Podstawy Programowania 1

Instrukcje sterujące

Arkadiusz Chrobot

Zakład Informatyki

27 października 2019

Notatki

Plan

Notatki

Instrukcje sterujące

Instrukcje sterujące lub *instrukcje zmieniające przepływ sterowania w programie* stanowią niezbędny element każdego języka programowania. Pozwalają one na wykonywanie lub wielokrotne wykonanie określonych instrukcji przetwarzania danych lub grup takich instrukcji, w zależności od wartości określonych warunków. Umożliwiają one zatem realizację złożonych algorytmów w programach komputerowych.

Notatki

Blok instrukcji

Blok instrukcji pozwala zgrupować instrukcje, które mają być przez program wykonane łącznie, jakby były pojedynczą instrukcją. Blok rozpoczyna znak `{`, a kończy znak `}`. Zastosowanie bloku widzieliśmy na przykładzie definicji funkcji `main()`, jednakże jest on używany także w innych elementach programów, również razem z instrukcjami sterującymi.

Notatki

Instrukcja warunkowa

Opis

Instrukcja warunkowa jest instrukcją decyzyjną, która steruje wykonaniem innych instrukcji lub grup instrukcji w zależności od wartości zawartego w niej warunku. Schemat instrukcji warunkowej jest następujący:

```
if(warunek)
    instrukcja;
else
    instrukcja_alternatywna;
```

Jeśli spełniony jest `warunek` to wykonywana jest `instrukcja` w przeciwnym przypadku `instrukcja_alternatywna`. Zarówno `instrukcja`, jaki i `instrukcja_alternatywna` mogą być pojedynczymi instrukcjami lub blokami instrukcji. Słowo kluczowe `else` może zostać pominięte wraz z instrukcją alternatywną. Warunek w instrukcji warunkowej może być dowolnym wyrażeniem.

Notatki

Instrukcja warunkowa

Uwagi

Język C pozwala pominąć nie tylko słowo kluczowe `else` i instrukcję alternatywną, ale również instrukcję znajdującą się tuż za warunkiem, poprzez postawienie średnika za zamykającym nawiasem okrągłym. Taki zapis ma niewielkie zastosowanie praktyczne. Jednym z najczęściej spotykanych błędów jest pomylenie w warunku instrukcji przypisania (`=`) z operatorem `==`. Zapis warunku z operatorem przypisania jest uznawany przez kompilator[1] za prawidłowy i w pewnych sytuacjach może być celowo i poprawnie wykorzystany przez programistę.

[1] Kompilator jedynie generuje ostrzeżenie domagając się, by programista umieścił taki warunek w dodatkowej parze nawiasów okrągłych

Notatki

Instrukcja warunkowa

Przykład

```
if(a==b)
        a=5;
else
        b=5;
```

Część programistów zaleca, aby stosować blok (umieszczać instrukcje w nawiasach klamrowych) nawet wtedy, gdy po warunku i/lub słowie kluczowym `else` występują tylko pojedyncze instrukcje:

```
if(a==b) {
        a=5;
} else {
        b=5;
}
```


Notatki

Instrukcja warunkowa

Zagnieżdżanie instrukcji warunkowych

Instrukcję warunkową można umieścić w innej instrukcji warunkowej. Taką czynność nazywa się zagnieżdżaniem, a instrukcję znajdującą się wewnątrz innej - instrukcją zagnieżdżoną:

```
if(a==3) {
        if(b==4)
                c=5;
}
```

Ta technika pisania programu, może sprawić, że powstały kod będzie nieczytelny. Lepiej w takiej sytuacji zastosować warunek złożony:

```
if(a==3 && b==4)
                c=5;
```

Należy jednak pamiętać o tym, że w języku C stosowane jest skracanie obliczania warunków logicznych.

