Politechnika
Wrocławska

Wskaźniki — próba podsumowania

wer. 14 z drobnymi modyfikacjami!

Wojciech Myszka

2025-02-17 07:55:52 +0000

Politechnika Wrocławska

Zmienna

1. Zmienna, to po prostu zmienna: miejsce w pamięci komputera do przechowywania jakiejś jednej wartości.
2. Każda zmienna ma jakiś adres, ale tymi adresami interesujemy się dosyć rzadko.

Politechnika Wrocławska

Tablica

1. Tablica to pojemnik do przechowywania wielu danych tego samego typu.
2. W komputerze — adres początku, typ elementu i numer elementu to wszystko co jest potrzebne aby dobrać się do konkretnego elementu.
3. adres(t[i]) = adres(t) + i * długość elementu
4. Z tego powodu w C tablice indeksowane są od 0 (zera) — bo pierwsza (o numerze zerowym) wartość w tablicy umieszczana jest na samym jej początku).

Politechnika Wrocławska

Adres

1. Adres — wartość jak każda inna.
2. Właściwie jest to liczba całkowita (**long int**), ale raczej nie należy (bezmyślnie) dokonywać podstawień typu **long int** $\rightarrow$ **int** $*$ lub odwrotnie!.
3. Inaczej działą arytmetyka na liczbach **long int**, a inaczej na wskaźnikach!
4. Do przechowywania adresów są potrzebne specjalne zmienne zwane „wskaźnikami" (*pointer*).
5. Do pobrania adresu obiektu służy operator jednoargumentowy & (*apersand*).
6. W języku C czym innym jest adres zmiennej int, czym innym adres zmiennej char, czym innym adres zmiennej double...
7. Do deklaracji wskaźników używamy specjalnego znaczka „$*$" (przed nazwą).
8. Żeby utrudnić wszystkim życie wymyślono też wskaźniki bez podania typu (**void**).
9. Na adresach (i zmiennych zawierających adresy) czyli wskaźnikach można wykonywać proste operacje: dodawanie i odejmowanie stałej oraz odejmowanie adresów tego samego typu.

Politechnika Wrocławska

Użycie wskaźników

1. Pobranie wartości (występuje zazwyczaj po prawej stronie znaku równości): ... = *ip.
2. Podstawienie wartości: *ip = ...

Politechnika Wrocławska

Adresowanie pośrednie

1. Adresowanie pośrednie czyli wpisanie jakieś wartości do miejsca pamięci wskazywanego przez adres (lub zmienną przechowującą adres):
   $*ip = 7;$

Politechnika Wrocławska

Funkcje I

1. Funkcje nie mają wiele wspólnego ze wskaźnikami.
2. Gdy argumentem funkcji jest zmienna, do wnętrza funkcji przekazywana jest jej wartość.
3. Gdy argumentem funkcji jest wyrażenie — do funkcji przekazywana jest jego wartość.
4. Co jest drugim argumentem funkcji
   scanf("%d", &i)
   (Nie zajmujmy się funkcją scanf, jest zbyt skomplikowana!) Co jest argumentem funkcji moja_funkcja(&i)?
5. Argumentem jest wyrażenie polegające na pobraniu adresu zmiennej i. Adres zmiennej i jest przekazywany do wnętrza funkcji!
6. Deklaracja funkcji powinna wyglądać jakoś tak:
   moja_funkcja(int *ip);

Politechnika Wrocławska

Funkcje II

7. Co funkcja może zrobić z przekazanym jej adresem? Niezbyt wiele, ale zawsze może użyć go w celu adresowania pośredniego, czyli zrobić coś takiego:
   $*ip = 127;$
   albo
   $*ip = (*ip) + 1;$
8. Parametrem funkcji może być element tablicy:
   inna_moja_funkcja(tablica[7])
   do wnętrza funkcji przekazywana jest wartość wyrażenia polegającego na pobraniu z tablicy jej siódmego elementu.

