Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра системного програмування
і спеціалізованих комп’ютерних системи

Лабораторна робота №1.2

з дисципліни
«Архітектура для програмістів»

Виконав:
студент групи КВ-11
Терентьєв Іван Дмитрович

Перевірив:
Молчанов О. А.

Київ – 2024

Загальне завдання

1. Реалізувати програму сортування масиву згідно із варіантом мовою Java (інформація про варіанти наведена в п. 4). Програма, записана мовою Java, має бути синтаксично і семантично відповідна програмі, записаній мовою C (це досягається завдяки синтаксичній та семантичній подібності цих двох мов для випадку даного завдання), що була реалізована в лабораторній роботі No1.1. Результатом виконання цього пункту є лістинг програми мовою Java.
2. Виконати трансляцію програми, написаної мовою Java, у байт-код Java за допомогою javac і javap (програми, що постачаються разом з пакетом openjdk) й встановити семантичну відповідність між командами мови Java та командами одержаного байт-коду Java шляхом додавання коментарів з поясненням. Результатом виконання даного пункту буде лістинг байт-коду програми із коментарями в коді, в яких наведено відповідний код програми, записаної мовою Java.
3. Розібратись і вміти пояснити, що виконують ті чи інші команди байт-коду Java, що будуть присутні в байт-коді програми, отриманого в другому пункті загального завдання; вміти пояснити зв’язок між кодом мовою Java та байт-кодом.
4. Виконати порівняльний аналіз відповідних семантичних частин програм, записаних мовою асемблера (лабораторна робота No1.1) та байт-кодом. Результатом виконання даного пункту буде таблиця із порівнянням відповідних частин асемблерного коду та байт-коду.

Завдання за варіантом 23

Задано двовимірний масив (матрицю) цілих чисел A[m][n]. Відсортувати окремо кожен стовпчик масиву алгоритмом №1 методу вставки (з лінійним пошуком зліва) за незменшенням.

1. Лістинг програми мовою Java

```
public static void Algo1(int[][] A) {
    int Elem, j;
    // main loop
    for (int r = 0; r < A.length; r++) {
        // secondary loop
        for (int i = 1; i < A[r].length; i++) {
            Elem = A[i][r];
            j = 0;
            // third loop
            while (Elem > A[j][r])
                j = j + 1;
            // fourth loop
            for (int k = i - 1; k >= j; k--)
                A[k + 1][r] = A[k][r];
            A[j][r] = Elem;
        }
    }
}
```

2. Лістинг програми байт-кодом Java з поясненнями

```
public static void Algo1(int[][]);
    Code:
    // method body start
      //main loop start: for (int r = 0; r < A.length; r++)
       0: iconst_0            // load constant 0 to stack
       1: istore_3            // store value to variable 3, int r = 0;
       //main loop exit condition start
       2: iload_3             // load int value of local variable r to stack
       3: aload_0             // load a reference of local variable A to stack
       4: arraylength         // get length of A
       5: if_icmpge     95    // check main loop continuation condition: r <
A.length
                              // and jump to label 95 when it fails
       //main loop exit condition end
       //main loop body start
       //secondary loop start: for (int i = 1; i < A[r].length; i++)
       8: iconst_1            // load constant 1 to stack
       9: istore        4     // store value to variable 4, int i = 1;
       //secondary loop exit condition start
      11: iload         4     // load int value of local variable i to stack
      13: aload_0             // load a reference of local variable A to stack
      14: iload_3             // load int value of local variable r to stack
      15: aaload              // load a refence from array A, specifically A[r] to
stack
      16: arraylength         // get length of A[r]
      17: if_icmpge     89    // check secondary loop continuation condition: i <
A[r].length
                              // and jump to label 89 when it fails
      //secondary loop exit condition end
      //secondary loop body start
      20: aload_0             // load a reference of local variable A to stack
      21: iload         4     // load int value of local variable i to stack
      23: aaload              // load a refence from array A, specifically A[i] to
stack
      24: iload_3             // load int value of local variable r to stack
      25: iaload              // load int from array A, specifically A[i][r];
      26: istore_1            // store value to variable 1, Elem = A[i][r];
      27: iconst_0            // load constant 0 to stack
      28: istore_2            // store value to variable 2, j = 0;
      // third loop start: while (Elem > A[j][r])
      // third loop exit condition start
      29: iload_1             // load int value of local variable Elem to stack
      30: aload_0             // load a reference of local variable A to stack
      31: iload_2             // load int value of local variable j to stack
      32: aaload              // load a refence from array A, specifically A[j] to
stack
      33: iload_3             // load int value of local variable r to stack
      34: iaload              // load int from array A, specifically A[j][r];
      35: if_icmple     45    // check third loop continuation condition Elem >
A[j][r]
                              // and jump to label 45 when it fails
      // third loop exit condition end
      // thrid loop body start
      38: iload_2             // load int value of local variable j to stack
      39: iconst_1            // load constant 1 to stack
      40: iadd                // add int: j + 1
```

```
      41: istore_2            // store value to variable 2, j = j + 1
      // third loop body end
      42: goto          29    // go to loop exit condition
      // third loop end
      // fourth loop start: for (int k = i - 1; k >= j; k--)
      45: iload         4     // load int value of local variable i to stack
      47: iconst_1            // load constant 1 to stack
      48: isub                // substract int: i - 1
      49: istore        5     // store value to variable 5, int k = i - 1;
      // fourth loop exit condition start
      51: iload         5     // load int value of local variable k to stack
      53: iload_2             // load int value of local variable j to stack
      54: if_icmplt     77    // check fourth loop continuation condition: k >= j
                              // and jump to label 77 when it fails
      // fourth loop exit condition end
      // fourth loop body start
      57: aload_0             // load a reference of local variable A to stack
      58: iload         5     // load int value of local variable k to stack
      60: iconst_1            // load constant 1 to stack
      61: iadd                // add int: k + 1
      62: aaload              // load a refence from array A, specifically A[k + 1]
to stack
      63: iload_3             // load int value of local variable r to stack
      64: aload_0             // load a reference of local variable A to stack
      65: iload         5     // load int value of local variable k to stack
      67: aaload              // load a refence from array A, specifically A[k] to
stack
      68: iload_3             // load int value of local variable r to stack
      69: iaload              // load int from array A, specifically A[k][r];
      70: iastore             // store value to array A, specifically A[k + 1][r] =
A[k][r];
      // fourth loop body end
      71: iinc          5, -1 // decrement k: k--
      74: goto          51    // go to loop exit condition
      // fourth loop end
      77: aload_0             // load a reference of local variable A to stack
      78: iload_2             // load int value of local variable j to stack
      79: aaload              // load a refence from array A, specifically A[j] to
stack
      80: iload_3             // load int value of local variable r to stack
      81: iload_1             // load int value of local variable Elem to stack
      82: iastore             // store value to array A, specifically A[j][r] =
Elem;
      // secondary loop body end
      83: iinc          4, 1    // increment i: i++
      86: goto          11      // go to loop exit condition
      // secondary loop end
      //main loop body end
      89: iinc          3, 1    // increment r: r++
      92: goto          2       // go to loop exit condition
      // main loop end
      95: return                // return nothing (method, just void)
    // function body end
```

