LABORATORIUM SOP C

KATALOG DOMOWY SOP_C

1. Zalogować się na szuflandię do swojego katalogu domowego.
2. Wyświetlić bieżącą ścieżkę.
3. W katalogu domowym utworzyć (jeśli nie ma) katalog SOP_C.
4. Stworzyć plik (jeśli nie ma) SOP_C\C_info.txt
5. Wpisać do pliku C_info.txt swoje imię i nazwisko, semestr i nazwę przedmiotu
6. Wyświetlić zawartość katalogu domowego w postaci listy.

HELLO SOP C WORLD p11.c

W katalogu SOP_C tworzymy plik p11.c, w którym umieścimy pierwszy program w C:

```
touch p11.c
```

Następnie za pomocą dowolnego edytora ASCII (vi, gedit, itp.) wpisujemy do niego zawartość:

```
#include <stdio.h>

main()
{
    printf("Hello SOP C world\n");
}
```

Powyższy program, od którego zaczyna chyba każdy, kto ma zamiar nauczyć się programowania w jakimkolwiek języku, ma bardzo proste działanie. Polega na wyświetleniu na ekranie tekstu Hello world. Omówmy poszczególne linie kodu, z którego składa się ten program. W pierwszej lini mamy do czynienia z dołączeniem pliku nagłówkowego, który jest "przewodnikiem" dla kompilatora, informującym go w jaki sposób ma postępować z funkcjami bibliotecznymi zawartymi w programie. W plikach nagłówkowych, dołączanych za pomocą dyrektywy #include, zawarte są informacje o standardowych funkcjach bibliotecznych. Chcąc użyć jakiejś funkcji w programie, musimy dołączyć plik nagłówkowy, w którym znajduje się "przepis" tej funkcji, aby poinformować w ten sposób kompilator, co powinien zrobić gdy napotka ów funkcję w programie. Jak widać, nasz program zawiera funkcję **printf()**. Ma tutaj miejsce to o czym mówiliśmy przed sekundą. Kompilator napotyka tą funkcję i sprawdza, czy został dołączony plik nagłówkowy, który zawiera informację, jak należy z nią postępować. Otóż w tym wypadku plik stdio.h zawiera informację związane z naszą funkcją **printf()**.Działanie polega na wyświetleniu na ekranie tekstu zawartego w cudzysłowach. W naszym wypadku, na ekranie pojawi się napis Hello world. Znak **\n** na końcu frazy jest znakiem formatującym, który powoduje zejście do nowej linii. Inne znaki formatujące to: **\b**- tabulacja, **\"**- cudzysłów, ****- znak \.
Zastanawiasz się pewnie, jaką role pełnią linijki kodu, które nie zostały przeze mnie jeszcze omówione. Linijka "main()" rozpoczyna funkcję główną programu, i zawsze musi być ona zawarta w programie, ponieważ rozpoczyna jego wykonanie. Funkcja **main()**, jak każda inna funkcja zawiera swoje ciało w nawiasach klamrowych, w których znajdują się instrukcje, składające się na ów treść.

KOMPILACJA i URURCHOMIENIE p11

Po zapisaniu zmian w pliku p11.c nadchodzi czas na kompilację programu. W tym celu w powłoce linuxa wykonujemy polecenie:

```
gcc p11.c -o p11
```

Gdybyśmy potrzebowali uzyskać więcej informacji o błędach i ostrzeżeniach, wówczas pomocne byłoby wywołanie z parametrem `-Wall`:

```
gcc -Wall p11.c -o p11
```

Aby uruchomić nasz program wykonujemy:

```
./p11
```

PRINTF do wyświetlania zmiennej p12.c

Teraz stwórzmy plik p12.c i wpiszmy do niego:

```
#include <stdio.h>

main()
{
    int num=1000;
    printf("%d jest wartoscia zmiennej num\n",num);
}
```