Politechnika Wrocławska

Funkcje III

9. Argumentem funkcji może być nazwa tablicy:
   inna_funkcja(tablica)
   Żeby to miało sens, deklaracja funkcji musi wyglądać jakoś tak:
   inna_funkcja(**int** t[ ]);
   Jeżeli dodatkowo funkcja znać będzie jeszcze długość tablicy — będzie mogła wykonywać na niej dowolne operacje...

```
void zerowanie(int t[ ], int n)
{
    int i;
    for(i = 0; i < n; i++)
       t[i] = 0;
}
```

Politechnika Wrocławska

Funkcje IV

```
void zerowanie(int *it, int n)
{
    int i;
    for(i = 0; i < n; i++)
       *(it + i) = 0;
}
```

Politechnika Wrocławska

Funkcje V

```
void zerowanie(int *it, int n)
{
    int i;
    for(i = 0; i < n; i++)
        it[i] = 0;
}
```

Politechnika Wrocławska

Tablice po raz drugi I

1. Tablice muszą być deklarowane.
2. W deklaracji trzeba podać ich długość.
3. Co prawda taki kod jest poprawny:

```
int n;
n = 10;
int tablica[n]
```

   ale jego działanie jest ograniczone do niezbyt wielkich n!
4. Znacznie lepsze rozwiązanie jest takie:

Politechnika Wrocławska

Tablice po raz drugi II

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

int main(void)
{
    int n, i;
    printf("Podaj␣n␣");
    scanf("%d", &n);
    int *tablica;
    tablica = malloc(n * sizeof(int));
    if (tablica != NULL) {
        for (i = 0; i < n; i++)
            tablica[i] = i;
/*  Tu reszta programu */
```

Politechnika Wrocławska

Tablice po raz drugi III

```
        free(tablica);
        return 0;
    } else {
        printf("Pomocy!␣Nie␣dostalem␣" \
               "pamieci!\n");
        return 1;
    }
}
```

Politechnika Wrocławska

Przykład pierwszy

```
int * f1(int N)
{
        int tab[N];
        return tab;
}
```

```
int main(int argc, char **argv
{
        int * T;
        int i;
        T = f1(1000);
        printf("%p\n", T);
        for(i=0;i < 1000; i++)
                T[i] = -1;
        return 0;
}
```

Politechnika Wrocławska

Przykład pierwszy prim

```
void g1(int N)
{
        int T[N];
        printf("g1: %p\n", T);
}

void g2(int M)
{
        float T[M];
        printf("g2: %p\n", T);
}
```

```
int main(int argc, char **argv
{
        g1(1000);
        g2(1000);
        return 0;
}
```

```
g1: 0x7ffcdd5f1d90
g2: 0x7ffcdd5f1d90
```

Politechnika Wrocławska

Przykład pierwszy prim

```
void g1(int N)
{
        int T[N];
        printf("g1: %p\n", T);
}

void g2(int M)
{
        float T[M];
        printf("g2: %p\n", T);
}
```

```
int main(int argc, char **argv
{
        g1(1000);
        g2(1000);
        return 0;
}
```

```
g1: 0x7ffcdd5f1d90
g2: 0x7ffcdd5f1d90
```

Politechnika Wrocławska

Przykład drugi

```
int * f3()
{
        static int tab[1000];
        return tab;
}
```

```
int main(int argc, char **argv
{
        int * T;
        int i;
        T = f3();
        printf("%p\n", T);
        for(i=0;i < 1000; i++)
                T[i] = -1;
        return 0;
}
```

Politechnika Wrocławska

Przykład trzeci

```
int * f2(int N)
{
        int * tab =
         malloc(N * sizeof(int
        return tab;
}
```

```
int main(int argc, char **argv
{
        int * T;
        int i;
        T = f2(1000);
        printf("%p\n", T);
        for(i=0;i < 1000; i++)
                T[i] = -1;
        return 0;
}
```

Politechnika Wrocławska

Przykład czwarty