Notatki

Instrukcja wielokrotnego wyboru

Instrukcja wielokrotnego wyboru pozwala wykonać zbiór instrukcji w zależności od wartości zmiennej typu `int` lub `char` lub typów od nich pochodnych. Schemat tej instrukcji jest następujący:

```
switch(zmienna) {
   case wartość_1: instrukcja_1;
                   break;
…
   case wartość_n: instrukcja_n;
                   break;
   default: instrukcja;
}
```

Liczba przypadków (`case`) jest ograniczona jedynie zakresem wartości przyjmowanych przez zmienną.

Notatki

Instrukcja wielokrotnego wyboru
Uwagi

Przypadek w instrukcji wielokrotnego wyboru może obejmować więcej niż jedną instrukcję. W takim wypadku wszystkie one muszą znajdować się przed słowem kluczowym `break`, które kończy zapis przypadku a podczas wykonania programu działanie instrukcji `switch`. Jeśli słowo `break` zostanie pominięte, to program przystąpi do realizacji następnego w kolejności przypadku, nie sprawdzając dla jakiej wartości zmiennej powinien on być wykonany. Czasem jest to przez programistów celowo wykorzystywane, lecz dosyć często stanowi błąd. Jeśli zmienna nie ma żadnej z wartości określonych w przypadkach, to zostaje wykonany przypadek domyślny oznaczony słowem kluczowym `default`. Ten przypadek może być całkowicie pominięty w zapisie instrukcji wielokrotnego wyboru.

Notatki

Instrukcja wielokrotnego wyboru
Przykład

```
switch(a) {
        case 1: puts("Jeden");
                break;
        case 2: puts("Dwa");
                break;
        case 3: puts("Trzy");
                break;
        default: puts("Inna wartość");
}
```

Jeśli zmienna `a` będzie miała wartość `1`, to program wypisze na ekran wyraz `Jeden`, jeśli `2`, to na ekranie pojawi się napis `Dwa`. Podobnie program zachowa się, gdy zmienna `a` będzie miała wartość `3`. Jeśli zmienna będzie miała inną wartość, niż trzy wyżej wymienione, to na ekranie pojawi się napis `Inna wartość`.

Notatki

Instrukcje iteracyjne

Instrukcje iteracyjne, nazywane krótko pętlami, pozwalają na powtarzanie określonej instrukcji lub grup instrukcji określoną (czasem nieskończoną) liczbę razy. Powtórzenie pętli nazywane jest w informatyce *iteracją* lub po prostu powtórzeniem. Zazwyczaj wynik każdej iteracji jest różny od wyniku jej poprzedniczki. W niektórych sytuacjach mogą one być takie same.

Notatki

Pętla `for`

Pętla `for` służy do powtarzania instrukcji lub bloku instrukcji określoną, z góry zadaną liczbę razy. Z tą pętlą najczęściej jest związana co najmniej jedna zmienna nazywana *licznikiem pętli* lub *zmienną sterującą*. Schemat tej pętli jest następujący:

```
for(inicjacja;warunek_kontynuacji;krok)
    instrukcja;
```

,gdzie `inicjacja` oznacza nadanie licznikowi lub licznikom pętli wartości początkowej, `warunek_kontynuacji` oznacza warunek, który musi spełniać wartość licznika lub liczników pętli, aby się ona wykonywała, natomiast `krok` określa zmianę wartości licznika lub liczników pętli. Umieszczona w pętli `instrukcja` może być pojedynczą instrukcją lub blokiem instrukcji. W obu przypadkach ta część instrukcji iteracyjnej nazywa się *ciałem pętli*. Licznikiem (licznikami) pętli `for` może być zmienna dowolnego z przedstawionych na wykładzie pierwszym typów. Najczęściej te zmienne mają jednoliterowe nazwy, choć zdarzają się wyjątki do tej reguły. Pętle `for` można zagnieżdżać.