Następnie go skompilujmy i wykonajmy poleceniem:

```
gcc p12.c -o p12
./p12
```

W naszym drugim programie, widzimy dodatkową możliwość instrukcji **printf()**. Mianowicie posługując się tą funkcją możemy wyświetlić zawartość zmiennej dowolnego typu. Ale czym jest zmienna i co to jest typ zmiennej? Definicja ściśle książkowa mówi, że jest to obszar pamięci komputera przechowujący pewne dane. O tym, jakiego rodzaju są te dane informuje nas typ zmiennej. Najprościej wytłumaczyć to w taki sposób, że zmienna jest swojego rodzaju pudełkiem, w którym coś przechowujemy. Odpowiednikiem nazwy pudełka jest nazwa naszej zmiennej występującej w programie, natomiast rodzaj przedmiotów przechowywanych w pudełku odpowiada typowi zmiennej. Podstawowymi typami, jakie może przechowywać zmienna są liczby całkowite (int), zmiennoprzecinkowe (float), znaki (char) lub ciągi znaków. O innych, bardziej zaawansowanych rodzajach zmiennych porozmawiamy w jednym z dalszych rozdziałów. Omówmy jeszcze pojęcia takie jak deklaracja oraz definicja zmiennej. Deklaracją zmiennej nazywamy powołanie do życia określonej zmiennej, przydzielenie pamięci pozbawione przypisania jej konkretnej wartości. O definicji zmiennej mówimy wówczas, gdy ma miejsce przypisanie wartości dla zadeklarowanej wcześniej zmiennej. Istnieje możliwość "załatwienia" deklaracji i definicji zmiennej w jednej linii kodu, tak jak zrobiliśmy to w naszym programie. Analizując linię "int num=1000", słowo kluczowe **int** określa typ zmiennej, **num** jest ustaloną przez nas nazwą zmiennej, która stoi po prawej stronie znaku przypisania (**=**), natomiast

1000 jest wartością, którą przypisujemy zmiennej **num**. Zatem w tej linijce uporaliśmy się z zadeklarowaniem zmiennej typu **int**, nadaniem jej nazwy **num** oraz przypisaniem jej wartości. Jeśli chcielibyśmy rozdzielić na dwie osobne linie w kodzie deklarację i definicję, pierwsza z nich miałaby postać "int num;", natomiast druga "num=1000". W przypadku odzielnej deklaracji i definicji nie można przestawić kolejności, ponieważ zmienna musi być wcześniej zadeklarowana, jeśli mamy zamiar nadać jej wartość.
Teraz posiadasz już wiedzę czym jest zmienna w programie. W celu wyświetlenia jej wartości, po raz kolejny posługujemy się funkcją **printf()**, jednak w tym przykładzie ma ona nieco inną budowę. W cudzysłowach, obok tesktu który ma być "otoczką" wyświetlonej wartości zmiennej posługujemy się znakiem przekształcenia- w naszym wypadku %d. Nie zawsze jednak będzie to taki znak. Kiedy chcemy wyświetlić to, co naprawdę znajduje się w konretnej zmiennej konkretnego typu, musimy posłużyć się znakiem przekształcenia charakterystycznym dla danego typu zmiennej- standardowo: %d- int, %f- float, %c- char, %s- string (ciąg znaków). Znak przekształcenia nie musi być jednak być "zgodny" z typem zmiennej. Na przykład co stanie się kiedy użyjemy do wyświetlenia zmiennej typu char znakiem przekształcenia %d? Jak można wyświetlić kod liczbowy znaku? W tym wypadku wyświetlony zostałby kod ASCII danego znaku. Po zamknięciu cudzysłowa oraz po przecinku który za nim występuje, podajemy zmienne które mają zostać wyświetlone w miejscach znaków przekształcenia.

DEFINE

```
#include <stdio.h>
#define NUM 1000

main()
{
    printf("%d jest wartoscia zmiennej NUM\n",NUM);
}
```

Podany przykład różni się od poprzedniego tylko i wyłącznie sposobem zadeklarowania i definicji zmiennej. Zdefiniowana została tutaj zmienna globalna. W tym celu posłużyliśmy się dyrektywą **#define**, w linijce z którą podaliśmy nazwę zmiennej- **NUM**, oraz jej wartość- **100**. Podczas definicji zmiennych musimy pamiętać o dość istotnej sprawie.

