblicza całkę z funkcji ln(x) na podanym przedziale [a,b] metodą trapezów:

```
#include "stdafx.h"
#include "math.h"

int main(int argc, char* argv[])
{
        double a = 1,b = 2;
        double krok = 0.1;
        double calka = 0;
        double dokladne;

        for(double i=a;i<b;i=i+krok)
                calka = calka + krok*0.5*(log(i)+log(i+krok));

        printf("Calka z funkcji ln(x) na przedziale: [%.2lf,%.2lf] wynosi: %3.6lf\n",a,b,calka);
        dokladne = ( b*log(b)-b ) - ( a*log(a)-a );    // tu korzystamy ze wzoru na całkę oznaczoną z ln(x)
        printf("Dokladne rozwiazanie wynosi: %lf\n",dokladne);
        printf("Blad pomiaru wynosi: %lf\n",fabs(calka - dokladne));

        return 0;
}
```

Zadanie 1. Przerób powyższy program tak, by użytkownik mógł podać z klawiatury krańce przedziału oraz krok całkowania.

Zadanie 2. Przerób powyższy program tak, by użytkownik podawał z klawiatury krańce przedziału oraz dopuszczalny błąd – program ma próbować rozwiązania dla coraz mniejszych kroków aż znajdzie taki krok, dla którego błąd jest już dopuszczalny przez użytkownika. Za każdą próbą należy wypisać krok oraz błąd pomiaru. Przyda się pętla **while**.

Poniższy kod oblicza wartość funkcji e^x w danym punkcie za pomocą szeregu Taylora. Przypomnienie:

$$f(x) = \left(\sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k\right) + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1}$$

Dla wygody pominiemy resztę Lagrange'a.

```
#include "stdafx.h"
#include "math.h"

int main(int argc, char* argv[])
{
	double punkt = 2.0;
	double dokladnie = exp(punkt);
	double szereg = 0;
	double liczbaKrokow = 15;
	double x0 = 0;


	for(int i=0;i<=liczbaKrokow;i++){
		int silnia = 1;
		for(int j=1;j<=i;j++) silnia = silnia * j;
		szereg = szereg + exp(x0)*pow(punkt-x0,(double)i)/(double)silnia;
	}

	printf("Wynik z szeregu: %.6lf\n",szereg);
	printf("Wynik dokładny:  %.6lf\n",dokladnie);
	printf("Blad pomiaru:    %.6lf\n",fabs(szereg-dokladnie));

	return 0;
}
```

Zadanie 3. Przerób powyższy program tak, by użytkownik mógł podać z klawiatury punkt, x0 oraz liczbę kroków.

Obliczenie wartości funkcji e^x było łatwe ze względu na wzór jej pochodnej.

Zadanie 4. Przerób powyższy program tak, by można było obliczyć wartość funkcji:
a) sin(x)
b) sqrt(x)
Wskazówka: dla sinusa przydadzą się **if**y badające podzielność przez 4

Zadanie 5. Przerób powyższy program tak, by użytkownik mógł podać z klawiatury punkt, x0 oraz dopuszczalny błąd. Program powinien liczyć szereg tak długo, aż błąd pomiaru będzie mniejszy od błędu podanego przez użytkownika. Przyda się pętla **while**.

Poniższy kod obraca punkt na płaszczyźnie dwuwymiarowej o dany kąt korzystając z macierzy obrotu:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

Funkcje trygonometryczne z biblioteki **math.h** przyjmują argumenty w radianach, trzeba więc zastosować przelicznik:

radiany = stopnie * π / 180

```
#include "stdafx.h"
#include "math.h"

#define M_PI 3.1415926

int main(int argc, char* argv[])
{
	double x = 0.5;
	double y = 2.4;
	double kat = 90;
	double radian = kat*M_PI/180.0;

	printf("Przed obrotem:		(%lf,%lf)\n",x,y);

	double x1,y1;
	x1 = cos(radian)*x  - sin(radian)*y;
	y1 = sin(radian)*x  + cos(radian)*y;
	printf("Po obrocie o %.2lf stopni: (%lf,%lf)\n",kat,x1,y1);

	return 0;
}
```

Zadanie 6. Przerób powyższy program tak, by użytkownik mógł podać z klawiatury dwa punkty (czyli odcinek) oraz kąt obrotu.