Notatki

Pętla `for`

Przykłady

```c
#include<stdio.h>

int a;

int main(void)
{
        for(a=0;a<5;a++)
                printf("%d\n",a);
        return 0;
}
```


Notatki

Pętla `for`

Przykłady

```c
#include<stdio.h>

int a;

int main(void)
{
        for(a=0;a<5;a++) {
                printf("%d\n",a);
        }
        return 0;
}
```


Notatki

Pętla `for`

Przykłady

```c
#include<stdio.h>

int a;

int main(void)
{
        for(a=1;a<=5;a++)
                printf("%d\n",a);
        return 0;
}
```


Notatki

Pętla for

Przykłady

```c
#include<stdio.h>

int a;

int main(void)
{
        for(a=0;a<7;a+=2)
                printf("%d\n",a);
        return 0;
}
```


Notatki

Pętla for

Przykłady

```c
#include<stdio.h>

int a;

int main(void)
{
        a=1;
        for(;a<=5;) {
                printf("%d\n",a);
                a++;
        }
        return 0;
}
```


Notatki

Pętla for

Przykłady

```c
#include<stdio.h>

int a;

int main(void)
{
        for(a=7;a>0;a--)
                 printf("%d\n",a);
        return 0;
}
```


Notatki

Pętla for

Przykłady

```c
#include<stdio.h>

int i,j;

int main(void)
{
        for(i=7,j=0;i>j;j++,i--)
                printf("%d %d\n",i,j);
        return 0;
}
```


Notatki

Pętla `for`

Przykłady

```c
#include<stdio.h>

double x;

int main(void)
{
        for(x=0.0;x<0.5;x+=0.01)
                printf("%.10f\n",x);
        return 0;
}
```


Notatki

Pętla `for`

Przykłady

```c
#include<stdio.h>

int a,i;

int main(void)
{
        for(i=0;i<5;i++) {
                a+=i;
                printf("%d\n",a);
        }
        return 0;
}
```


Notatki

Pętla `for`

Przykłady

```c
int a;
int main(void)
{
        for(a=0;a<5;a++)
                ;
        return 0;
}
```


Notatki

Pętla `while`

Pętla `while` powtarza wykonanie objętych nią instrukcji tak długo, jak długo spełniony jest zawarty w niej warunek. Schemat takiej pętli jest następujący:

```
while(warunek_kontynuacji)
   instrukcja;
```

Podobnie, jak w przypadku pętli `for`, ciało pętli `while` może być pojedynczą instrukcją, blokiem instrukcji lub nawet być puste. Liczba powtórzeń takiej pętli nie jest z góry zadana, dlatego musi w niej być zawarte wyrażenie lub grupa wyrażeń, które spowodują, że po skończonej liczbie iteracji ta pętla się zakończy. Pętle `while` można zagnieżdżać.

Notatki

Pętla while

Przykłady

```c
#include<stdio.h>

char a;

int main(void)
{
        while(a!='q')
                scanf(" %c",&a);
        return 0;
}
```


Notatki

Pętla while

Przykłady

```c
#include<stdio.h>

char a;

int main(void)
{
        while(a!='q') {
                scanf(" %c",&a);
        }
        return 0;
}
```


Notatki

Pętla while

Przykłady

```c
#include<stdio.h>

int x,y;

int main(void)
{
        while(y>=0) {
                scanf("%d",&y);
                x+=y;
        }
        return 0;
}
```


Notatki

Pętla do…while

Pętla `do…while` jest podobna do pętli `while` nie tylko w zapisie, również w działaniu. Podstawowa różnica między nimi polega na tym, że ciało w tej pierwszej *zawsze* wykona się co najmniej raz, ponieważ warunek jest w niej sprawdzany na końcu. Schemat tej pętli jest następujący:

```
do
   ciało
while(warunek);
```

,gdzie, podobnie jak w przypadku innych pętli `ciało` może być pojedynczą instrukcją, blokiem instrukcji lub może być puste.

