Dynamiczne struktury danych

Wstęp do Informatyki i Programowania

Maciek Gębala

28 listopada 2024

Notatki

Stos i kolejka

Stos (LIFO - Last In First Out)

Struktura danych z operacjami wkładania (push) i ściągania (pop) elementów, z zachowanym porządkiem, że pierwszy ściągany jest element ostatnio włożony.
Dodatkowo mamy operację sprawdzania czy struktura jest pusta (operacja ściągnięcia elementu z pustej struktury wywołuje błąd).

Kolejka (FIFO - First In First Out)

Struktura danych z operacjami wkładania (append, push_back) i ściągania (pop) elementów, z zachowanym porządkiem, że pierwszy ściągany jest element włożony jako pierwszy.
Dodatkowo mamy operację sprawdzania czy struktura jest pusta (operacja ściągnięcia elementu z pustej struktury wywołuje błąd).

Zakładamy, że nie mamy ograniczeń na liczbę elementów przechowywanych w strukturze.

Notatki

Listy jednokierunkowe

Lista jednokierunkowa składa się z węzłów, węzeł reprezentowany jest strukturą posiadającą dwa pola: `elem` przechowującą dane węzła i `next` przechowującą wskazanie na następny węzeł na liście (lub `null` (pusty wskaźnik) w przypadku gdy nie ma kolejnego elementu).

Aby dostać się do węzłów musimy mieć wskaźnik `first` wskazujący pierwszy węzeł na liście (jeśli first jest pustym wskaźnikiem, to lista jest pusta).

W niektórych przypadkach przydaje się też wskaźnik `last` na ostatni element listy.

Notatki

Implementacja w Adzie i C

Chcemy zaimplementować w Adzie i C bibliotekę, która ma funkcję zarówno stosu jak i kolejki dla liczb całkowitych oraz będzie oparta na listach jednokierunkowych.

Dodatkowo dodamy procedurę umożliwiającą wypisanie całej listy.

Notatki

list.ads

```
1  with Ada.Unchecked_Deallocation;
2
3  package list is
4     type ListT is private;
5
6     function isEmpty (l : ListT) return Boolean;
7     function Pop (l : in out ListT) return Integer;
8     procedure Push (l : in out ListT; e : Integer);
9     procedure Append (l : in out ListT; e : Integer);
10
11    procedure Print (l : ListT);
12    function Length (l : ListT) return Integer;
```


Notatki

list.ads

```
13 private
14    type Node;
15    type NodePtr is access Node;
16    type Node is record
17       elem : Integer := 0;
18       next : NodePtr := null;
19    end record;
20
21    type ListT is record
22       first : NodePtr := null;
23       last  : NodePtr := null;
24    end record;
25
26    procedure Free is
27       new Standard.Ada.Unchecked_Deallocation (Node, NodePtr);
28 end list;
```


Notatki

list.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2
3  package body list is
4     function isEmpty (l : ListT) return Boolean is
5     begin
6        return l.first = null;
7     end isEmpty;
8
9     function Pop (l : in out ListT) return Integer is
10       n : NodePtr := l.first;
11       e : Integer := n.elem;
12    begin
13       l.first := n.next;
14       if l.first = null then -- last element
15          l.last := null;
16       end if;
17       Free (n);
18       return e;
19    end Pop;
```


Notatki

list.adb

```
21    procedure Push (l : in out ListT; e : Integer) is
22       n : NodePtr := new Node;
23    begin
24       n.elem := e;
25       n.next := l.first;
26       l.first := n;
27       if l.last = null then -- first element
28          l.last := n;
29       end if;
30    end Push;
31
32    procedure Append (l : in out ListT; e : Integer) is
33       n : NodePtr := new Node;
34    begin
35       n.elem := e;
36       if l.first = null then -- first element
37          l.first := n;
38       else
39          l.last.next := n;
40       end if;
41       l.last := n;
42    end Append;
```


Notatki

list.adb

```
44    procedure Print (l : ListT) is
45       n : NodePtr := l.first;
46    begin
47       while n /= null loop
48          Put (n.elem'Image);
49          n := n.next;
50       end loop;
51       Put_Line ("␣(" & Length (l)'Image & "␣)");
52    end Print;
53
54    function Length (l : ListT) return Integer is
55       i : Integer := 0;
56       n : NodePtr := l.first;
57    begin
58       while n /= null loop
59          i := i + 1;
60          n := n.next;
61       end loop;
62       return i;
63    end Length;
64 end list;
```


Notatki

listtest.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2  with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
3  with Ada.Strings.Unbounded; use  Ada.Strings.Unbounded;
4  with Ada.Strings.Unbounded.Text_IO; use Ada.Strings.Unbounded.Text_IO;
5
6  with list; use list;
7
8  procedure listTest is
9     l : ListT;
10    r : Integer;
11    command : Unbounded_String;
12    continue : Boolean := True;
13 begin
14    while continue loop
15       Put ("Command:␣");
16       Get_Line (command);
17       if command = "Pop" then
18          if not isEmpty (l) then
19             r := Pop (l);
20             Put_Line ("Result:␣" & r'Image);
21          else
22             Put_Line ("Error␣-␣stack␣is␣empty!");
23          end if;
```