```
char * f4()
{
        char * tekst=
               "Ala␣ma␣kota";
        return tekst;
}
```

```
int main(int argc, char **argv
{
        char * N;
        N = f4();
        printf("%p\n", N);
        printf("%s\n", N);
        N[1]='Z';
        return 0;
}
```

Politechnika Wrocławska

Tablice znakowe: problemy

Bardzo naturalny zapis:

```
char * a = "Ala ma kota";
char * b;
```

Czy można napisać

```
b = a;
```

Tak. Zapis oznacza tyle, że adres (wskaźnik) zawarty w zmiennej a zostaje przepisany do zmiennej b. Oba wskaźniki będą wskazywały na te same miejsce.

Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota" („a" z wyrazu „ma") zostanie przypisany do zmiennej c.

A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma" zamienić na „mm"?

Nie. Co prawda a to „tablica", ale jej zawartość jest stałą. A stałych nie można zmieniać!

Politechnika Wrocławska

Tablice znakowe: problemy

Bardzo naturalny zapis:

```
char * a = "Ala ma kota";
char * b;
```

Czy można napisać

```
b = a;
```

Tak. Zapis oznacza tyle, że adres (wskaźnik) zawarty w zmiennej a zostaje przepisany do zmiennej b. Oba wskaźniki będą wskazywały na te same miejsce.
Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota" („a" z wyrazu „ma") zostanie przypisany do zmiennej c.
A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma" zamienić na „mm"?
Nie. Co prawda a to „tablica", ale jej zawartość jest stałą. A stałych nie można zmieniać!

Politechnika Wrocławska

Tablice znakowe: problemy

Bardzo naturalny zapis:

```
char * a = "Ala ma kota";
char * b;
```

Czy można napisać

```
b = a;
```

Tak. Zapis oznacza tyle, że adres (wskaźnik) zawarty w zmiennej a zostaje przepisany do zmiennej b. Oba wskaźniki będą wskazywały na te same miejsce.

Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota" („a" z wyrazu „ma") zostanie przypisany do zmiennej c.

A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma" zamienić na „mm"?

Nie. Co prawda a to „tablica", ale jej zawartość jest stałą. A stałych nie można zmieniać!

Politechnika Wrocławska

Tablice znakowe: problemy

Bardzo naturalny zapis:

```
char * a = "Ala ma kota";
char * b;
```

Czy można napisać

```
b = a;
```

Tak. Zapis oznacza tyle, że adres (wskaźnik) zawarty w zmiennej a zostaje przepisany do zmiennej b. Oba wskaźniki będą wskazywały na te same miejsce.
Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota" („a" z wyrazu „ma") zostanie przypisany do zmiennej c.
A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma" zamienić na „mm"?
Nie. Co prawda a to „tablica", ale jej zawartość jest stałą. A stałych nie można zmieniać!

Politechnika Wrocławska

Tablice znakowe: problemy

Bardzo naturalny zapis:

```
char * a = "Ala ma kota";
char * b;
```

Czy można napisać

```
b = a;
```

Tak. Zapis oznacza tyle, że adres (wskaźnik) zawarty w zmiennej a zostaje przepisany do zmiennej b. Oba wskaźniki będą wskazywały na te same miejsce.
Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota" („a" z wyrazu „ma") zostanie przypisany do zmiennej c.
A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma" zamienić na „mm"?
Nie. Co prawda a to „tablica", ale jej zawartość jest stałą. A stałych nie można zmieniać!

Politechnika Wrocławska

Tablice znakowe: problemy

Bardzo naturalny zapis:

```
char * a = "Ala ma kota";
char * b;
```

Czy można napisać

```
b = a;
```

Tak. Zapis oznacza tyle, że adres (wskaźnik) zawarty w zmiennej a zostaje przepisany do zmiennej b. Oba wskaźniki będą wskazywały na te same miejsce.
Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota" („a" z wyrazu „ma") zostanie przypisany do zmiennej c.
A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma" zamienić na „mm"?
Nie. Co prawda a to „tablica", ale jej zawartość jest stałą. A stałych nie można zmieniać!

Politechnika Wrocławska

Tablice znakowe: problemy

Bardzo naturalny zapis:

```
char * a = "Ala ma kota";
char * b;
```

Czy można napisać

```
b = a;
```

Tak. Zapis oznacza tyle, że adres (wskaźnik) zawarty w zmiennej a zostaje przepisany do zmiennej b. Oba wskaźniki będą wskazywały na te same miejsce.
Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota" („a" z wyrazu „ma") zostanie przypisany do zmiennej c.
A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma" zamienić na „mm"?
Nie. Co prawda a to „tablica", ale jej zawartość jest stałą. A stałych nie można zmieniać!

Tablice znakowe: problemy (cd)

Inny naturalny zapis:

```
char a[] = "Ala ma kota";
char b[100];
```

Czy można napisać

```
b = a;
```

Nie. Zapis oznacza tyle, że adres (wskaźnik) związany z nazwą a chcemy przypisać do b. Ale a oraz b to stałe typu **char** $*$.
Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota" („a" z wyrazu „ma") zostanie przypisany do zmiennej c.
A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma" zamienić na „mm"?
Oczywiście. 'm' nadpisze zawartą w tablicy literę 'a'.

Tablice znakowe: problemy (cd)

Inny naturalny zapis:

```
char a[] = "Ala ma kota";
char b[100];
```

Czy można napisać

```
b = a;
```

Nie. Zapis oznacza tyle, że adres (wskaźnik) związany z nazwą a chcemy przypisać do b. Ale a oraz b to stałe typu **char** $*$.
Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota" („a" z wyrazu „ma") zostanie przypisany do zmiennej c.
A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma" zamienić na „mm"?
Oczywiście. 'm' nadpisze zawartą w tablicy literę 'a'.

Tablice znakowe: problemy (cd)

Inny naturalny zapis:

```
char a[] = "Ala ma kota";
char b[100];
```

Czy można napisać

```
b = a;
```

Nie. Zapis oznacza tyle, że adres (wskaźnik) związany z nazwą a chcemy przypisać do b. Ale a oraz b to stałe typu **char** $*$.

Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota” („a” z wyrazu „ma”) zostanie przypisany do zmiennej c.

A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma” zamienić na „mm”?

Oczywiście. 'm' nadpisze zawartą w tablicy literę 'a'.

Tablice znakowe: problemy (cd)

Inny naturalny zapis:

```
char a[] = "Ala ma kota";
char b[100];
```

Czy można napisać

```
b = a;
```

Nie. Zapis oznacza tyle, że adres (wskaźnik) związany z nazwą a chcemy przypisać do b. Ale a oraz b to stałe typu **char** $*$.
Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota" („a" z wyrazu „ma") zostanie przypisany do zmiennej c.
A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma" zamienić na „mm"?
Oczywiście. 'm' nadpisze zawartą w tablicy literę 'a'.

Politechnika Wrocławska

Tablice znakowe: problemy (cd)

Inny naturalny zapis:

```
char a[] = "Ala ma kota";
char b[100];
```

Czy można napisać

```
b = a;
```

Nie. Zapis oznacza tyle, że adres (wskaźnik) związany z nazwą a chcemy przypisać do b. Ale a oraz b to stałe typu **char** $*$.
Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota" („a" z wyrazu „ma") zostanie przypisany do zmiennej c.
A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma" zamienić na „mm"?
Oczywiście. 'm' nadpisze zawartą w tablicy literę 'a'.

Tablice znakowe: problemy (cd)

Inny naturalny zapis:

```
char a[] = "Ala ma kota";
char b[100];
```

Czy można napisać

```
b = a;
```

Nie. Zapis oznacza tyle, że adres (wskaźnik) związany z nazwą a chcemy przypisać do b. Ale a oraz b to stałe typu **char** $*$.
Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota" („a" z wyrazu „ma") zostanie przypisany do zmiennej c.
A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma" zamienić na „mm"?
Oczywiście. 'm' nadpisze zawartą w tablicy literę 'a'.

Politechnika Wrocławska

Tablice znakowe: problemy (cd)

Inny naturalny zapis:

```
char a[] = "Ala␣ma␣kota";
char b[100];
```

Czy można napisać

```
b = a;
```

Nie. Zapis oznacza tyle, że adres (wskaźnik) związany z nazwą a chcemy przypisać do b. Ale a oraz b to stałe typu **char** $*$.
Czy można napisać tak:

```
char c = a[5];
```

Tak. Piąty znak z napisu „Ala ma kota" („a" z wyrazu „ma") zostanie przypisany do zmiennej c.
A czy można napisać tak:

```
a[5] = 'm';
```

żeby wyraz „ma" zamienić na „mm"?
Oczywiście. 'm' nadpisze zawartą w tablicy literę 'a'.

Politechnika Wrocławska

Zmiana rozmiaru tablicy

Wszyscy(??) znają ten trik:

```
double x[] = {
    0., 1., 2., // 3., 4., 5.,
};
int n = sizeof ( x ) / sizeof ( double );
```

pozwalający na łatwą zmianę rozmiaru tablicy. Jest on przydatny zwłaszcza podczas testowania programów.

Politechnika Wrocławska

Dziwne zastosowania I

Wiele osób uznało to za „uniwersalną" metodę wyznaczania rozmiaru tablic. I stosuje ją w bardzo różnych sytuacjach.

1. Do wyznaczania długości tekstów:

```
char a[] = "Ala␣ma␣kota";
int n = sizeof( a ) / sizeof( char );
```

a czasami nawet sprytniej:

```
int n = sizeof( a ) / sizeof( char ) - 1;
```

Lepiej użyć funkcji strlen!
A taki prosty programik robi to samo:

```
n = 0;
while (a[n]) n++;
```

Politechnika Wrocławska

Dziwne zastosowania II

2. Wewnątrz funkcji. Czyli jakoś tak:
   W funkcji main mamy:

```
double a[] = {1., 2., 3., 4.,};
double avg = srednia( a );
```

A w funkcji:

```
double srednia (double x[])
{
        double srednia = 0.;
        int n = sizeof( x ) / sizeof( double );
        int i;
        for(i = 0; i < n; i++)
                srednia += x[i];
        return srednia / n;
```

Politechnika Wrocławska

Dziwne zastosowania III

```
}
```

I cały problem polega na tym, że mierzymy nie to o czym myślimy. Porządny kompilator zgłosi taki komunikat:

```
warning: ‘sizeof’ on array function parameter ‘x’
will return size of ‘double *’ [-Wsizeof-array-argument]
  int n = sizeof( x ) / sizeof( double );
                  ^
```

Gdyż to, co przed chwilą, było tablicą — w funkcji jest tylko i wyłącznie **adresem** początku tablicy. Resztę załatwia magia wskaźników.
Jako dobrą zasadę możemy przyjąć, że jeżeli tworzymy funkcję operującą na tablicach — jednym z parametrów **musi być** rozmiar tablicy.

Politechnika Wrocławska

Zmiana przydziału pamięci

Zadanie jest takie:

- Wczytać mamy dane nieznanej długości.
  - Może to być linia tekstu.
  - Może to być strumień danych (nieznanej objętości), który należy wczytać w całości zanim będzie mógł być przetworzony.

W instrukcjach laboratoryjnych rozważane są różne pomysły jak problem rozwiązać, tu opiszę dokładniej jedno z rozwiązań.

Funkcja czytanie I

