الجمهورية اليمنية

وزارة التعليم العالي

والبحث العلمي

جامعة صنعاء

كلية الحاسوب وتقنية المعلومات

قسم علوم حاسوب

2024-2025

تكليف/

الأستاذة

هبة المروعي

الطالب/

طماح العبدي

24160116

Memory Management in Structs (Assignment 1)

In C/C++, data inside a struct is stored sequentially, but alignment (padding) may be added to ensure efficient memory access. This can increase the total size of the struct beyond the sum of its individual field sizes.

To reduce memory consumption, directives like #pragma pack(1) or __attribute__((packed)) can be used, but they may negatively affect performance due to unaligned memory access.

Pointers within struct do not store the actual data but instead hold memory addresses that point to the data. Understanding memory allocation in structs helps optimize both memory usage and performance, especially when dealing with dynamically allocated memory on the heap.

Time Complexity of Functions (Assignment 2)

1. O(1) - Constant Time Complexity

- Accessing the first element of an array takes constant time regardless of the array size.

```
int getFirstElement(int array[]) {
  return array[0];
}
```

2. O(log n) - Logarithmic Time Complexity

- Binary search divides the range into two halves at each step, making it more efficient than linear search.

```
int binarySearch(int array[], int start, int end, int target) {
  while (start <= end) {
    int mid = start + (end - start) / 2;
    if (array[mid] == target) return mid;
    if (array[mid] < target) start = mid + 1;
    else end = mid - 1;
  }
  return -1;
}
```

3. O(n) - Linear Time Complexity

- Searching for the maximum value requires checking all elements.

```
int findMaximum(int array[], int size) {
   int maximum = array[0];
   for (int i = 1; i < size; i++) {
      if (array[i] > maximum) maximum = array[i];
   }
   return maximum;
}
```

4. O(n log n) - Linearithmic Time Complexity

- QuickSort is an efficient sorting algorithm that recursively partitions the array.

```
void quickSort(int array[], int low, int high) {
   if (low < high) {
      int pivotIndex = partition(array, low, high);
      quickSort(array, low, pivotIndex - 1);
      quickSort(array, pivotIndex + 1, high);
   }
}
```

5. O(n²) - Quadratic Time Complexity

- Selection Sort repeatedly selects the smallest element and moves it to the correct position.

```
void selectionSort(int array[], int size) {
   for (int i = 0; i < size - 1; i++) {
      int minIndex = i;
      for (int j = i + 1; j < size; j++) {
         if (array[j] < array[minIndex]) minIndex = j;
      }
      swap(array[minIndex], array[i]);
   }
}
```

6. O(2^n) - Exponential Time Complexity

- The recursive Fibonacci sequence leads to repeated calculations, making it inefficient.

```
int fibonacci(int n) {
```

```
    if (n <= 1) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

7. O(n!) - Factorial Time Complexity

- Generating all permutations of an array grows extremely fast as n increases.

```
void generatePermutations(int array[], int start, int end) {
    if (start == end) {
        printArray(array, end + 1);
    } else {
        for (int i = start; i <= end; i++) {
            swap(array[start], array[i]);
            generatePermutations(array, start + 1, end);
            swap(array[start], array[i]);
        }
    }
}
```

Balanced Parentheses Check (Assignment 3)

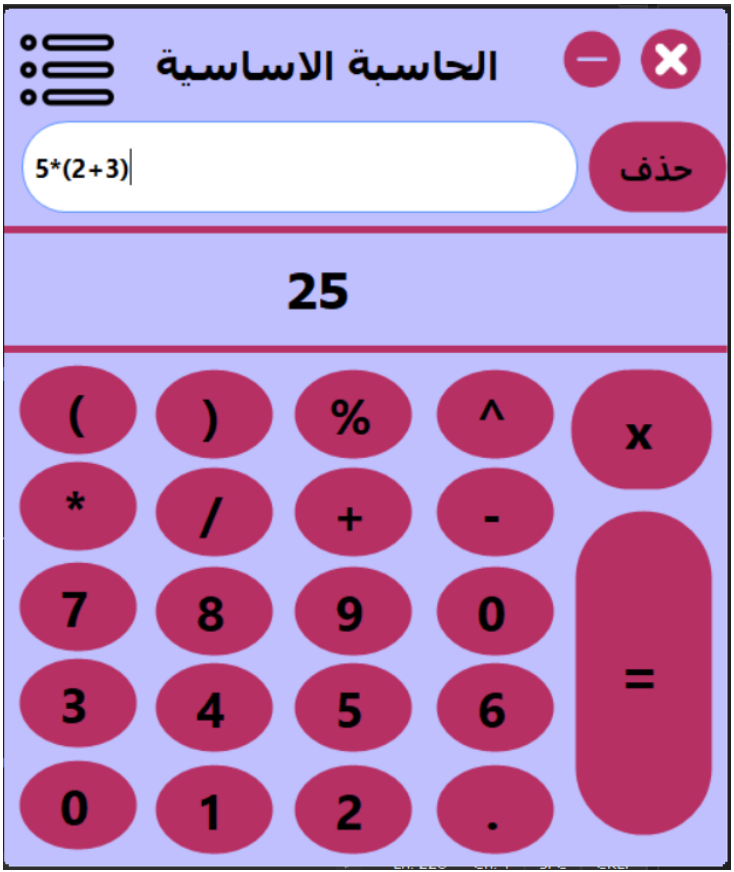
This function verifies whether an expression contains correctly matched parentheses, curly braces, and square brackets using a stack.

