CS 1110
Prelim III: Review Session

Info

- My name: Bruno Abrahao
  - We have two other TA's in the room to help you individually
    - Beibei Zhu
    - Suyong Zhao
  - Ask them questions at any time
  - This set of slides will be posted in the course Website!

Review session

- Let's make this interactive
  - Ask questions
  - All questions are smart
  - More fun
- We'll do exercises
  - Have pen and paper ready!

What's in the exam?

- The material of the previous Prelims
- Arrays
- For loops
- While loops
- Algorithms

We will talk about the new stuff today!

What's in the exam?

- The material of the previous Prelims
- Arrays
- For loops
- While loops
- Algorithms

Question 3 (20 points) a) Consider the program segment below. Draw all variables (with their respective values) and objects created by execution of this program segment.

```
int[][] c= new int[3][2];
int[] z= new int[] {3, 2, 1};
String[] s= new String[2];
z= new int[2];
```

(b) Give an expression to reference the second element of z.

(c) What is the result of the expression `s[1].length()` after execution of the code above?

(d) Give the declaration of a single variable v to store the values 1 and "Hi" somewhere at the same time.

Array: object

Can hold a fixed number of values of the **same type**.

The **type** of the array:

- int[]
- String[]
- Integer[]

Basic form of a declaration: **int[] x**

Does not create array, it only declares x. x's initial value is **null**.

x null int[]

Array creation: **new int**[4]

Assignement: **int**[] t= **new int**[4]

t a0 int[]

Elements of array are numbered: 0, 1, 2, …, x.length–1;

	a0
0	0
	0
1	0
2	0
3	

7

Array: length

Array length: an instance field of the array.

This is why we write x.length, not x.length( )

Length field is **final:** cannot be changed.

Length remains the same once the array has been created.

	a0
	length 4
0	5
1	7
2	4
3	-2

The length is not part of the array type.

The type is **int[]**

An array variable can be assigned arrays of different lengths.

```
int[] x;
x= new int[4];
x= new int[32];
```

8

int[] x ;

x null int[]

x= **new int**[4];

Create array object of length 4, store its name in x

x a0 int[]

x[2]= 5;
x[0]= -4;

Assign 5 to array element 2 and -4 to array element 0

x[2] is a reference to element number 2 of array x

int k= 3;
x[k]= 2* x[0];
x[k-1]= 6;

Assign 2*x[0], i.e. -8, to x[3]
Assign 6 to x[2]

Arrays

	a0
0	0
	0
1	0
2	0
3	

	a0
0	-4
	0
1	5
2	0
3	

	a0
0	-4
	0
1	6
2	-8
3	

9

Difference between Vector and array

Declaration: **int[]** a; Vector v;

Elements of a: **int** values Elements of v: any Objects

Creation: a= **new int**[n]; v= **new** Vector();

Array always has n elements Number of elements can change

Reference: a[e] v.get(e)

Change element: a[e]= e1; v.set(e, e1);

Array locations a[0], a[1], … in successive locations in memory. Access takes same time no matter which one you reference.

Elements all the same type (a primitive type or class type)

Can't tell how Vectors are stored in memory. Referencing and changing elements done through method calls

Elements of any Object type (but not a primitive type). Casting may be necessary when an element is retrieved.

Array initializers

Instead of

int[] c= **new** int[5];

c[0]= 5; c[1]= 4; c[2]= 7; c[3]= 6; c[4]= 5;

Use an array initializer:

int[] c= **new int**[] {5, 4, 7, 6, 5};

array initializer: values must have the same type, in this case, **int**. Length of the array is the number of values in the list

a0
5
4
7
6
5

11

Question 2 (10 points). a) Write a single statement that declares and initializes a two-dimensional **int** array b to look like the table below.

1	3	6	10
2	5	9	13
4	8	12	15
7	11	14	16

12

Two-dimensional Array

d		0	1	2	3
	0	5	4	7	3
		4	8	9	7
	1	5	1	2	3
		4	1	2	9
	2	6	7	8	0
	3				
	4				

Type of d is **int[][]**

("**int** array array")

To declare variable d:

int[][] d.

To create a new array and assign it to d:

d= **new int**[5][4];

To reference element at row r column c:

d[r][c]

13

Multi-dimensional arrays initializer

d		0	1	2	3
	0	5	4	7	3
		4	8	9	7
	1	5	1	2	3
		4	1	2	9
	2	6	7	8	0
	3				
	4				

Using an array initializer:

int[][] d= **new int**[][]{ {5,4,7,3}, {4,8,9,7}, {5,1,2,3}, {4,1,2,9}, {6,7,8,0} };

14

Question 3 (20 points) a) Consider the program segment below. Draw all variables (with their respective values) and objects created by execution of this program segment.

```
int[][] c= new int[3][2] ;
int[] z= new int[] {3, 2, 1};
String[] s= new String[2];
z= new int[2];
```

b) Give an expression to reference the second element of z.

c) What is the result of the expression `s[1].length()` after the execution of the code above?

d) Give the declaration of a single variable v to store the values 1 and "Hi" at the same time.