```
1  /*
2   * napis.c
3   * Copyright 2016 wojciech myszka <myszka@norka.eu.org
4   */
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  char * czytaj(void)
9  {
```

dN (linia 10) zawiera informację o „kwancie" przydzielanych bajtów. Może to być praktycznie dowolna wartość: gdy pamięć zostanie zapełniona, przydzielony zostanie następny kwant. Gdy odczyt danych się zakończy, niezapełniona danymi pamięć będzie zwolniona.

Politechnika Wrocławska

Funkcja czytanie II

```
10 #define dN 10
11     int N = dN; // początkowy przydział
12     int i;
13 /*
14  * Przydzielamy funkcją malloc pamięć na dN znaków
15  */
16     char * bufor;
17     bufor = (char *) malloc(N);
```

Po każdym wykonaniu funkcji malloc/ calloc / realloc należy sprawdzić czy została ona wykonana poprawnie. Jakikolwiek błąd powoduje, że funkcja zwraca wartość 0. W języku C zdefiniowana jest stałą NULL typu wskaźnikowego o wartości zero. Przyjęło się, że obszar pamięci ozaczynający się od adresu 0 jest zarezerwowany i niedostępny do programów.

Funkcja czytanie III

Gdy wystąpi błąd (co raczej jest mało prawdopodobne) — funkcja napis zwraca wartość NULL co jest informacją dla programu wywołującego, że coś poszło źle.

```
18      if ( bufor == NULL )
19          return bufor;
```

Odczyt znaków wykonujemy w nieskończonej pętli (linia 21). Pętla się kończy gdy strumień danych wejściowych się skończy (warunek EOF w linii 24) lub napotkamy znak końca wiersza.
Rozpoznawanie końca wiersza ma sens gdy czytamy informację tekstową, poszczególne „rekordy" oddzielane są znakami \\n (naciśnięcia klawisza Enter). Program będzie działał gdy odrzucimy drugą część warunku w linii 24.

Funkcja czytanie IV

```
20      i = 0; // Liczba przeczytanych znaków
21      while ( 1 )
22      {
23          bufor[i] = getchar();
24          if ( bufor[i] == EOF || bufor[i] == '\n' )
```

Gdy uznajemy, że nie będzie już żadnych dodatkowych informacji — program najpierw „kończy" tekst znakiem o kodzie ASCII 0 (linia 26), a później zwalnia niewykorzystną pamięć używając funkcji realloc. Argumentem te funkcji jest liczba wykorzystanych bajtów pamięci (i przeczytanych i dodane zero, zatem i + 1).

Funkcja czytanie V

```
25          {
26              bufor[i] = 0; // koniec tekstu
27              bufor = realloc(bufor, i + 1); // zwalniam
28                                             // nadmiarową pamię
29              return bufor;
30          }
31          i++;
```

Przeczytany znak został zapisany w pamięci, zatem zwiększamy i (linia 31) oczekując na kolejny znak. Gdy okaże się że grozi przekroczenie pamięci (linia 32) — będziemy musieli przydzielić kolejny kwant pamięci. Zmienna N (linia 36) gromadzi informacje o sumarycznym rozmiarze bufora i jest odpowiednio ukatualniana.

Funkcja czytanie VI

```
32          if ( i >= N ) // Czy skonczyla sie przydzielon
33                                                  // pamiec
34          {
35              bufor = realloc(bufor, i + dN);
36              N += dN;
37              printf("%p %d\n", bufor, N);
38          }
39      }
40  }
```

Wydruk (polecenie printf w linii 37) ma charakter diagnostyczny. Aby uprościć kod nie sprawdzam, czy polecenie realloc wykonało się poprawnie (i funkcja zwróciła wartość różną od zera), ale w prawdziwych programach powinno to się robić.

Politechnika Wrocławska

Funkcja czytanie VII

