

Wstęp do Informatyki i Programowania

Laboratorium nr 5 4, 10, 16, 17 i 18 grudnia

Zmodyfikujmy implementację listy przedstawionej na wykładzie.

- Dodajmy do struktury listy licznik elementów listy `length`, który będzie pamiętał aktualną liczbę elementów listy, i zmodyfikujmy odpowiednio procedury i funkcje, w tym szczególnie funkcję zwracającą długość listy.
- Załóżmy, że numerujemy niejawnie elementy listy od 1 do jej długości. Wprowadźmy teraz operacje
 - `get(l,i)` - zwracającą wartość i -tego elementu listy (bez modyfikacji struktury listy);
 - `put(l,i,e)` - podstawiającą nową wartość e w i -tym elemencie listy.
 - `insert(l,i,e)` - wstawiającą nowy element e między dotychczasowym $i - 1$ a i -tym (lista l wydłuża się o jeden element, dodatkowo `insert(l,1,e)=push(l,e)` i `insert(l,length(l)+1,e)=append(l,e)`).
 - `delete(l,i)` - usuwa i -ty element listy (lista l skraca się o jeden element).
- Dodajmy jeszcze procedurę `clean(l)` usuwającą wszystkie elementy listy.

Zadanie 1C (12 pkt)

Zmodyfikuj program w języku C z wykładu tak, aby `list.h` miał teraz postać z Rysunku 1 a program główny `listtest.c` umożliwiał przetestowanie biblioteki np. w takiej sesji jak na Rysunku 2 (pamiętaj aby to program główny kontrolował poprawność danych).

Zadanie 1A (12 pkt)

Zmodyfikuj program w języku Ada z wykładu tak, aby `list.ads` miał teraz postać z Rysunku 3 a program główny `listtest.adb` umożliwiał przetestowanie biblioteki np. w takiej sesji jak na Rysunku 4 (pamiętaj aby to program główny kontrolował poprawność danych).

Języki używane na wykładzie pozwalają na definiowanie typu będącego wskazaniem na funkcję. Przeanalizuj przykłady z Rysunku 5 i użyj zawarte w nich techniki do napisania poniższych zadań.

Zadanie 2C (6 pkt)

Założmy, że funkcja f typu `functype` oblicza wartość ciągłej funkcji $f(x)$.

Napisz w języku C funkcję

```
double findzero(functype f, double a, double b, double eps),
```

która dla zadanych końców przedziału $[a, b]$, gdzie $a < b$ i $f(a) \cdot f(b) < 0$ znajduje miejsce zerowe funkcji będące rozwiązaniem równania $f(x) = 0$, dla $a \leq x \leq b$, zadaną dokładnością $\varepsilon > 0$ (tzn. znaleziona wartość nie różni się od faktycznej o więcej niż ε)¹. Użyj metody bisekcji.

Przetestuj swój program dla $f(x) = \cos(x/2)$, $a = 2$, $b = 4$ i $\varepsilon \in \{10^{-k} : k = 1, 2, \dots, 8\}$.

¹Z ciągłości funkcji f i z faktu, że końce przedziałów są różnych znaków, wynika istnienie rozwiązania.

Zadanie 2A (6 pkt)

Założmy, że funkcja f typu `FuncType` oblicza wartość ciągłej funkcji $f(x)$.

Napisz w języku Ada funkcję

```
function FindZero(f : FuncType; a, b, eps: Long_Float) return Long_Float
```

która dla zadanych końców przedziału $[a, b]$, gdzie $a < b$ i $f(a) \cdot f(b) < 0$ znajduje miejsce zerowe funkcji będące rozwiązaniem równania $f(x) = 0$, dla $a \leq x \leq b$, zadaną dokładnością $\varepsilon > 0$ (tzn. znaleziona wartość nie różni się od faktycznej o więcej niż ε). Użyj metody bisekcji.

Przetestuj swój program dla $f(x) = \cos(x/2)$, $a = 2$, $b = 4$ i $\varepsilon \in \{10^{-k} : k = 1, 2, \dots, 8\}$.

Zadanie 2P (6 pkt)

Założmy, że funkcja f oblicza wartość ciągłej funkcji $f(x)$.

Napisz w języku Python funkcję

```
findZero(f, a, b, eps)
```

która dla zadanych końców przedziału $[a, b]$, gdzie $a < b$ i $f(a) \cdot f(b) < 0$ znajduje miejsce zerowe funkcji będące rozwiązaniem równania $f(x) = 0$, dla $a \leq x \leq b$, zadaną dokładnością $\varepsilon > 0$ (tzn. znaleziona wartość nie różni się od faktycznej o więcej niż ε). Użyj metody bisekcji.

Przetestuj swój program dla $f(x) = \cos(x/2)$, $a = 2$, $b = 4$ i $\varepsilon \in \{10^{-k} : k = 1, 2, \dots, 8\}$.

```

1  #pragma once
2
3  #include <stdbool.h>
4
5  typedef struct node {
6      int elem;
7      struct node* next;
8  } node;
9  typedef node* node_ptr;
10
11 typedef struct list_t {
12     node_ptr first;
13     node_ptr last;
14     int length;
15 } list_t;
16 typedef list_t* list;
17
18 bool is_empty(list l);
19
20 int pop(list l);
21 void push(list l, int e);
22 void append(list l, int e);
23
24 int get(list l, int i);
25 void put(list l, int i, int e);
26 void insert(list l, int i, int e);
27 void delete(list l, int i);
28
29 void print(list l);
30 int length(list l);
31 void clean(list l);