SCANF p13.c

Teraz stwórzmy plik p13.c i wpiszmy do niego:

```
#include <stdio.h>

main()
{
    float l1,l2;
    printf("podaj 1 liczbe\n");
    scanf("%f",&l1);
    printf("podaj 2 liczbe\n");
    scanf("%f",&l2);
    printf("suma: %f\n",l1+l2);
}
```

Następnie go skompilujmy i wykonajmy poleceniem:

```
gcc p13.c –o p13
./p13
```

Działanie programu polega na wczytaniu dwóch liczb zmiennoprzecinkowych i wyświetleniu ich sumy. W celu wczytania liczb posługujemy się funkcją **scanf()** z omawianego wcześniej pliku nagłówkowego **stdio.h**. Poruszając temat składni, w cudzysłowach podajemy znak przekształcenia informujący kompilator jakiego typu zmienną wczytujemy, natomiast po przecinku przekazujemy adres zmiennej której chcemy przypisać wartość, poprzedzając jej nazwę znakiem **&** co oznacza, że wartość która wpiszemy zostanie zapisana pod adres tej zmiennej w pamięci. Instrukcja **printf()** jest na tyle inteligentna, że potrafi wykonać zadaną operacje na podanych po przecinku zmiennych. W naszym programie funkcja **printf()** w pierwszej kolejności dodaje do siebie wartości zmiennych **l1** i **l2**, a następnie wstawia obliczoną wartość w miejsce znaku przekształcenia %f.

SCANF p14.c

Teraz stwórzmy plik p14.c i wpiszmy do niego:

```
#include <stdio.h>

main()
{
    float w,l,h;

    printf("jednostki musza byc zgodne!\n");
    printf("podaj szerokosc prostopadloscianu\n");
    scanf("%f",&w);
    printf("podaj dlugosc prostopadloscianu\n");
    scanf("%f",&l);
    printf("podaj wysokosc prostopadloscianu\n");
    scanf("%f",&h);
    printf("objetosc wynosi %f j^3\n",w*l*h);
}
```

Następnie go skompilujmy i wykonajmy poleceniem:

```
gcc p14.c –o p14
./p14
```

Powyższy program pozwala na obliczenie objętości prostopadłościanu dla podanych rozmiarów. Różnica, biorąc pod uwagę poprzedni program polega tylko na tym, że dodawanie zostało zamienione na mnożenie, oraz zamiast na dwóch, działanie jest wykonywane na trzech zmiennych.

FUNKCJE p15.c

Teraz stwórzmy plik p15.c i wpiszmy do niego:

```
#include <stdio.h>

float przelicz(float w_dolarach);

main()
{
    float dolary,funty;
    printf("podaj kwote w dolarach\n");
    scanf("%f",&dolary);
    funty=przelicz(dolary);
    printf("%f $ to %.2f funta/funtow\n",dolary,funty);
}

float przelicz(float w_dolarach)
{
    return w_dolarach/1.98;
}
```

Następnie go skompilujmy i wykonajmy poleceniem:

```
gcc p15.c –o p15
./p15
```

Powyższy program został wzbogacony o funkcję przeliczającą podaną kwotę w dolarach na kwotę w funtach. Ograniczmy się do informacji, że funkcja przeliczająca pobiera kwotę w dolarach, wykonuje na niej operację i zwraca jakąś wartość- w tym wypadku kwotę w funtach. Co należy zapamiętać z tego przykładu to to, że program główny, funkcja główna **main()** nie musi wykonywać wszystkich operacji w programie. Może w swoim wnętrzu wywoływać inne funkcje, które zajmą się zleconymi zadaniami.

FUNKCJE p16.c

Teraz stwórzmy plik p16.c i wpiszmy do niego:

```
#include <stdio.h>

void wyp_licz(int num);

main()
{
    int liczba;
    printf("podaj liczbe calkowita\n");
    scanf("%d",&liczba);
    wyp_licz(liczba);
}

void wyp_licz(int num)
{
    printf("podana liczba wynosi %d\n",num);
}
```

Następnie go skompilujmy i wykonajmy poleceniem:

```
gcc p16.c –o p16
./p16
```

Zwróćmy uwagę na jedną rzecz. Otóż funkcje mogą być deklarowane i definiowane na kilka sposobów, dlatego z punktu widzenia możemy wyróżnić następujące możliwości:

- "pobieram wartość/wartości i zwracam wartość/wartości do programu głównego",
- "pobieram wartość/wartości, ale nic nie zwracam",
- "nie pobieram żadnych wartości, a mimo tego zwracam coś do programu głównego" oraz
- "nic nie pobieram, ani nic nie zwracam".