```
41 int main(int argc, char **argv)
42 {
43     char * tekst;
44     char bufor[10];
45     tekst = czytaj();
46     if ( tekst != NULL ) // Sprawdzamy czy coś przeczy
47         printf("przeczytalem: %d znakow\n \"%s\"\n",
48                 (int) strlen(tekst), tekst);
49     free(tekst); // Zwalniamy przydzieloną pamięć
50     return 0;
51 }
```

Politechnika Wrocławska

Metoda połowienia

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 $[A, B]$ (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 wybierając $0 < A < 3$ i $3{,}5 < B < 6$.

Powyższe zadanie można również zaprogramować korzystając z rekurencji!

Politechnika Wrocławska

Realizacja

```
1  double polowienie(double A, double B)
2  {
3     double C;
4  pocz:
5     C = ( A + B ) / 2.;
6     if ( f(A) * f(C) < 0 )
7        B = C;
8     else if ( f(A) * f(C) > 0 )
9        A = C;
10    else
11       return C;
12    if ( fabs(A - B) > 0.001 )
13       goto pocz;
14    return C;
15 }
```

Tu jest użyta instrukcja **goto** ale można inaczej.

Politechnika Wrocławska

Inna realizacja

```
1  double polowienie(double A, double B)
2  {
3     double C;
4     do
5     {
6        C = ( A + B ) / 2.;
7        if ( f(A) * f(C) < 0 )
8           B = C;
9        else if ( f(A) * f(C) > 0 )
10          A = C;
11       else
12          return C;
13    }
14    while ( fabs(A - B) > 0.00000001 );
15    return C;
16 }
```

Z pętlą **do**.

Politechnika Wrocławska

Jeszcze inna realizacja

Zastąpimy pętlę **do** rekurencją.

```
1  double polowienie(double A, double B)
2  {
3     double C;
4        C = ( A + B ) / 2.;
5        if ( f(A) * f(C) < 0 )
6           B = C;
7        else if ( f(A) * f(C) > 0 )
8           A = C;
9        else
10          return C;
11    if ( fabs(A - B) >  0.00000001 )
12       return polowienie(A, B);
13    else
14       return C;
15 }
```

Politechnika Wrocławska

Jak z tego skorzystać?

Aby z tego programu skorzystać, trzeba jego kod dołączyć do naszego oraz napisać funkcję pomocniczą f

```
double f(double x)
{
        return sin(x);
}
```

oraz wywołać:

```
u = polowienie(2., 4.);
```

Politechnika Wrocławska

Wskaźnik do funkcji

1. Deklarazja zwykłej zmiennej wygląda tak:

```
typ nazwa;
```

2. A wskaźnik deklaruje się tak:

```
typ * inna_nazwa;
```

3. Funkcję deklarujemy tak (mam na myśli prototyp):

```
typ_funkcji nazwa_funkcji( typ_argumentu );
```

4. A wskaźnik? Przez analogię:

```
typ_funkcji * nazwa_funkcji( typ_argumentu );
```

Politechnika Wrocławska

Połowienie jeszcze inaczej

```
1  double polowienieR( double A, double B, double f(double) )
2  {
3      double C;
4      C = ( A + B ) / 2.;
5      if ( f(A) * f(C) < 0 )
6          B = C;
7      else if ( f(A) * f(C) > 0 )
8          A = C;
9      else
10         return C;
11     if ( fabs(A - B) > 0.00000001 )
12         return polowienieR(A, B, f);
13     else
14         return C;
15 }
```

Politechnika Wrocławska

Połowienie jeszcze inaczej(cd) I

```
double polowienieR( double A, double B, double f(double) )
```

Zwracam uwagę na trzeci argument w definicji funkcji polowienieR — jest to funkcja. Zatem z polowienieR można korzystać w sposób następujący:

```
double f(double x)
{
    return sin(x);
}
...
    double ( *g )(double);
    g = f;
...
    y = polowienieR(A, B, g);
```

Połowienie jeszcze inaczej(cd) II

albo

```
y = polowienieR(A, B, f);
```

albo

```
y = polowienieR(A, B, sin);
```

Dodatkowo po wykonaniu podstawienia g = f (albo g = sin) symbol (**zmienna**) g staje się „aliasem" (przezwiskiem) podstawionej funkcji.