Notatki

Pętla do…while

Przykłady

```c
#include<stdio.h>

char a;

int main(void)
{
        do
                scanf(" %c",&a);
        while(a!='q');
        return 0;
}
```


Notatki

Pętla do…while

Przykłady

```c
#include<stdio.h>

char a;

int main(void)
{
        do {
                scanf(" %c",&a);
        } while(a!='q');
        return 0;
}
```


Notatki

Pętla do…while

Przykłady

```c
#include<stdio.h>

int x,y=1;

int main(void)
{
        do {
                x+=1;
                y*=x;
        } while(x!=10);
        return 0;
}
```


Notatki

Słowo kluczowe break

Słowo kluczowe **break** oprócz instrukcji wielokrotnego wyboru może być także użyte wewnątrz dowolnej pętli. Zazwyczaj jest ono wtedy umieszczone także w instrukcji warunkowej. Jeśli dojdzie do jego wykonania, to przerwie ono działanie pętli, kończąc ją tym samym wcześniej niż wynikałoby to z jej warunku kontynuacji.

Notatki

Słowo kluczowe `continue`

Słowo kluczowe `continue` jest używane wyłącznie wewnątrz dowolnego rodzaju pętli. Podobnie jak `break` występuje ono wówczas w instrukcji warunkowej. Nie przerywa ono jednak całości wykonania pętli, a jedynie jej bieżące powtórzenie (iterację).

Notatki

Słowo kluczowe `continue`

Przykład

```c
#include <stdio.h>

int i;

int main(void)
{
    for(i=-5;i<=5;i++) {
        if(i==0)
            continue;
        printf("Wynik dzielenia 5 przez %d: %f\n",i,5.0/i);
    }

    return 0;
}
```


Notatki

Słowo kluczowe `goto`

Słowo kluczowe (instrukcja) `goto` (zlepek dwóch angielski słów *go* i *to*) powoduje przeniesie sterowania do wskazanego etykietą miejsca w programie. Ta etykieta może znajdować się zarówno powyżej jak i poniżej wystąpienia instrukcji `goto`, a nawet wskazywać na miejsce jej wystąpienia. Choć początkowo ta instrukcja wydaje się bardzo użyteczna, to jej stosowanie w nowoczesnych językach programowania pociąga za sobą wiele problemów. W początkach techniki komputerowej była ona nadużywana, co prowadziło do powstawania nieczytelnych programów komputerowych. Doprowadziło to do tak dramatycznej sytuacji, że jeden z pionierów informatyki, Edsger Dijkstra zdecydował się publicznie zabronić jej używania. W języku C `goto` używana jest zazwyczaj do obsługi sytuacji wyjątkowych i (przez doświadczonych programistów) do usprawniania działania programów. **Należy za wszelką cenę unikać jej używania w innych, nieuzasadnionych przypadkach.**

Notatki

Słowo kluczowe `goto`

Przykład

```c
#include <stdio.h>

int i;

int main(void)
{

label_1: i++;
         printf("%d\n",i);
         if(i==15)
            goto label_2;
         goto label_1;
label_2:
    return 0;
}
```


Notatki

Silnia

Opis

Silnia jest działaniem matematycznym na liczbach naturalnych, które zdefiniowane jest następująco:
$0! = 1$
$1! = 1$
$n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \ldots \cdot (n-1) \cdot n$
Program na następnym slajdzie realizuje to działanie za pomocą pojedynczej pętli `for`. Jej licznik (zmienna `i`) służy także do przechowywania kolejnych liczb naturalnych, które mnożne są przez siebie. Proszę zwrócić uwagę, na użycie zmiennej `factorial`, która służy nie tylko do zapamiętania wyniku końcowego, ale także wyników częściowych. Pętla `do…while` służy do ograniczenia użytkownikowi możliwości zlecenia programowi policzenia silni dla liczby większej niż 20. Wynik byłby większy niż może pomieścić typ `unsigned long long int`. Proszę zwrócić uwagę, że program działa poprawnie także wtedy, gdy każemy mu liczyć silnię z zera. Pętla `for` się wprawdzie nie wykona ani razu, ale prawidłowy wynik od początku będzie w zmiennej `factorial`.