IF..ELSE..

```
#include <stdio.h>

main()
{
    int liczba;
    printf("podaj liczbe calkowita\n");
    scanf("%d",&liczba);
    if(liczba%2==0)
        printf("liczba parzysta\n");
    else
        printf("liczba nieparzysta\n");
}
```

Działanie podanego programu, polega na sprawdzaniu czy podana liczba jest parzysta czy nieparzysta. W celu osiągnięcia takiego efektu posługujemy się instrukcją sterującą **if**. Za słowem kluczonym **if**, w nawiasie podajemy warunek, lub warunki, które muszą zostać spełnione żeby istrukcja mogła się wykonać. Porównanie wartości podczas sprawdzania warunku uzyskujemy za pomocą dwóch operatorów przypisana- **==**. Jeśli warunek nie jest spełniony, zwrócona wartość wynosi zero, i instrukcje nie wykonują się. W przypadku gdy nasza instrukcja sterująca zawiera w sobie jedną instrukcję możemy pominąć nawiasy klamrowe, które muszą się pojawić w sytuacji, kiedy instrukcja sterująca zawiera więcej niż jedną instrukcję. Użycie nawiasów klamrowych zobaczymy w kolejnych przykładach.
Instrukcje składowe instrukcji sterującej **else** są wykonywane, podczas gdy warunek instrukcji sterującej **if** nie został spełniony. Sprawdzanie parzystości liczby uzyskujemy za pomocą operatora reszty dzielenia **%**. Sprawdzamy, czy reszta z dzielenia podanej liczby wynosi 0.

IF..ELSE.. IF

```
#include <stdio.h>

main()
{
    float a,b;
    int opcja;
```

```
    printf("podaj liczbe a\n");
    scanf("%f",&a);
    printf("podaj liczbe b\n");
    scanf("%f",&b);
    printf("wybierz operacje:\n1- suma, 2- iloczyn\n");
    scanf("%d",&opcja);
    if(opcja==1)
        printf("suma wynosi %f\n",a+b);
    else if(opcja==2)
        printf("iloczyn wynosi %f\n",a*b);
```

Przedstawiony przykład, na życzenie użytkownika oblicza sumę lub różnicę podanych liczb. W pierwszej kolejności zapisane zostają liczby. W następnym etapie program pyta nas, czy chcemy policzyć sumę czy różnicę wczytanych liczb. Następuje porównanie, sprawdzenie warunków. Jeśli warunki którejś z instrukcji sterujących są spełnione, wykonane zostają instrukcje w niej zawarte, o czym mówiliśmy wcześniej. Różnica między użytą w tym przykładzie instrukcją **else if**, a użytą w poprzednim programie instrukcją **else** polega na tym, że warunki tej pierwszej możemy określić sami, natomiast warunek instrukcji **else** jest prawdziwy, wtedy, kiedy warunki innych instrukcji sterujących (**if, else if**) nie są spełnione.

FOR

```
#include <stdio.h>

main()
{
    int i;
    for(i=17;i<=100;i++)
    {
        if(i%17==0)
            printf("%d ",i);
    }
    printf("\n");
}
```

Podany przykład wyświetla liczby podzielne przez 17 z przedziału [17,100]. Działanie pętli **for** już znamy. Omijając linię kodu "if(i%17==0)" działanie programu, jak już się domyślasz polegałoby na wyświetleniu liczb od 17 do 100. Jednak nie byłoby to rozwiązanie o które nam chodzi. Dodając instrukcję sterującą, posługujemy się warunkiem, który sprawdza czy reszta z dzielenia kolejnych liczb z przedziału [17,100] wynosi zero. Jeśli tak, logicznym jest że ta liczba jest podzielna przez 17 i zostaje ona wyświetlona na ekranie, co chcieliśmy osiągnąć.

FOR .. IF

```
#include <stdio.h>

main()
{
```

```
    int i;
    for(i=17;i<=100;i++)
    {
        if(i%17==0)
            printf("%d ",i);
    }
    printf("\n");
}
```

Podany przykład wyświetla liczby podzielne przez 17 z przedziału [17,100]. Działanie pętli **for** już znamy. Omijając linię kodu "if(i%17==0)" działanie programu, jak już się domyślasz polegałoby na wyświetleniu liczb od 17 do 100. Jednak nie byłoby to rozwiązanie o które nam chodzi. Dodając instrukcję sterującą, posługujemy się warunkiem, który sprawdza czy reszta z dzielenia kolejnych liczb z przedziału [17,100] wynosi zero. Jeśli tak, logicznym jest że ta liczba jest podzielna przez 17 i zostaje ona wyświetlona na ekranie, co chcieliśmy osiągnąć.

