Wstęp do Informatyki i Programowania

Laboratorium nr 5
4, 5, 10 grudnia i 7, 8, 9 stycznia

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 1 (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 2 (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 funkcje. Przeanalizuj przykłady z Rysunku 5 i użyj zawarte w nich techniki do napisania poniższych zadań.

Zadanie 3 (6 pkt)
Załóżmy, 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 $a\leqslant x\leqslant b$, z 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 $\varepsilon\in\{10^{-k}\ :\ k=1,2,\ldots,8\}$.

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

Zadanie 4 (6 pkt)

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

Napisz w języku Ada funkcję

```
function FindZero(f : FuncType; a, b, eps: Float) return 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 $a \leqslant x \leqslant b$, z 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 $\varepsilon \in \{10^{-k} \ : \ k = 1, 2, \ldots, 8\}$.

Zadanie 5 (6 pkt)

Załóżmy, 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 $a \leqslant x \leqslant b$, z 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 $\varepsilon \in \{10^{-k} \ : \ k = 1, 2, \ldots, 8\}$.

```c
#pragma once

#include <stdbool.h>

typedef struct node {
  int elem;
  struct node* next;
} node;
typedef node* node_ptr;

typedef struct list_t {
  node_ptr first;
  node_ptr last;
  int length;
} list_t;
typedef list_t* list;

bool is_empty(list l);

int  pop(list l);
void push(list l, int e);
void append(list l, int e);

int  get(list l, int i);
void put(list l, int i, int e);
void insert(list l, int i, int e);
void delete(list l, int i);

void print(list l);
int  length(list l);
void clean(list l);
```

Rysunek 1: Kod `list.h`

```
szmaragd:~/lab5/c$ ./listtest
Command: print
Result:  ( 0 )
Command: push
Value: 1
Result: OK
Command: append
Value: 2
Result: OK
Command: insert
Index: 3
Value: 3
Result: OK
Command: insert
Index: 2
Value: 4
Result: OK
Command: print
Result:  1 4 2 3 ( 4 )
Command: get
Index: 3
Result: 2
Command: put
Index: 3
Value: 5
Result: OK
Command: print
Result:  1 4 5 3 ( 4 )
Command: delete
Index: 3
Result: OK
Command: pop
Result: 1
Command: print
Result:  4 3 ( 2 )
Command: clean
Result: OK
Command: print
Result:  ( 0 )
Command: get
Index: 1
Error - bad index!
Command: insert
Index: 2
Value: 2
Error - bad index!
Command: insert
Index: 1
Value: 2
Result: OK
Command: pop
Result: 2
Command: pop
Error - stack is empty
Command: exit
```

Rysunek 2: Przykładowa sesja z `listtest.c`

```
with Ada.Unchecked_Deallocation;

package list is
   type ListT is private;

   function isEmpty (l : ListT) return Boolean;

   function Pop (l : in out ListT) return Integer;
   procedure Push (l : in out ListT; e : Integer);
   procedure Append (l : in out ListT; e : Integer);

   function Get (l : ListT; i : Integer) return Integer;
   procedure Put (l : in out ListT; i : Integer; e : Integer);
   procedure Insert (l : in out ListT; i : Integer; e : Integer);
   procedure Delete (l : in out ListT; i : Integer);

   procedure Print (l : ListT);
   procedure Clean (l : in out ListT);
   function Length (l : ListT) return Integer;
private
   type Node;
   type NodePtr is access Node;
   type Node is record
      elem : Integer := 0;
      next : NodePtr := null;
   end record;

   type ListT is record
      first : NodePtr := null;
      last  : NodePtr := null;
      length : Integer := 0;
   end record;

   procedure Free is
      new Standard.Ada.Unchecked_Deallocation (Node, NodePtr);
end list;
```

Rysunek 3: Kod `list.ads`

```
szmaragd:~/lab5/ada$ ./listtest
Command: Print
Result:  ( 0 )
Command: Push
Value: 1
Result: OK
Command: Append
Value: 2
Result: OK
Command: Insert
Index: 3
Value: 3
Result: OK
Command: Insert
Index: 2
Value: 4
Result: OK
Command: Print
Result:  1 4 2 3 ( 4 )
Command: Get
Index: 3
Result:  2
Command: Put
Index: 3
Value: 5
Result: OK
Command: Print
Result:  1 4 5 3 ( 4 )
Command: Delete
Index: 3
Result: OK
Command: Pop
Result:  1
Command: Print
Result:  4 3 ( 2 )
Command: Clean
Result: OK
Command: Print
Result:  ( 0 )
Command: Get
Index: 1
Error - bad index!
Command: Insert
Index: 2
Value: 2
Error - bad index!
Command: Insert
Index: 1
Value: 2
Result: OK
Command: Pop
Result:  2
Command: Pop
Error - stack is empty!
Command: Exit
```

Rysunek 4: Przykładowa sesja z `listtest.adb`

```c
#include <stdio.h>
#include <math.h>
typedef double (*functype)(double);
void write(functype f, double x) {
  printf("%f\n", f(x));
}
double square1(double x) {
  return (x-1.0)*(x+2.0);
}
int main() {
  double x;
  scanf("%lf", &x);
  write(sin, x);
  write(cos, x);
  write(sqrt, x);
  write(square1, x);
  return 0;
}
```

```ada
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Float_Text_IO; use Ada.Float_Text_IO;
with Ada.Numerics.Elementary_Functions; use Ada.Numerics.Elementary_Functions;
procedure FuncTest is
   type FuncType is access function (x : Float) return Float;
   procedure Write (f : FuncType; x : Float) is
   begin
      Put_Line (f (x)'Image);
   end Write;
   function Square1 (x : Float) return Float is
   begin
      return (x - 1.0) * (x + 2.0);
   end Square1;
   x : Float;
begin
   Get (x);
   Write (Sin'Access, x);
   Write (Cos'Access, x);
   Write (Sqrt'Access, x);
   Write (Square1'Access, x);
end FuncTest;
```

```python
from math import *

def write(f, x) :
    print(f(x))

def square1(x) :
    return (x-1.0)*(x+2.0)

def main() :
    x=float(input(""))
    write(sin, x)
    write(cos, x)
    write(sqrt, x)
    write(square1, x)

if __name__ == "__main__":
    main()
```

Rysunek 5: Przykładowe programy z typem funkcyjnym