Notatki

Silnia

Kod

```c
#include <stdio.h>

unsigned long long int factorial = 1;
unsigned char i,number;

int main(void)
{
    do {
       printf("Podaj liczbę naturalną mniejszą niż 21, ");
       printf("dla której chcesz obliczyć silnię:\n");
       scanf("%hhu",&number);
    } while(number>20);

    for(i=1;i<=number;i++)
        factorial*=i;

    printf("Silnia z %hhu to %llu\n",number,factorial);

    return 0;
}
```


Notatki

Silnia

Kod - inny zapis

```c
#include <stdio.h>

unsigned long long int factorial = 1;
unsigned char i,number;

int main(void)
{
    do {
       printf("Podaj liczbę naturalną mniejszą niż 21, ");
       printf("dla której chcesz obliczyć silnię:\n");
       scanf("%hhu",&number);
    } while(number>20);

    for(i=1;i<=number;factorial*=i,i++)
        ;

    printf("Silnia z %hhu to %llu\n",number,factorial);

    return 0;
}
```


Notatki

Największy wspólny dzielnik

Opis

Kolejny przykład, to program, który liczy Największy Wspólny Dzielnik (ang. *Greatest Common Divisor* - GCD). Jest on implementacją algorytmu z wykładu pierwszego, ale nie do końca wierną. Wprawdzie nazwy zmiennych zostały zachowane, ale ze względu na czytelność zapisu zdecydowałem, że lepiej będzie dopuścić do wykonania przypisań $m = n$ i $n = r$, nawet po tym, jak obliczanie reszty da w wyniku zero. Powoduje to dodatkową rozbieżność. Wynik końcowy nie jest zapisany w zmiennej n, lecz w m. Ponadto obliczanie Największego Wspólnego Dzielnika zostało ograniczone do liczb naturalnych, a program zawiera dodatkowe zabezpieczenie w postaci pętli `while`, które nie pozwala wprowadzić dzielnika o wartości zero.

Notatki

Największy wspólny dzielnik

Kod

```c
#include <stdio.h>

unsigned int r, n, m;

int main(void)
{
    puts("Podaj dwie liczby naturalne.");

    scanf("%u",&m);
    scanf("%u",&n);

    while(n==0) {
        puts("Dzielnik musi być różny od zera!");
        scanf("%u",&n);
    }

    do {
        r=m%n;
        m=n;
        n=r;
    } while(r!=0);

    printf("Największym wspólnym dzielnikiem podanych liczb jest %u\n",m);
    return 0;
}
```


Notatki

Równanie kwadratowe

Opis

Kolejny program liczy pierwiastki równania kwadratowego, ale używając wzorów, które są odporne na akumulację błędów zaokrąglenia, charakterystycznych dla typów zmiennoprzecinkowych i dających się zaobserwować dla przypadku, gdy $a \cdot c \ll b$ jeśli używamy typu `float`. Te wzory to: $q \equiv -\frac{1}{2} \cdot [b + sgn(b) \cdot \sqrt{\Delta}]$, $x_1 = \frac{q}{a}$ oraz $x_2 = \frac{c}{q}$, gdzie sgn to funkcja *signum*, która daje wartość 1, gdy $b > 0$, -1 gdy $b < 0$ i 0 gdy $b = 0$. Funkcję *signum* zrealizowano w nim za pomocą zagnieżdżonych operatorów trójargumentowych. Program jest zabezpieczony na wypadek, gdyby użytkownik wprowadził zero jako wartość współczynnika a. Użyta w programie funkcja `sqrt()` pochodzi z biblioteki matematycznej włączanej za pomocą nagłówka `math.h` i liczy pierwiastek kwadratowy z podanej liczby.

Notatki

Równianie kwadratowe

Kod

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

float a,b,c,delta,q;

int main(void)
{
    puts("Podaj współczynniki równania kwadratowego");
    do {
        printf("a= ");
        scanf("%f",&a);
        if(a==0.0)
            puts("Wartość współczynnika 'a' nie może wynosić 0! Wprowadź go jeszcze raz.");
    } while(a==0.0);
    printf("b= ");
    scanf("%f",&b);
    printf("c= ");
    scanf("%f",&c);
    delta = b*b-4*a*c;
    if(delta>=0) {
        q = (b<0) ? -0.5*(b-sqrt(delta)) : (b==0.0) ? 0.0 : -0.5*(b+sqrt(delta));
        if(delta!=0.0)
            printf("x1=%f x2=%f\n",q/a,c/q);
        else
            printf("x=%f\n",q/a);
    } else
        puts("Brak rozwiązań w dziedzinie liczby rzeczywistych.");