Zadanie 7. Przerób powyższy program tak, by użytkownik mógł podać z klawiatury punkt lub odcinek w płaszczyźnie trójwymiarowej oraz dwa kąty obrotu. Program powinien odpowiednio obrócić podany punkt/odcinek.

5.2 Własne funkcje/procedury

Większości ludzi wydaje się, że funkcja musi koniecznie pobierać jakieś argumenty i musi też zwracać wartość. W rzeczywistości jest to tylko jeden z rodzajów funkcji. Przykład:

```
#include "stdafx.h"

int pomnozRazyDwa(int x){
        int wynik = x*2;
        return wynik;
}

int main(int argc, char* argv[]){
        int x = 7;
        int y = pomnozRazyDwa(x);
        printf("%d\n",y);
        return 0;
}
```

To jest nasza funkcja, która mnoży podany argument przez dwa.

Czarny int oznacza, że zwracamy liczbę całkowitą.
Niebieski int oznacza, że pobrany argument jest liczbą całkowitą.

Zadanie 8. Sprawdź, że ...

a) ... funkcja w nieco skróconej postaci:

```
int pomnozRazyDwa(int x){
        return x*2;
}
```

zadziała tak samo dobrze.

b) ... x*2 można otoczyć nawiasami i nic złego się nie stanie.

c) ... zawartość funkcji **main** może wyglądać następująco:

```
int x = 7;
printf("%d\n",pomnozRazyDwa(x));
return 0;
```

d) ... jeżeli po powyższej przeróbce w funkcji **main** nazwiesz **x** jakoś inaczej (np. **z**), to program nadal zadziała – funkcja interesuje się tylko tym, by otrzymać liczbę całkowitą i absolutnie nie obchodzi jej, jak nazwiesz tę zmienną w **main**.

Studenci prawie zawsze myślą, że jeżeli funkcja przyjmuje parametr o nazwie x, to w main też trzeba jej przekazać zmienną o nazwie x, co jest totalną bzdurą :(

e) ... zawartość **main** może wyglądać następująco:

```
printf("%d\n",pomnozRazyDwa(7));
return 0;
```

Zadanie 9. Napisz jakąś inną funkcję, która pobiera argument jednego typu, a zwraca argument innego typu. Funkcja może robić coś zupełnie trywialnego.
Wywołaj nową funkcję w swoim programie

Zadanie 10. Napisz funkcję, która pobiera liczbę całkowitą i zwraca wartość boolowską – funkcja może np. odpowiadać na pytanie, czy liczba jest parzysta (**true**), czy nie (**false**).
Wywołaj funkcję w swoim programie - przydadzą się **if**y !!!

Zadanie 11. Napisz funkcję, która pobiera liczbę całkowitą **n** i zwraca wartość **n!** – w ciele funkcji skorzystaj z pętli **for**.

Funkcja może oczywiście pobierać wiele argumentów – oddzielamy je przecinkami:

```
int ileMiejscZerowych(double a,double b,double c){
        double delta = b*b-4*a*c;
        if(delta > 0) return 2;
        else if(delta ==0) return 1;
        else if(delta < 0) return 0;
}

przykład użycia:
  printf("%d\n",ileMiejscZerowych(1.0,-3.0,1.0));
```

Mimo że trzy argumenty są tego samego typu, trzeba ten typ napisać przed każdą zmienną. Częstym błędem jest pisanie: int ileMiejscZerowych(double a,b,c){ ....

Jeszcze częstszym błędem jest zapominanie o zwracaniu wartości.

Zadanie 12. Sprawdź, że jeżeli z powyższej funkcji usuniesz oba słowa **else**, to funkcja będzie działała tak samo dobrze. Dzieje się tak dlatego, że po wykonaniu któregoś **if**a (czyli po **return**) program wyskakuje z funkcji więc i tak nie doczyta, czy dalej jest **if**, czy **else if**.

Zadanie 13. Sprawdź, że jeżeli wykasujesz ostatni **if** i zostawisz tylko **return 0;** to program zadziała tak samo dobrze – skoro delta nie wpadła w poprzednie **if**y, więc musi być mniejsza od zera i nie trzeba tego sprawdzać trzecim **if**em, można od razu śmiało zwrócić zero.

Zadanie 14. Napisz funkcję, która pobiera dwa lub więcej argumentów, z których każdy jest innego typu. Funkcja może robić coś trywialnego, ale musi jakoś wykorzystywać wszystkie argumenty.