15

Question 2 (10 points). Write a single statement that declares and initializes a two-dimensional **int** array b to look like the table below.

1	3	6	10
2	5	9	13
4	8	12	15
7	11	14	16

16

What's in the exam?

- The material of the previous Prelims
- Arrays
- For loops
- While loops
- Algorithms

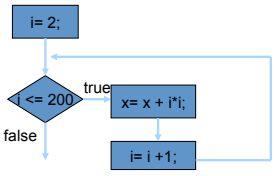
17

Execution of the for-loop

The for-loop:

```
for (int i= 2; i <= 200; i= i +1) {
   x= x + i*i;
}
```

loop counter: i
initialization: **int** i= 2;
loop condition: i <= 200;
increment: i= i + 1
repetend or body: { x= x + i; }



To execute the for-loop.

1. Execute initialization.
2. If loop condition false, terminate execution.
3. Execute repetend.
4. Execute increment, repeat from step 2.

Called a "flow chart"

18

Note on ranges.

2..5 contains 2, 3, 4, 5. It contains 5+1 – 2 = 4 values

2..4 contains 2, 3, 4. It contains 4+1 – 2 = 4 values

2..3 contains 2, 3. It contains 3+1 – 2 = 2 values

2..2 contains 2. It contains 2+1 – 2 = 1 values

The number of values in m..n is n+1 – m.

2..1 contains . It contains 1+1 – 2 = 0 values

3..1 contains . This is an invalid range!

In the notation m..n, we require always, without saying it, that

m <= n + 1 .

If m = n + 1, the range has 0 values.

19

Invariants

- **Assertions**: true-false statements (comments) asserting your beliefs about (the current state of) your program.

  // x is the sum of 1..n <- asserts a specific relationship between x and n

- **Invariant:** an assertion about the variables that is true before and after each iteration (execution of the repetend).

20

Finding an invariant

```
// Store in double variable v the sum
//   1/1 + 1/2 + 1/3 + 1/4 + 1/5 + … + 1/n

 v= 0;
 // invariant: v = sum of 1/i for i in 1..k-1
for (int k= 1; k <= n;  k= k +1) {
        Process k;
}
// v =1/1 + 1/2 + … + 1/n
```

Command to do something and

equivalent postcondition

The increment is executed after the repetend and before the next iteration

What is the invariant?

1 2 3 … k-1 k k+1 … n

21

Spring'06 - Question 3 (20 points). Arrays and loops.
A tridiagonal array m is a square array in which, in each row k (0 ≤ k < m.length), all elements are 0 except perhaps elements m[k][k-1], m[k][k], and m[k][k+1] (if they exist).The following matrix is tridiagonal (* is any integer)

```
* * 0 0 0 0 0
* * * 0 0 0 0
0 * * * 0 0 0
0 0 * * * 0 0
0 0 0 * * * 0
0 0 0 0 * * *
0 0 0 0 0 * *
```

Complete method isTridiagonal, whose specification is given below.

```
/** = "array m is tridiagonal".
Precondition: m is square (number of rows = number of
columns). */
public static boolean isTridiagonal(int[][] m) {
```

22

```
public static boolean isTridiagonal(int[][] m){
  // inv: rows 0..i-1 have tridiagonal property
  for (int i= 0; i != m.length; i= i+1) {

      // return false if row k contains a non-zero
      // where it should have a 0.
      for (int j= 0; j != m.length; j= j+1) {

          if (j != i-1 && j != i && j != i+1 &&
              m[i][j] != 0) {

              return false;
          }
      }
  }
  return true;

}
```

23

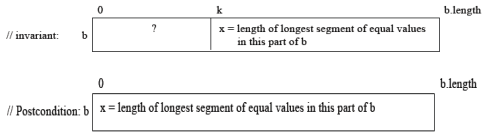
What's in the exam?

- The material of the previous Prelims
- Arrays
- For loops
- While loops
- Algorithms

24

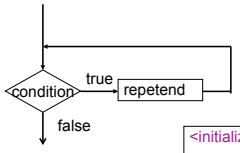
Fall'06, Question 1 (20 points). Array b is in ascending order. Each value may occur many times. Here is an example: {3, 3, 5, 5, 5, 5, 7, 9, 9, 9}. In this example, the length of the longest segment of equal values is 4, since 5 occurs 4 times and the other values occur fewer times.

Write a single **while** loop that stores in x the length of the longest segment of equal values in array b. The post-condition is given below, as is the invariant. **No credit will be given for a loop that does not use this invariant at all**. You may assume that b has at least one element, although it is not necessary



25

Beyond ranges of integers: the while loop



<*condition*>: a boolean expression.