    return 0;
}
```


Notatki

Kod dwójkowy

Opis

Czasem występuje potrzeba wypisania na ekranie reprezentacji dwójkowej liczby dziesiętnej. Niestety, standard C99 języka C nie przewiduje specjalnego ciągu formatującego dla funkcji `printf()`, który umożliwiłby zrobienie tego w prosty sposób. Na szczęście problem staje się prostszy jeśli przypomnimy sobie, że każda liczba jest w sposób dwójkowy zapisana w pamięci komputera. Trzeba tylko ten zapis „wyciągnąć" na ekran. Robi to następny przykładowy program. Wypisuje on na ekran ośmiobitową wartość zmiennej typu `char` za pomocą pojedynczej pętli `for`. W tej pętli wartości kolejnych bitów (począwszy od najstarszego) zmiennej `number` wyznaczane są w operacji maskowania (iloczynu bitowego). Drugim argumentem tej operacji jest wartość stałej `MASK` (jedynka na najstarszym bicie, pozostałe są równe zero), przesunięta w prawo o zadaną licznikiem pętli liczbę miejsc.

Notatki

Kod dwójkowy

Kod

```c
#include <stdio.h>

#define MASK 128 // 10000000

int i;
char number;

int main(void)
{
    puts("Podaj liczbę, którą chcesz wpisać w postaci binarnej");
    scanf("%hhi",&number);
    for(i=0;i<8*sizeof(number);i++)
        printf("%d",number&(MASK>>i)?1:0);
    return 0;
}
```


Notatki

Liczby pierwsze

Opis

Liczby pierwsze, to takie liczby naturalne większe od jeden, które dzielą się bez reszty wyłącznie przez jeden i przez siebie. Znajdywanie takich liczb jest na tyle skomplikowane, że duże liczby pierwsze mają zastosowanie w kryptografii. Kolejny program szuka takich liczb w przedziale od 3 do maksymalnej liczby mieszczącej się w typie `unsigned long long int`. Niestety, algorytm, który on stosuje jest mało efektywny. W najprostszej formie polega on na generowaniu kolejnych liczby naturalnych, które są dzielone przez wszystkie liczby naturalne, większe od jeden i mniejsze od nich samych. Program stosuje jego trochę udoskonaloną wersję. Liczby do sprawdzenia generowane są w zewnętrznej pętli `for`, ale są to wyłącznie liczby nieparzyste. Wewnętrzna pętla `for` dzieli je przez wszystkie liczby naturalne począwszy od 2, a skończywszy na części całkowitej pierwiastka z bieżąco badanej liczby, powiększonej o jeden. Jeśli ta liczba nie podzieli się bez reszty w trakcie takiego sprawdzania, to znaczy to, że jest pierwsza. Proszę zwrócić uwagę na użycie w programie zmiennej typu `bool` oraz instrukcji `break`.

Notatki

Liczby pierwsze

Kod

```c
#include <stdio.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>

unsigned long long int candidate, divisor;
bool prime;

int main(void)
{
    puts("Liczby pierwsze od 3");
    for(candidate=3;candidate<=ULLONG_MAX;candidate+=2) {
        prime=true;
        for(divisor=2;divisor<sqrt(candidate)+1;divisor++)
            if(candidate%divisor==0) {
                prime = false;
                break;
            }
        if(prime)
            printf("%llu ",candidate);
    }