Studenci zwykle nie rozróżniają pojęcia „zwrócić wartość" (np. do jakiejś zmiennej) oraz „wyświetlić wartość" – z reguły wybierają to drugie, ponieważ nie rozumieją instrukcji return.

Żadna z powyższych przykładowych funkcji nie wyświetlała wartości.

Podobnie jak zmienne, funkcje nie mogą nazywać się w pełni dowolnie. Nazwa funkcji może zawierać litery, cyfry oraz znak podkreślenia _, ale nie może zaczynać się od cyfry. Częstym błędem w nazywaniu funkcji jest pisanie spacji i używanie polskich liter. Dobrą praktyką jest nadawanie funkcji takiej nazwy, by było wiadomo co ta funkcja robi.

Funkcja w C/C++ może zwrócić tylko pojedyńczy rezultat. Jeżeli potrzebna jest para liczb, to można użyć dwóch funkcji lub wywołać funkcję dwukrotnie. Nie możemy wymagać od funkcji, by zwróciła np. tablicę wielowymiarową. Gdy już poznamy struktury, będziemy mogli to obejść :)

Funkcje mogą **coś zwracać i nic nie pobierać**. Przykład:

```
#include "stdafx.h"

char pobierzLiterke(void){
        printf("Podaj literke: ");
        char x = getchar();
        return x;
}

int main(int argc, char* argv[]){

        char literka = pobierzLiterke();
        printf("Podales: %c\n",literka);

        return 0;
}
```

void (ang: pustka, próżnia) oznacza, że funkcja nie pobiera żadnego argumentu. Sprawdź, że jeżeli wykasujesz to słowo i zostawisz puste nawiasy, to program zadziała tak samo dobrze. Pisanie void jest jednak bardziej eleganckie, niż jego opuszczanie.

Tu nic nie ma w nawiasach. Sprawdź, że jeżeli wpiszesz w nawiasach słowo void, to program nie zadziała - void piszemy w nagłówku funkcji, a nie podczas jej wywołania !!!

Funkcja może **coś pobierać i nic nie zwracać**. Przykład:

```
#include "stdafx.h"

void wyswietlMacierz(int m[4][4]){

        for(int i=0;i<4;i++){
                for(int j=0;j<4;j++) printf("%d ",m[i][j]);
                printf("\n");
        }

}

int main(int argc, char* argv[]){

        int macierz[4][4] = {{1,2,3,4},{0,9,8,7},{7,6,5,4},{1,1,1,2}};

        wyswietlMacierz(macierz);

        return 0;
}
```

Skoro funkcja zwraca „nic", to jest typu void. Zauważ, że w tej funkcji nigdzie nie występuje instrukcja return.

Przy okazji mamy tu przykład funkcji, która pobiera tablicę :)

Zauważ, że w main do funkcji wyswietlMacierz przekazujemy tylko nazwę zmiennej „macierzowej" – bez jej wymiarów.

Skoro funkcja nic nie zwraca, to nie przypiszemy do żadnej zmiennej jej wartości, tak jak to było w poprzednich przykładach. Nie napiszemy więc: jakasZmienna = wyswietlMacierz(macierz);
Ale 90% studentów i tak by to napisało :(

W Pascalu funkcje, które nic nie zwracały, były nazywane *procedurami*. My będziemy nazywać je *funkcjami*. W Javie zarówno funkcje jak i procedury są nazywane *metodami*.

W razie potrzeby nagłego zakończenia działania funkcji nic nie zwracającej, można z niej „wyskoczyć" przy pomocy instrukcji **return**, np:

```
void wyswietlMacierz(int m[4][4]){
        for(int i=0;i<4;i++){
                for(int j=0;j<4;j++){
                        if(m[i][j] == 0) return ;
                        printf("%d ",m[i][j]);
                }
                printf("\n");
        }
}
```

Chcemy wyświetlać zawartość macierzy aż do pierwszego napotkanego zera. W przypadku zera – wyskakujemy instrukcją return ;
Ten return jest „pusty", ponieważ funkcja jest typu void.

