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 №2

з дисципліни
«Архітектура комп'ютерів 2. Програмне забезпечення»

Виконав:
Студент групи КВ-82
Іваненко Олександр Андрійович

Перевірив:
Молчанов О. А.

Київ – 2021

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 №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. Виконати порівняльний аналіз відповідних семантичних частин програм, записаних мовою асемблера (лабораторна робота №1) та байт-кодом. Результатом виконання даного пункту буде таблиця із порівнянням відповідних частин асемблерного коду та байт-коду

Завдання за варіантом 12

Задано двовимірний масив (матрицю) цілих чисел A[m,n]. Відсортувати окремо кожен рядок масиву алгоритмом №3 методу вставки (з лінійним пошуком справа з використанням бар'єру) за незбільшенням.

1. Лістинг програми мовою Java

```
private static void sort_mat(int a[][], int m, int n) {
    int l;
    for(int k = 0; k < m; k++){
        for (int i = 2; i < n+1; i++){
            a[k][0] = a[k][i];
            l = i;
            while (a[k][0] < a[k][l-1]){
                a[k][l] = a[k][l - 1];
                l--;
            }
            a[k][l] = a[k][0];
        }
    }
  }
```

2. Лістинг програми мовою байт-коду Java з поясненнями

```
private static void sort_mat(int[][], int, int);
    Code:
//method body start
//main loop start
//main loop init
       0: iconst_0              // load constant 0 to stack
       1: istore        4       // int k = 0;
//main loop condition
       3: iload         4       // load int value of local variable k to stack
       5: iload_1               // load int value of formal variable size to stack
       6: if_icmpge     97      // check main loop condition k < m
                                // and jump to label 97 if fails
//main loop body start
//second loop start
//second loop init
       9: iconst_2              // load constant 2 to stack
      10: istore        5       // int i = 2;
//second loop condition
      12: iload         5       // load int value of local variable I to stack
      14: iload_2               // load int value of formal variable size to stack
      15: iconst_1              // load constant 1 to stack
      16: iadd                  // perform operation add of last to ints in stack
                                // n + 1
      17: if_icmpge     91      // check second loop condition i < n + 1
//second loop body start
      20: aload_0               // load value of formal arr to stack
      21: iload         4       // load int value of local variable k to stack
      23: aaload                // a[k]
      24: iconst_0              // load constant 0 to stack
      25: aload_0               // a[k][0]
      26: iload         4       // load int value of local variable k to stack
      28: aaload                // a[k]
```

```
      29: iload         5       // load int value of local variable i to stack
      31: iaload                // a[k][i]
      32: iastore               // load a[k][0] = a[k][i] to stack
      33: iload         5       // load int value of local variable i to stack
      35: istore_3              //  l = i;
//while loop start
//while loop condition
      36: aload_0               // load value of formal arr to stack
      37: iload         4       // load int value of local variable k to stack
      39: aaload               // a[k]
      40: iconst_0             // load constant 0 to stack
      41: iaload               // a[k][0]
      42: aload_0              // load value of formal arr to stack
      43: iload         4      // load int value of local variable k to stack
      45: aaload               // a[k]
      46: iload_3              // load int value of local variable l to stack
      47: iconst_1             // load constant 1 to stack
      48: isub                 // l - 1
      49: iaload               // load a[k][l-1] to stack
      50: if_icmpge     73     // condition of while loop if false jump 73
//while loop body start
      53: aload_0              // load value of formal arr to stack
      54: iload         4      // load int value of local variable k to stack
      56: aaload               // a[k]
      57: iload_3              // load int value of local variable l to stack
      58: aload_0              // a[k][l]
      59: iload         4      // load int value of local variable k to stack
      61: aaload               // a[k]
      62: iload_3              // load int value of local variable l to stack
      63: iconst_1             // load constant 1 to stack
      64: isub                 // l - 1
      65: iaload               // load a[k][l - 1] to stack
      66: iastore              // load a[k][l] = a[k][l - 1]to stack
```

```
      67: iinc          3, -1  // l--;
//while loop body end
      70: goto          36     // to while body start
//while loop end
      73: aload_0              // load value of formal arr to stack
      74: iload         4      // load int value of local variable k to stack
      76: aaload               // a[k]
      77: iload_3              // load int value of local variable l to stack
      78: aload_0              // load a[k][l] to stack
      79: iload         4      // load int value of local variable k to stack
      81: aaload               // a[k]
      82: iconst_0             // load constant 0 to stack
      83: iaload             // load a[k][0] to stack
      84: iastore            // load a[k][l] = a[k][0] to stack
//second loop body end
//second loop update
      85: iinc          5, 1 // i++;
      88: goto          12   // go to second loop start
//second loop end
//main loop body end
//main loop update
      91: iinc          4, 1 // k++;
      94: goto          3    // go to main loop start
//main loop end
      97: return             // end function
```