WHILE

```
#include <stdio.h>

main()
{
    char a;
    printf("podaj litere\n");
    scanf("%c",&a);
    while(a!=0)
    {
        printf(".");
        a--;
    }
    printf("\n");
}
```

Kolejną przydatną pętlą jest pętla **while**. Jej działanie jest podobne do działania pętli for. Instrukcje zawarte w pętli while są wykonywane wówczas, gdy spełniony jest warunek. Tłumacząc to w inny sposób, dopóki warunek jest spełniony wykonuj instrukcje. Przytoczony przykład pobiera od użytkownika znak, a następnie wyświetla ilość kropek równą liczbie w kodzie ASCII podanego znaku. Zauważ jedną rzecz. Pętla while porównuje znak z liczbą? Właśnie tak. Podczas takiego porównania podstawiany jest automatycznie kod ASCII znaku i mamy wtedy sytuację porównania dwóch liczb. To umożliwia nam wyświetlenie odpowiedniej liczby kropek. Przykładowo, jeśli kod ASCII podanego znaku wynosi 20, pętla while sprawdza czy warunek jest spełniony, i oczywiście jest. Następnie wyświetlana zostaje kropka, i zachodzi dekrementacja, czyli zmiejszenie kodu ASCII znaku o 1.

CONTINUE

```
#include <stdio.h>

main()
```

```
{
    int i;
    printf("liczby nieparzyste z przedzialu 1-100\n");
    for(i=1;i<=100;i++)
    {
        if(i%2==0)
            continue;
        printf("%d ",i);
    }
    printf("\n");
}
```

Powyższy przykład jest bardzo zbliżony do wcześniejszych programów. Nie ma w nim żadnych trudności. Została wprowadzona nowa instrukcja sterująca **continue**. Jej działanie polega na przerwaniu aktualnego kroku przetwarzania pętli i automatycznego skoku do następnego kroku wykonywania. Instrukcja **continue** zamieszczona jest w instrukcji **if** co oznacza, że będzie wykonana tylko gdy spełniony zostanie określony warunek. Wracając do przykładu, którego zadaniem jest wyświetlenie liczb nieparzystych z przedziału 1-100, instrukcja **continue** zostaje uruchomiona w momencie, kiedy reszta z dzielenia aktualnej liczby w pętli przez dwa wynosi zero, co nie pozwala na wyświetlenie liczb parzystych, bo przecież o to dokładnie nam chodzi.

SWITCH .. CASE

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

main()
{
    float kat;
    int opcja;
    printf("ile stopni ma kat?\n");
    scanf("%f",&kat);
    printf("1- sinus, 2- cosinus\n");
    scanf("%d",&opcja);
    switch(opcja)
    {
        case 1:
            printf("sin %f stopni=%.3f\n",kat,sin(kat/180*M_PI));
            break;
        case 2:
            printf("cos %f stopni=%.3f\n",kat,cos(kat/180*M_PI));
            break;
        default:
            printf("blad\n");
    }
}
```

Pierwszą nową rzeczą, jaka pojawia się w tym programie, jest instrukcja sterująca **switch** Działanie tej instrukcji polega na zdefiniowanie działań dla różnych wyników jednego wyrażenia. W nawiasie obok słowa kluczowego **switch** jest określone wyrażenie, dla którego są określone odpowiednie działania. Polega to dokładnie na tym, że w zależności od wartości wyrażenia w nawiasie uruchamiane są odpowiednie instrukcje. W naszym przykładzie, obliczającym sinus lub cosinus kąta, sterowanie

odbywa się właśnie poprzez instrukcje **switch**. W przypadku, gdy wartość zmiennej opcja wynosi 1 (**case 1**), program pozwala obliczyć nam wartość funkcji sinus dla podanego kąta, natomiast jeśli zmienna opcja ma wartość 2 (**case 2**), cosinus. Określone są też instrukcje, które zostaną wykonane kiedy wartość zmiennej opcja nie będzie równa żadnej z wartości stojących przy **case** (**default**). Drugą ważną nowością, jest dołączenie do programu pliku nagłówkowego **math.h**. Zauważ, że korzystamy w programie z funkcji **sin(), cos()**, oraz ze stałej odpowiadającej liczbie pi **M_PI**, które są zawarte właśnie w tym pliku. Ważna informacją, które z tego wynika jest to, że chcąc posłużyć się jakąś funkcją matematyczną zawartą w pliku nagłówkowym **math.h** musimy użyć dyrektywy **#include** i dołączyć ten plik nagłówkowy.

DO .. WHILE