Istnieją też funkcje, które nic nie pobierają i nic nie zwracają. Służą często do wyświetlania menu lub innych komunikatów. Oto większy przykład:

```
#include "stdafx.h"
#include "conio.h"

// nic nie pobiera i nic nie zwraca
void menu(void){
	printf("\n\nM E N U \n");
	printf("0. Wyjscie z programu\n");
	printf("1. Liczba pierwiastkow r-nia kwadratowego - sposob 1\n");
	printf("2. Liczba pierwiastkow r-nia kwadratowego - sposob 2\n");
}

// nic nie pobiera i cos zwraca
char pobierzOpcje(void){
	printf("Co wybierasz? ");
	char x = getch();
	return x;
}

// cos pobiera i cos zwraca
int liczbaRozwiazanSposob1(double a,double b,double c){
	double delta = b*b-4*a*c;
	if(delta > 0) return 2;
	if(delta ==0) return 1;
	return 0;
}

// cos pobiera i nic nie zwraca
void liczbaRozwiazanSposob2(double a,double b,double c){
	double delta = b*b-4*a*c;
	if(delta > 0) printf("2 rozwiazania\n");
	else if(delta ==0) printf("1 rozwiazanie\n");
	else printf("0 rozwiazan\n");
}

int main(int argc, char* argv[]){

	while(true){

		menu();
		char odp = pobierzOpcje();

		if(odp == '0') break;
		else if(odp == '1' || odp == '2'){
			double x,y,z;
			printf("\nSposob %c - podaj trzy liczby oddzielone spacją: ",odp);
			scanf("%lf %lf %lf",&x,&y,&z);
			if(odp == '1')  printf("Liczba rozwiazan: %d",liczbaRozwiazanSposob1(x,y,z));
			else liczbaRozwiazanSposob2(x,y,z);
		}
		else printf("Nieznana opcja\n");
	}

	return 0;
}
```

Zadanie 15. Przerób powyższy przykład – połącz funkcję **menu** oraz **pobierzOpcje** w jedną funkcję, która wyświetli menu i jednocześnie pobierze od użytkownika opcję, którą zwróci do zmiennej **odp** funkcji **main**.

Zadanie 16. Wykonaj eksperyment – w funkcji **liczbaRozwiazanSposob1** zakomentuj ostatnią linijkę. Skompiluj program i zauważ, że wyświetli się warning: **not all control paths return a value** – oznacza to, że nie wszystkie możliwości zwracają jakąś wartość – może się zdarzyć, że argumenty pobrane przez funkcję nie wpadną w żadnego **if**a i funkcja nic nie zwróci. **Jeżeli funkcja nie jest typu void, to musi zawsze zwracać jakąś wartość, niezależnie od pobranych argumentów !!!**

Zadanie 17. Napisz program, w którym w funkcji **main** podasz z klawiatury macierz 4x4 i obliczysz jej wyznacznik za pomocą funkcji **double wyznacznik(double macierz[4][4])**.

Zadanie 18. Napisz program, w którym w funkcji **main** podasz z klawiatury 10 liczb, a następnie posortujesz i wyświetlisz je za pomocą funkcji **void posortujOrazWyswietl(int liczby[10])**.

Zadanie 19. Przerób poprzednie zadanie – utwórz funkcję
void posortujOrazWyswietl2(int liczby[10], int ile)
która posortuje i wyświetli tylko **ile** pierwszych liczb.

Zadanie 20*. Napisz program, w którym w funkcji **main** podasz z klawiatury 5 cyfr wczytywanych do tablicy char[5]. Za pomocą funkcji **int zamienTekstNaLiczbe(char tab[5])** zamień napis będący liczbą na prawdziwą liczbę. Program powinien być idiotoodporny – jeżeli użytkownik poda nie-cyfrę, należy zakończyć działanie programu lub w pętli **while** domagać się cyfry aż do skutku.

5.3 Rekurencje, funkcje w funkcjach, prototypy funkcji

Poniższy kod oblicza silnię za pomocą rekurencji:

```
#include "stdafx.h"

int silnia(int n){
        if(n<=1) return 1;
        else return n*silnia(n-1);
}

int main(int argc, char* argv[]){

        for(int i=0;i<6;i++){
                int wynik = silnia(i);
                printf("%d! = %d\n",i,wynik);
        }

        return 0;
}
```

Zadanie 21. Napisz funkcję znajdującą NWD dwóch liczb **a** i **b** korzystając z algorytmu Euklidesa: **int nwd(int a,int b)**
- jeżeli **b** wynosi zero, to zwróć **a**
- w przeciwnym razie zwróć największy wspólny dzielnik liczb **b** oraz **a mod b**

Zadanie 22*. Napisz funkcję znajdującą rekurencyjnie n-tą liczbę Fibonacciego: **int fib(int n)**
- jeżeli **n** wynosi jeden lub dwa, to zwróć **1**
- w przeciwnym razie zwróć **fib(n-1)+fib(n-2)**