    return 0;
}
```


Notatki

Kosinus

Opis

W bibliotece matematycznej języka C dostępna jest funkcja `cos()`, która wyznacza kosinus kąta podanego w radianach. Warto jednak wiedzieć jak wartość takiej funkcji trygonometrycznej wyznaczyć bez pomocy biblioteki matematycznej. Jedną z metod jest użycie szeregu MacLaurina, który dla funkcji `cos(x)` przyjmuje następującą postać:
$cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots + (-1)^k \cdot \frac{x^{2k}}{(2k)!} + \ldots$
Jeśli podzielimy kilka pierwszych wyrazów tego szeregu parami przez siebie, to dojdziemy do wniosku, że każdy następny różni się od poprzedniego o czynnik $-\frac{x^2}{(2i)(2i-1)}$, gdzie i określa pozycję wyrazu w szeregu, przy czym $i = 1$ ma wyraz $-\frac{x^2}{2!}$.

Notatki

Kosinus

Opis - ciąg dalszy

Program na następnym slajdzie liczy wartość kosinusa dla kąta równego $\pi/3$ radianów. W pętli `while` wartości kolejnych wyrazów są wyliczane i zapamiętywane w zmiennej `term`, zmienna `cosinus` zapamiętuje sumę wszystkich dotychczas obliczonych w pętli wyrazów szeregu, a `i` wyznacza numer pozycji kolejnego wyrazu. Pętla zatrzymuje się wtedy, gdy wartość wyliczonego z szeregu kosinusa będzie równa wartości zwróconej przez funkcję `cos()`. Tych wartości nie możemy jednak porównywać bezpośrednio, jedynie z pewną dokładnością. Tę dokładność definiuje w programie stała EPSILON - jedenaście miejsc po przecinku. Porównujemy jej wartość z błędem bezwzględnym przybliżenia wartości kosinusa, czyli wartością bezwzględną z różnicy obliczonej wartości kosinusa i podanej przez funkcję `cos()`. Wartość bezwzględna liczona jest za pomocą funkcji `fabs()`, również dostępnej w bibliotece matematycznej.

Notatki

Kosinus

Kod

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

#define EPSILON 1e-11

double cosinus = 1, term = 1, i = 1;
const double x = M_PI/3;

int main(void)
{
    while(fabs(cos(x)-cosinus)>EPSILON) {
        term *= -1.0*x*x/((2*i-1)*(2*i));
        cosinus += term;
        i++;
    }

    printf("Wartość kosinusa dla kąta %f wynosi %f\n",x,cosinus);

    return 0;
}
```


Notatki

Funkcja eksponencjalna

Opis

Podobnie jak w przypadku kosinusa możemy policzyć wartość funkcji eksponencjalnej. Dla niej szereg MacLaurina przyjmuje następującą postać:
$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^k}{k!} + \ldots$
Postępując podobnie jak poprzednio ustalimy, że każdy następny wyraz szeregu różni się od poprzedniego o czynnik $\frac{x}{i}$, gdzie $i > 0$ jest numerem pozycji wyrazu w szeregu. Program na następnym slajdzie liczy wartość eksponenty dla podanego przez użytkownika wykładnika w analogiczny sposób, jak liczył kosinusa program z poprzedniego slajdu. Główna różnica w konstrukcji pętli `while` polega na tym, że sprawdzany jest błąd względny wyliczonej z szeregu wartości i wartości funkcji `exp()` (również z biblioteki matematycznej), zamiast błędu bezwzględnego. Można w tym programie zastosować pierwszą z wymienionych metod, gdyż wartość zmiennej `exponent` nigdy nie jest równa zero.

Notatki

Funkcja eksponencjalna

Kod

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

#define EPSILON 1e-11

double exponent = 1.0, x, i=1, term = 1;

int main(void)
{
    puts("Podaj wykładnik potęgi, do której chcesz podnieść liczbę e");
    scanf("%lf",&x);
    while(fabs((exp(x)-exponent)/exponent)>EPSILON) {
        term *= (x/i);
        exponent += term;
        i++;
    }

    printf("Wartość e^x wynosi %f\n",exponent);

    return 0;
}
```


Notatki

Podziękowania

W prezentacji wykorzystałem materiały udostępnione przez dra inż. Grzegorza Łukawskiego oraz mgra inż. Leszka Ciopińskiego.

Notatki

Pytania

?

Notatki

KONIEC

Dziękuję Państwu za uwagę!

Notatki

Notatki