```
#include <stdio.h>

main()
{
    int i;
    do
    {
        printf("jaka jest tajna liczba?\n");
        scanf("%d",&i);
    }while(i!=6);
    printf("zezwalam na dostep\n");
}
```

W podanym programie, demonstrujemy działanie nowej pętli, mianowicie pętli **do while**. Od swojej siostry **while** różni się ona tylko tym, że wykonanie instrukcji wykonywane jest przed sprawdzeniem warunku. Taka właściwość dość często się przydaje. W powyższym programie wygląda to w taki sposób, że zostajemy zapytani o tajną liczbę, a następnie dopiero program sprawdza czy podana liczba jest równa tej właściwej tajnej liczbie.

LOSOWANIE

```
#include <stdio.h>

main()
{
    int a,b,wynik,i,punkty=0;
    srand(time(NULL));
    printf("przed Toba 20 losowych pytan\n");
    for(i=0;i<20;i++)
    {
        a=rand()%10+1;
        b=rand()%10+1;
        printf("%d*%d= ?\n",a,b);
        scanf("%d",&wynik);
        if(wynik==a*b)
        {
            printf("brawo!\n");
```

```
            punkty++;
        }
        else
            printf("blad, poprawny wynik to %d\n",a*b);
    }
    printf("zdobyles %d punktow na 20 mozliwych\n",punkty);
    if(punkty>=0&&punkty<=4)
        printf("dostajesz 1\n");
    else if(punkty>4&&punkty<=8)
        printf("dostajesz 2\n");
    else if(punkty>8&&punkty<=12)
        printf("dostajesz 3\n");
    else if(punkty>12&&punkty<=16)
        printf("dostajesz 4\n");
    else if(punkty>16&&punkty<=20)
        printf("dostajesz 5\n");

}
```

Działanie polega na wylosowaniu 20 mnożeń i podawaniu kolejnych wyników. Coś ala test z tabliczki mnożenia. W razie podania poprawnego wyniku, otrzymujemy pozytywny komunikat i zwiększana jest ilość zdobytych punktów. Po sprawdzeniu 20 działań program informuje nas, jaką ilość punktów zdobyliśmy, na 20 możliwość i na jaką zasługujemy ocenę.

GO TO

```
#include <stdio.h>

main()
{
    int a,b,ilosc=0,punkty=0,wynik,opcja;
    srand(time(NULL));
    poczatek:
        a=rand()%100+1;
        b=rand()%100+1;
        printf("%d+%d= ?\n",a,b);
        scanf("%d",&wynik);
        if(wynik==a+b)
        {
            printf("brawo!\n");
            punkty++;
        }
        else
            printf("blad, poprawny wynik to %d\n",a+b);
        ilosc++;
        printf("kontynuowac?1- tak, 2- koniec\n");
        scanf("%d",&opcja);
        if(opcja==1)
            goto poczatek;
        if(opcja==2)
        {
            printf("Twoj wynik to %d/%d\n",punkty,ilosc);
            exit(1);
        }
```

```
}
```

Tutaj, po każdym działaniu, program pyta nas, czy chcemy kontynuować zabawę. Jeśli decydujemy się na dalsze dodawania, korzystamy z instrukcji **goto** w celu powrócenia do części programu, która znajduje się przed fragmentem zadającym pytania. Zwróć uwagę, że przy zadaniu każdego pytania zawsze inkrementowana jest zmienna **ilosc** odpowiadająca za ilość zadanych pytań. W razie poprawnej odpowiedzi zwiększana o jeden jest także ilość punktow, i wartości tych zmiennych pozwalają na obliczenie końcowego wyniku.

ZADANIA

Z11:

Napisz program obliczający sumę dowolnie wielu liczb wprowadzonych przez użytkownika i wypisujący wynik na standardowe wyjście.

Z12:

Napisz program kalkulator umożliwiający operacje dodawania, odejmowania, mnożenia i dzielenia.