3. Порівняльний аналіз асемблерного коду і байт-коду Java

№	Код мовою C	Код мовою Java	Асемблерний код	Байт-код Java	Опис
1	`int r = 0;`	`int r = 0;`	`mov DWORD PTR -16[rbp], 0`	`iconst_0` `istore_3`	Визначення змінної r і запис в неї значення 0. Через необхідність роботи з неявними параметрами, що беруться зі стеку, байт-код налічує дві інструкції для аналогічного коду, записаного мовою асемблера

2	`A[i][r]`	`A[i][r]`	`mov eax, DWORD PTR -12[rbp]`
			`cdqe`
			`lea rdx, 0[0+rax*8]`
			`mov rax, QWORD PTR -40[rbp]`
			`add rax, rdx`
			`mov rax, QWORD PTR [rax]`
			`mov edx, DWORD PTR -16[rbp]`
			`movsx rdx, edx`
			`sal rdx, 2`
			`add rax, rdx`
			`mov eax, DWORD PTR [rax]`
				`aload_0`
				`iload 4`
				`aaload`
				`iload_3`
				`iaload`
					Отримання значення з визначеної комірки двовимірного масиву. В асемблерному коді відбувається приведення типів даних, і вирахування адреси, через що кількість команд більша, ніж в байт-коді

3	`r < n`	`r < A.length`	`.L9:`
			`...`
			`mov eax, DWORD PTR -16[rbp]`
			`cmp eax, DWORD PTR -48[rbp]`
			`jl .L9`
				`2: iload_3`
				`aload_0`
				`arraylength`
				`if_icmpge 95`
				`...`
				`95: return`
					Перевірка умови виходу з циклу. Через використання .length замість передачі розміру через параметри методу, потрібний додатковий байт-код у Java

4	`for (int r = 0; r < n; r++)`	`for (int r = 0; r < A.length; r++)`	`<#1>`
			`jmp .L2`
			`.L9:`
			`…`
			`add DWORD PTR -16[rbp], 1`
				`<#1>`
				`<#3> // goto 2`
				`...`
				`iinc 3, 1`
				`goto 2`
					Реалізація циклу з лічильником. Відмінність полягає в розміщенні та принципі роботи виходу з циклу за умовою.

			.L2: <#3>		
5	while (Elem > A[j][r])	while (Elem > A[j][r])	jmp .L4 .L5: … .L4: ... cmp DWORD PTR -4[rbp], eax jg .L5	// third loop exit condition start 29: iload_1 ... if_icmple 45 ... goto 29 45: iload 4	Реалізація циклу з умовою. Різниця полягає знов в розміщенні та принципі роботи виходу з циклу.