#	Код мовою C	Код мовою Java	Асемблерний код	Байт-код Java	Опис
1	`int k = 0;`	`int k = 0;`	`mov DWORD PTR -8[rbp], 0`	`0: iconst_0` `1: istore  4`	визначення змінної k 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	`k < m;`	`k < m;`	`mov eax, DWORD PTR -8[rbp]`
			`cmp eax, DWORD PTR 24[rbp]`
			`jl .L12`
				`3: iload   4`
				`5: iload_1`
				`6: if_icmpge 97`
					перевірка умови виходу з циклу

3	`a[k][0]`	`a[k][0]`	`mov eax, DWORD PTR -8[rbp]`
			`cdqe`
			`sal rax, 4`
			`mov rdx, rax`
			`mov rax, QWORD PTR 16[rbp]`
			`lea rcx, [rdx+rax]`
				`20: aload_0`
				`21: iload   4`
				`23: aaload`
				`24: iconst_0`
				`25: aload_0`
					отримання значення з визначеної комірки масиву (за індексом). В асемблерному коді відбувається приведення типів даних, і вирахування адреси, через що кількість команд більша, ніж в байт-коді

4	`a[k][i]`	`a[k][i]`	`mov eax, DWORD PTR -8[rbp]`
			`cdqe`
			`sal rax, 4`
			`mov rdx, rax`
			`mov rax, QWORD PTR 16[rbp]`
			`add rdx, rax value to RDX`
			`mov eax, DWORD PTR -12[rbp]`
			`cdqe`
			mov eax, DWORD PTR [rcx+rax*4]
				`26: iload     4`
				`28: aaload`
				`29: iload     5`
				`31: iaload`
					отримання значення з визначеної комірки масиву (за індексом). В асемблерному коді відбувається приведення типів даних, і вирахування адреси, через що кількість команд більша, ніж в байт-коді

5	a[k][0] = a[k][i];	a[k][0] = a[k][i];	<#3> <#4> mov DWORD PTR [rdx], eax	<#3> <#4> 32: iastore	операція присвоєння, оскільки два останні елементи вже знаходяться у стеку у байт-коді не потрібно вказувати які елементи потрібно присвоїти, на відміну від асемблеру, де команда mov з операндами
6	l = i;	l = i;	mov eax, DWORD PTR -12[rbp] mov DWORD PTR -4[rbp], eax	33: iload 5 35: istore_3	операція присвоєння, потрібно спочатку покласти до стеку значення комірки і тільки потім istore для комірки l (тобто istore_3)
7	k++;	k++;	add DWORD PTR -8[rbp], 1	91: iinc 4, 1	операція інкременту. В ассемблері ми додаємо до 1, в байт-коді присутня операція iinc, яка виконує схожу дію.
8	for(<init>;<cond>; <update>) ...	for(<init>;<cond>; <update>) ...	<init> L12: jmp .L8 ... <update> L8: <cond> jl .L12 (to start body)	<init> 3: <cond> 6: if_icmpge 97 ... <update> 94: goto 3 97:	реалізація циклу for відрізняється. В асемблері після ініціалізації стрибок на мітку умови циклу. Якщо умова дійсна – стрибок на початок тіла циклу(після ініціалізації). В байт-коді – навпаки, йде ініціалізація і перевірка умови на початку, якщо умова не дійсна – стрибок на мітку після циклу. Після циклу є стрибок на мітку початоку циклу(а саме на умову, після ініціалізації)
9	while(<cond>) ...	while(<cond>) ...	jmp .L9 L10: ... L9: <cond> jl .L10	36: <cond> 50: if_icmpge 73 ... goto 36 73:	Різниця циклу while в асембері та байт-коді схожа з циклом for. У асемблері одразу стрибок на перевірку умови, якщо умова дійсна – стрибок на мітку тіла циклу. У байт-коді навпаки –

					перевірка умови спочатку, якщо вона не дійсна стрибок на мітку після циклу. Остання команада циклу – стрибок на початок циклу(на умову).