```
#include <iostream>
#include <stack>
using namespace std;

bool isBalanced(string expression) {
    stack<char> stack;
    for (char ch : expression) {
        if (ch == '(' || ch == '{' || ch == '[') {
            stack.push(ch);
        } else if (ch == ')' || ch == '}' || ch == ']') {
            if (stack.empty()) return false;
```

```
            char top = stack.top();
            stack.pop();
            if ((ch == ')' && top != '(') ||
               (ch == '}' && top != '{') ||
               (ch == ']' && top != '[')) {
               return false;
            }
         }
      }
      return stack.empty();
}
```

الان سأقدم لكم برنامج مكتبي متكامل للعمليات الحسابية مع الواجهة مستخدما الخوارزميات الشهيرة لتحويل التعبير الرياضي الطبيعي الى التعبير prefix and postfix expressions بكل الحالات الممكنة ويتم تخزين العمليات في قاعدة البيانات لكي يتم استرجاعها لاحقا وضهور اخر العمليات التي تم تسجيلها في الالة الحاسبة



```
 1  using Guna.UI2.WinForms;
 2  using System;
 3  using System.Collections.Generic;
 4  using System.ComponentModel;
 5  using System.Data;
 6  using System.Drawing;
 7  using System.Linq;
 8  using System.Text;
 9  using System.Threading.Tasks;
10  using System.Windows.Forms;
11  using System.Data.SqlClient;
12  using DataAccessLayer;
13
14  namespace calclator
15  {
16      public partial class Form1 : Form
17      {
18
19
20
21          static int Precedentce(char op)
22          {
23
24              if (op == '+' || op == '-')
25                  return 1;
26              else if (op == '*' || op == '/' || op == '%') return 2;
27              else if (op == '^') return 3;
28              else return 0;
29
30          }
31
32          static string Printstack(Stack<string> stack)
33          {
34              string ex = "";
35              while (stack.Count() > 0)
36                  ex = stack.Pop().ToString() + ex;
37              return ex;
38
39          }
40          static double CalclateArithemtic(double num1, double num2, char ↵
               op)
41          {
42              if (op == '+')
43                  return num1 + num2;
44              else if (op == '-')
45                  return num1 - num2;
46              else if (op == '*')
47                  return num1 * num2;
48              else if (op == '/')
49                  if (num2 == 0)
50                  {
51                      MessageBox.Show("invalid divided on zero");
52                      //throw new Exception("it is zero");
```

```
53                      return num1 / num2;
54
55                  }
56                  else
57                  return num1 / num2;
58
59                  else if (op == '^')
60                      return Math.Pow(num1, num2);
61
62                  else return 0;
63
64          }
65          static Stack<string> ConvertToPostfix(string exp)
66          {
67              Stack<string> output = new Stack<string>();
68              Stack<char> stack = new Stack<char>();
69              string number = "";
70              foreach (char token in exp)
71              {
72                  if (!char.IsWhiteSpace(token))
73                      if (char.IsLetterOrDigit(token))
74                          number += token.ToString();
75                      else if (token == '(')
76
77                          stack.Push(token);
78                      else if (token == ')')
79                      {
80                          output.Push(number);
81                          number = string.Empty;
82                          while (stack.Count() > 0 && (stack.First() !=
                    '('))
83                              output.Push(stack.Pop().ToString());
84                          try
85                          {
86                              stack.Pop();
87                          }
88                          catch (Exception e)
89                          {
90                              //Console.WriteLine(e.ToString());
91                              MessageBox.Show("the expression is not
                    valid !");
92
93                          }
94                      }
95                      else
96                      {
97                          output.Push(number);
98                          number= string.Empty;
99
100                         while (stack.Count() > 0 && (Precedentce
                    (stack.First()) >= Precedentce(token)))
101                             output.Push(stack.Pop().ToString());
102                         stack.Push(token);
```