Czasem zdarza się, że dwie funkcje korzystają z siebie na wzajem, np:

```
#include "stdafx.h"

int dodajDwieLiczby1(int a,int b){
        if(b==1) return ++a;
        else return dodajDwieLiczby2(a+1,b-1);
}

int dodajDwieLiczby2(int a,int b){
        if(b==1) return ++a;
        else return dodajDwieLiczby1(a+1,b-1);
}


int main(int argc, char* argv[]){

        int suma = dodajDwieLiczby(13,5);
        printf("%d\n",suma);

        return 0;

}
```

Te dwie funkcje robią dokładnie to samo – dodają rekurencyjnie dwie liczby, przy czym przerzucają na zmianę do siebie wynik. Nie ma to większego sensu, ale doskonale ilustruje pewien problem ...

Spróbuj skompilować powyższy program – nie uda się to, ponieważ program kompilowany jest „od góry". Kompilator napotka na instrukcję `else return dodajDwieLiczby2(a+1,b-1);` i zgłosi, że nie wie, co to jest funkcja **dodajDwieLiczby2**. Faktycznie, nie wie o jej istnieniu, ponieważ znajduje się ona niżej w kodzie programu i jeszcze jej nie przeczytał. Jeżeli przerzucisz funkcję **dodajDwieLiczby2** nad funkcję **dodajDwieLiczby1**, to nadal będzie zgłaszany błąd, ponieważ tym razem kompilator nie będzie potrafił zinterpretować linijki `else return dodajDwieLiczby1(a+1,b-1);` . W takich sytuacjach stosuje się *prototypy funkcji*. Przywróć program do pierwotnego stanu. Zaraz pod **includami** wpisz linijkę:

int dodajDwieLiczby2(int a,int b);

Ponownie spróbuj skompilować program – tym razem się to uda, ponieważ właśnie powiedziałeś kompilatorowi, że gdzieś tam w kodzie zawiera funkcję **dodajDwieLiczby2**, która pobiera dwie liczby całkowite i zwraca liczbę całkowitą. Kompilator nie musi na razie znać treści tej funkcji, wystarczy, że będzie wiedział o jej istnieniu podczas kompilowania niebieskiego fragmentu z funkcją **dodajDwieLiczby1**.

Zadanie 23. Sprawdź, że jeżeli zamiast dopiero co dodanej linijki wpiszesz bardziej ogólnie (bez **a** i **b**): **int dodajDwieLiczby2(int,int);** to program zadziała tak samo dobrze.

5.4 Funkcje ze zmienną liczbą argumentów i funkcje przeciążone

Czasami zdarzy się, że jedna funkcja przyda się nam do obsługi różnej liczby argumentów. Przykład:

```
#include "stdafx.h"

int posumuj(int a1=0,int a2=0,int a3=0){
        return a1+a2+a3;
}

int main(int argc, char* argv[]){

        int suma = posumuj();
        printf("%d \n",suma);

        int sumka = posumuj(2);
        printf("%d \n",sumka);

        int sumeczka = posumuj(2,3);
        printf("%d \n",sumeczka);

        int sumunia = posumuj(2,3,7);
        printf("%d \n",sumunia);

        return 0;
}
```

Ten zapis oznacza: *„pobieraj trzy argumenty, ale gdybyś któregoś nie dostał, to przyjmij że jest on równy zero”.*

Zadanie 24. Przerób powyższą funkcję tak, by obliczała średnią otrzymanych argumentów. Jeżeli któryś jest zerem, to nie należy go uwzględniać – należy więc na początku zliczyć, ile argumentów jest różnych od zera.

Jeżeli jedna funkcja przyjmuje argumenty różnych typów, to wówczas mówimy o *przeciążeniu* funkcji:

```
#include "stdafx.h"

int nastepny(int i){
        return i+1;
}

double nastepny(double a){
        return a+0.00001;
}

char nastepny(char literka){
        if(literka>64 && literka <=90) return literka+32;
        else return literka;
}

int main(int argc, char* argv[]){

        int a = 7;
        double b = 7;
        char c = 'D';

        printf("%d %lf %c\n",nastepny(a),nastepny(b),nastepny(c));

        return 0;
}
```

Te trzy funkcje mają tę samą nazwę, lecz przyjmują argumenty różnych typów.