Notatki

listtest.adb

```
24       elsif command = "Push" then
25          Put ("Value:␣");
26          Get (r);
27          Skip_Line;
28          Push (l, r);
29          Put_Line ("Result:␣OK");
30       elsif command = "Append" then
31          Put ("Value:␣");
32          Get (r);
33          Skip_Line;
34          Append (l, r);
35          Put_Line ("Result:␣OK");
36       elsif command = "Print" then
37          Put ("Result:␣");
38          Print (l);
39       elsif command = "Length" then
40          r := Length (l);
41          Put_Line ("Result:␣" & r'Image);
42       elsif command = "Exit" then
43          continue := False;
44       else
45          Put_Line ("Unknown␣command!");
46       end if;
47    end loop;
```


Notatki

listtest.adb

```
48    --  clean list
49    while not isEmpty (l) loop
50       r := Pop (l);
51    end loop;
52 end listTest;
```


Notatki

list.h

```
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 } list_t;
15 typedef list_t* list;
16
17 bool is_empty(list l);
18 int  pop(list l);
19 void push(list l, int e);
20 void append(list l, int e);
21
22 void print(list l);
23 int  length(list l);
```


Notatki

list.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "list.h"
4
5  bool is_empty(list l) {
6    return l->first == NULL;
7  }
8  int pop(list l) {
9    node_ptr n = l->first;
10   int e = n->elem;
11   l->first = l->first->next;
12   if (l->first == NULL)  // last element
13     l->last = NULL;
14   free(n);
15   return e;
16 }
17 void push(list l, int e) {
18   node_ptr n = malloc(sizeof(node));
19   n->elem = e;
20   n->next = l->first;
21   l->first = n;
22   if (l->last == NULL)  // first element
23     l->last = n;
24 }
```


Notatki

list.c

```
25 void append(list l, int e) {
26   node_ptr n = malloc(sizeof(node));
27   n->elem = e;
28   if (l->first == NULL)  // first element
29     l->first = n;
30   else
31     l->last->next = n;
32   l->last = n;
33 }
34 void print(list l) {
35   node_ptr n = l->first;
36   while (n != NULL) {
37     printf("␣%d", n->elem);
38     n = n->next;
39   }
40   printf("␣(␣%d␣)\n", length(l));
41 }
```


Notatki

list.c

```
42 int length(list l) {
43   int i = 0;
44   node_ptr n = l->first;
45   while (n != NULL) {
46     i = i + 1;
47     n = n->next;
48   }
49   return i;
50 }
```


Notatki

listtest.c

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  #include <stdbool.h>
5  #include "list.h"
6
7  int main() {
8    char command[20];
9    bool cont = true;
10   int r;
11   list l = malloc(sizeof(list_t));
12   l->first = l->last = NULL;
13   while (cont) {
14     printf("Command: ");
15     scanf("%s", command);
16     if (!strcmp(command, "pop")) {
17       if (!is_empty(l)) {
18         r = pop(l);
19         printf("Result: %d\n", r);
20       } else {
21         printf("Error - stack is empty!\n");
22       }
23     }
```


Notatki

listtest.c

```
24     else if (!strcmp(command, "push")) {
25       printf("Value: ");
26       scanf("%d", &r);
27       push(l, r);
28       printf("Result: OK\n");
29     }
30     else if (!strcmp(command, "append")) {
31       printf("Value: ");
32       scanf("%d", &r);
33       append(l, r);
34       printf("Result: OK\n");
35     }
36     else if (!strcmp(command, "print")) {
37       printf("Result: ");
38       print(l);
39     }
40     else if (!strcmp(command, "length")) {
41       r = length(l);
42       printf("Result: %d\n", r);
43     }
44     else if (!strcmp(command, "exit")) {
45       cont = false;
46     }
```


Notatki

listtest.c

```
47     else
48       printf("Unknown command!\n");
49   }
50   while (!is_empty(l))
51     pop(l);
52   free(l);
53
54   return 0;
55 }
```


Notatki

Inne operacje

Na ćwiczeniach i laboratorium uzupełnimy listę operacji dla listy jednokierunkowej.

Notatki

Listy - rozszerzenia

Lista dwukierunkowa

Lista dwukierunkowa składa się z węzłów posiadających trzy pola: `elem` przechowuje dane, `next` przechowuje wskazanie na następny węzeł na liście, a `prev` przechowuje wskazanie na poprzedni węzeł na liście.

Listy cykliczne

Listą cykliczną nazywamy listę, w której za ostatnim elementem jest jej pierwszy element.

Notatki

Notatki

Notatki

Notatki