```

Rysunek 1: Kod list.h

```

1  szmaragd:~/lab5/c$ ./listtest
2  Command: print
3  Result:  ( 0 )
4  Command: push
5  Value: 1
6  Result: OK
7  Command: append
8  Value: 2
9  Result: OK
10 Command: insert
11 Index: 3
12 Value: 3
13 Result: OK
14 Command: insert
15 Index: 2
16 Value: 4
17 Result: OK
18 Command: print
19 Result:  1 4 2 3 ( 4 )
20 Command: get
21 Index: 3
22 Result: 2
23 Command: put
24 Index: 3
25 Value: 5
26 Result: OK
27 Command: print
28 Result:  1 4 5 3 ( 4 )

29 Command: delete
30 Index: 3
31 Result: OK
32 Command: pop
33 Result: 1
34 Command: print
35 Result:  4 3 ( 2 )
36 Command: clean
37 Result: OK
38 Command: print
39 Result:  ( 0 )
40 Command: get
41 Index: 1
42 Error - bad index!
43 Command: insert
44 Index: 2
45 Value: 2
46 Error - bad index!
47 Command: insert
48 Index: 1
49 Value: 2
50 Result: OK
51 Command: pop
52 Result: 2
53 Command: pop
54 Error - stack is empty
55 Command: exit

```

Rysunek 2: Przykładowa sesja z listtest.c

```

1  with Ada.Unchecked_Deallocation;
2
3  package list is
4      type ListT is private;
5
6      function isEmpty (l : ListT) return Boolean;
7
8      function Pop (l : in out ListT) return Integer;
9      procedure Push (l : in out ListT; e : Integer);
10     procedure Append (l : in out ListT; e : Integer);
11
12     function Get (l : ListT; i : Integer) return Integer;
13     procedure Put (l : in out ListT; i : Integer; e : Integer);
14     procedure Insert (l : in out ListT; i : Integer; e : Integer);
15     procedure Delete (l : in out ListT; i : Integer);
16
17     procedure Print (l : ListT);
18     procedure Clean (l : in out ListT);
19     function Length (l : ListT) return Integer;
20 private
21     type Node;
22     type NodePtr is access Node;
23     type Node is record
24         elem : Integer := 0;
25         next : NodePtr := null;
26     end record;
27
28     type ListT is record
29         first : NodePtr := null;
30         last  : NodePtr := null;
31         length : Integer := 0;
32     end record;
33
34     procedure Free is
35         new Standard.Ada.Unchecked_Deallocation (Node, NodePtr);
36 end list;

```

Rysunek 3: Kod list.ads

```

1  szmaragd:~/lab5/ada$ ./listtest
2  Command: Print
3  Result:  ( 0 )
4  Command: Push
5  Value: 1
6  Result: OK
7  Command: Append
8  Value: 2
9  Result: OK
10 Command: Insert
11 Index: 3
12 Value: 3
13 Result: OK
14 Command: Insert
15 Index: 2
16 Value: 4
17 Result: OK
18 Command: Print
19 Result:  1 4 2 3 ( 4 )
20 Command: Get
21 Index: 3
22 Result:  2
23 Command: Put
24 Index: 3
25 Value: 5
26 Result: OK
27 Command: Print
28 Result:  1 4 5 3 ( 4 )

29 Command: Delete
30 Index: 3
31 Result: OK
32 Command: Pop
33 Result:  1
34 Command: Print
35 Result:  4 3 ( 2 )
36 Command: Clean
37 Result: OK
38 Command: Print
39 Result:  ( 0 )
40 Command: Get
41 Index: 1
42 Error - bad index!
43 Command: Insert
44 Index: 2
45 Value: 2
46 Error - bad index!
47 Command: Insert
48 Index: 1
49 Value: 2
50 Result: OK
51 Command: Pop
52 Result:  2
53 Command: Pop
54 Error - stack is empty!
55 Command: Exit

```

Rysunek 4: Przykładowa sesja z listtest.adb

```

1  #include <stdio.h>
2  #include <math.h>
3  typedef double (*functype)(double);
4  void write(functype f, double x) {
5      printf("%f\n", f(x));
6  }
7  double square1(double x) {
8      return (x-1.0)*(x+2.0);
9  }
10 int main() {
11     double x;
12     scanf("%lf", &x);
13     write(sin, x);
14     write(cos, x);
15     write(sqrt, x);
16     write(square1, x);
17     return 0;
18 }

```

```

1  with Ada.Text_IO; use Ada.Text_IO;
2  with Ada.Long_Float_Text_IO; use Ada.Long_Float_Text_IO;
3  with Ada.Numerics.Long_Elementary_Functions;
4  use Ada.Numerics.Long_Elementary_Functions;
5  procedure FuncTest is
6      type FuncType is access function (x : Long_Float) return Long_Float;
7      procedure Write (f : FuncType; x : Long_Float) is
8          begin
9              Put_Line (f (x)'Image);
10         end Write;
11         function Square1 (x : Long_Float) return Long_Float is
12             begin
13                 return (x - 1.0) * (x + 2.0);
14             end Square1;
15         x : Long_Float;
16     begin
17         Get (x);
18         Write (Sin'Access, x);
19         Write (Cos'Access, x);
20         Write (Sqrt'Access, x);
21         Write (Square1'Access, x);
22     end FuncTest;

```

```

1  from math import *
2
3  def write(f, x) :
4      print(f(x))
5
6  def square1(x) :
7      return (x-1.0)*(x+2.0)
8
9  def main() :
10     x=float(input(""))
11     write(sin, x)
12     write(cos, x)
13     write(sqrt, x)
14     write(square1, x)
15
16 if __name__ == "__main__":
17     main()

```

Rysunek 5: Przykładowe programy z typem funkcyjnym