Zadanie 25. Napisz pięć funkcji o nazwie **mojTyp** zwracające **void**, przyjmujące parametry typu: **void**, **int**, **double**, **char** oraz **bool** (jeden parametr dla jednej funkcji) które wypisują np:
„Jestem typu double, mam wartość 8.5”
albo:
„Jestem typu void i nie mam wartości”.

Wywołaj swoje funkcje w funkcji **main**, np:

```
mojTyp();
mojTyp(2);
mojTyp(2.4);
mojTyp('c');
mojTyp(false);
```

5.5 Zadania

Zadanie 26. Napisz funkcję **int potegujSzybko(int a, int b, int c)**, która obliczy $\mathbf{a^b}$ **mod c** korzystając z algorytmu szybkiego potęgowania.

Zadanie 27. Napisz funkcję **void zakoduj(char wiadomosc[20],int przesuniecie)**, która zakoduje podaną wiadomość kodem Cezara. Zadbaj o to, by literki pod koniec alfabetu przeszły na literki z początku alfabetu (zastosuj operację modulo).

Zadanie 28. Napisz funkcję **int binarneNaDziesietne(char liczba[10])**, która zamieni tekst reprezentujący liczbę binarną na liczbę dziesiętną.

Zadanie 29. Napisz funkcję **int jakiesNaDziesietne(char liczba[10],int podstawa)**, która zamieni tekst reprezentujący liczbę w systemie o podstawie *podstawa* na liczbę dziesiętną.

Zadanie 30. Zadeklaruj **globalną** tablicę całkowitoliczbową o 10 elementach. Wypełnij ją dowolnymi wartościami w funkcji **main**. Napisz funkcję **void posortuj(void)**, która posortuje tę tablicę. Następnie wyświetl zawartość posortowanej tablicy w funkcji **main**.

Zadanie 31. Napisz funkcję typu **void** o zmiennej liczbie argumentów (max. 3 argumenty domyślnie równe zero). Jeżeli podano jeden argument, to funkcja ma wypisać np. „Długość wynosi 5”. Dla dwóch argumentów: „Pole wynosi ...” zaś dla trzech „Objętość wynosi ...”. Dla zera argumentów: „punkt – brak wielkości”.

5.6 Najczęstsze błędy

Zapominanie o zaincludowaniu biblioteki **math.h** podczas korzystania z jej funkcji.
Przekonanie, że funkcja **log()** to logarytm o podstawie 2. Tak na prawdę jest to **ln()** !!!
Zapominanie, że funkcje trygonometryczne przyjmują argumenty w radianach, a nie w stopniach.
Zapominanie o zwracaniu wartości w funkcjach nie-**void**owych.

Złe przekazywanie parametrów, np:

```
int funkcja(double a,b,c){ ....
```

albo **wartosc = funkcja(1 2 3);**

Próba wyświetlania wartości zwracanej przez funkcję typu **void** – a przecież takie funkcje nie zwracają wartości.
Zwracanie na siłę jakiejś wartości w funkcji typu **void**.

Najczęstszy błąd: wyświetlanie wartości zamiast jej zwracania.

Nieprawidłowe przekazywanie macierzy do funkcji.
Niedopuszczanie myśli o istnieniu pustego **return ;**.
Nieumiejętne korzystanie z rekurencji i zapętlanie się przy tym.
Brak prototypów funkcji, gdy są potrzebne.
Nierozróżnianie funkcji o zmiennej liczbie argumentów od funkcji przeciążonych.

5.7 Quiz

1. Co tu jest nie tak? Popraw błędy na czerwono.

```
int suma(int x,int y);
{
    return x+y;
}
```

```
int max(int x,int y)
{
    if(x>y) return x;
}
```

```
void funkcja(int x,int y)
{
    return x+y;
}
```

2. Co wyświetli się na ekranie i dlaczego?

```
#include "stdafx.h"
#include "math.h"

int f2(double a){
        return (int)floor(a);
}

double f1(double a=0,double b=0,double c=0){
        return f2(a+b+a*b+c*c);
}

int main(int argc, char* argv[])
{
    double x=0.7, y=0.9;
    printf("%d\n",f2(f1(0.7,0.9))-f2(f1(1.5)));
    return 0;
}
```

_____
_____
_____
_____

3. Podaj własny i oryginalny (a więc inny niż menu i inne komunikaty) przykład zastosowania funkcji, która nic nie pobiera i nic nie zwraca.

_____
_____
_____