```
103                     }
104             }
105             if (number != string.Empty)
106             {
107                 output.Push(number);
108                 number = string.Empty;
109             }
110             while (stack.Count() > 0)
111                 if (output.First() == '('.ToString())
112                 {
113                     //throw new Exception("the parenthese is not matching !");
114                     MessageBox.Show(" the parenthese is not matching !");
115                     break;
116                 }
117                 else
118                     output.Push(stack.Pop().ToString());
119
120
121
122
123
124
125             return ReserveStack(output);
126         }
127         static double Calclatepostfix(Stack<string> ex)
128         {
129             ex.Reverse<string>();
130             Stack<double> oprands = new Stack<double>();
131             string[] operators ="/,+,*,-,^,%".Split(',');
132             double num1 = 0;
133             double num2 = 0;
134             string c;
135             while (ex.Count()>0)
136             {
137                 c=ex.Pop();
138                 if (c.All(char.IsDigit))
139                     oprands.Push((int.Parse(c.ToString())));
140                 else if (Array.Exists(operators, op => op == c))
141                 {
142                     num2 = Convert.ToDouble(oprands.Pop());
143                     num1 = Convert.ToDouble(oprands.Pop());
144
145                     oprands.Push(CalclateArithemtic(num1, num2, Convert.ToChar(c)));
146                 }
147             }
148             return oprands.Pop();
149
150         }
151         static Stack<string> ReserveStack(Stack<string> stack) {
152             Stack<string> ret = new Stack<string>();
```

```
153             foreach (string item in stack)
154             {
155                 ret.Push(item);
156             }
157 
158             return ret ;
159         }
160         static string ReplaceParenthece(string exp)
161         {
162             char[] chars = exp.ToCharArray();
163             Array.Reverse(chars);
164             //exp=new string(exp.Reverse().ToArray());
165 
166             for (int i = 0; i < chars.Length; i++)
167             {
168                 if (chars[i] == '(')
169                 {
170                     chars[i] = ')';
171                 }
172                 else if (chars[i] == ')')
173                     chars[i] = '(';
174 
175             }
176             return new string(chars);
177         }
178         static string ReverseExpression(string exp)
179         {
180             char[] chars = exp.ToCharArray();
181             Array.Reverse(chars);
182             return new string(chars);
183         }
184         static double CalclatePrefix(Stack<string> exp)
185         {
186 
187             Stack<double> operands = new Stack<double>();
188             Stack<string> operators = new Stack<string>();
189             string[] ops = { "+", "-", "*", "/","%", "^" };
190 
191 
192             foreach (string c in exp)
193             {
194 
195                 if (c.All(char.IsDigit))
196                     operands.Push(double.Parse(c.ToString()));
197 
198                 else if (Array.Exists(ops, item => item == c))
199                 {
200                     operators.Push(c);
201                 }
202             }
203             double num1 = 0;
204             double num2 = 0;
205             char op;
```

```
206             while (operators.Count > 0)
207             {
208                 num2 = operands.Pop();
209                 num1 = operands.Pop();
210                 op =Convert.ToChar( operators.Pop());
211                 operands.Push(CalclateArithemtic(num1, num2, op));
212             }
213 
214 
215 
216             return operands.Pop();
217         }
218 
219         public Form1()
220         {
221             InitializeComponent();
222         }
223 
224         private void Form1_Load(object sender, EventArgs e) {
225 
226 
227         }
228 
229         private void guna2GradientButton1_Click(object sender,
              EventArgs e)
230         {
231             label2.Text= Calclatepostfix(ConvertToPostfix
                  (gtxt.Text.ToString())).ToString();
232             //label1.Text = Printstack(ConvertToPostfix
                  (gtxt.Text.ToString()));
233 
234 
235 
236         }
237 
238         private void guna2CircleButton1_Click(object sender, EventArgs
              e)
239         {
240 
241             Guna2CircleButton button =  (Guna2CircleButton)sender;
242 
243             string[] operators = "/,*,-,+,^,%".Split(',');
244 
245             if (gtxt.Text.Length == 0 && Array.Exists
                  (operators,item=>item==button.Text.ToString())) {
246                 MessageBox.Show("must start with number or
                      parentheces !");
247             }
248             else
249             {
250                 gtxt.Text += button.Text.ToString();
251             }
252         }
```

```
253
254         private void guna2ControlBox1_Click(object sender, EventArgs e)
255         {
256
257         }
258
259         private void lblreasalt_Click(object sender, EventArgs e)
260         {
261
262         }
263
264         private void gdelete_Click(object sender, EventArgs e)
265         {
266             gtxt.Text= null;
267             lblreasalt.Text= "??";
268         }
269
270         private void geqqul_Click(object sender, EventArgs e)
271         {
272             lblreasalt.Text=Calclatepostfix(ConvertToPostfix
                (gtxt.Text)).ToString();
273         }
274
275         private void gredo_Click(object sender, EventArgs e)
276         {
277             gtxt.Text = gtxt.Text.Substring(0,gtxt.Text.Length-1);
278
279         }
280
281         private void guna2ImageButton1_Click(object sender, EventArgs
              e)
282         {
283             Form frm = new frmRecord();
284             frm.ShowDialog();
285         }
286
287         private void guna2Separator2_Click(object sender, EventArgs e)
288         {
289
290         }
291     }
292 }
293
```