<*repetend*>: sequence of statements.

```
<initialization>;
while (<condition>) {
    <repetend>
        - Process next piece of input;
        - make ready for the next piece of input;
}
```

In comparison to for-loops: we get a broader notion of "there's still stuff to do" (not tied to integer ranges), but *we* must ensure that "condition" becomes false (the statement that makes progress toward termination is in repetend).

26

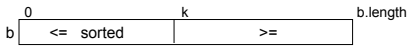
The while loop: 4 loopy questions. Allows us to focus on one thing at a time and thus separate our concerns.

```
// Set c to the number of 'e's in String s.
k= 0; c= 0;

// inv: c = #. of 'e's in s[0..k-1]
while (k < s.length()) {

    if (s.charAt(k) == 'e')
        c= c + 1;
    k= k+ 1;
}
// c = number of 'e's in s[0..n-1]
```

1. How does it start? ((how) does init. make inv true?)
2. When does it stop? (From the invariant and the falsity of loop condition, deduce that result holds.)
3. (How) does it make progress toward termination?
4. How does repetend keep invariant true?

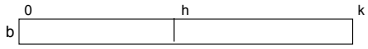
27

Horizontal notation for arrays, strings, Vectors



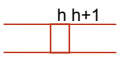
Example of an assertion about an array b. It asserts that:

1. b[0..k–1] is sorted (i.e. its values are in ascending order)
2. Everything in b[0..k–1] is ≤ everything in b[k..b.length–1]



Given the index h of the First element of a segment and the index k of the element that Follows the segment, the number of values in the segment is k – h.

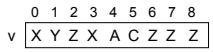
b[h .. k – 1] has k – h elements in it.



(h+1) – h = 1

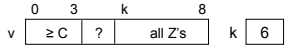
28

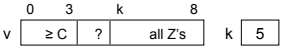
Understanding assertions about lists

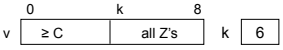


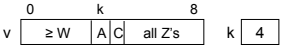
This is a list of Characters

Indicate whether each of these assertions is true or false.





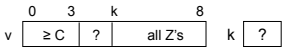




29

Common mistake #1

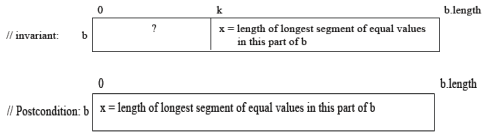
```
// Where is k?
```



30

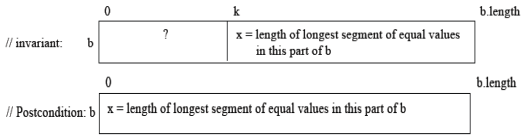
Fall'06, Question 1 (20 points). Array b is in ascending order. Each value may occur many times. Here is an example: {3, 3, 5, 5, 5, 5, 7, 9, 9, 9}. In this example, the length of the longest segment of equal values is 4, since 5 occurs 4 times and the other values occur fewer times.

Write a single loop (either a while-loop or a for-loop) that stores in x the length of the longest segment of equal values in array b. The post-condition is given below, as is the invariant. **No credit will be given for a loop that does not use this invariant at all**. You may assume that b has at least one element, although it is not necessary

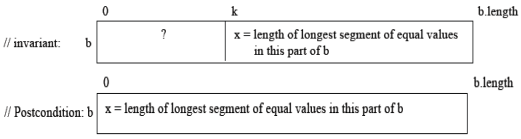


31

1. How should the loop start? (how to make the invariant true?)
2. When can loop stop? (when is the post condition true?)
3. How can repetend make progress toward termination?
4. How does the repetend keep the invariant true?



32




```
x= 0;
k= b.length;
while (0 != k) {
      if (b[k-1] == b[k+x-1]){
                  x= x + 1;
      }
      k= k - 1
}
```

33

Common mistake #2

```
// Returning from nowhere

x= 0;
k= b.length;
while (0 != k) {
      if (b[k-1] == b[k+x-1]){
                  x= x + 1;
      }
      k= k - 1
}
return x;
```

34

Common mistake #3

```
// Bad style: unnecessary variables

x= 0;
max= 0;
k= b.length;
while (0 != k) {
      if (b[k-1] == b[k+max-1]){
                  max= max + 1;
      }
      k= k - 1
      x= max;
}
```

35

What's in the exam?

- The material of the previous Prelims
- Arrays
- For loops
- While loops
- Algorithms

36

Algorithms

- Binary Search
- Dutch National Flag
- Insertion Sort
- Selection Sort
- Partition

37

Algorithms

- Binary Search
- Dutch National Flag
- Insertion Sort
- Selection Sort
- Partition

See Quiz on April 3

38

Common mistake #4

- **Memorizing the algorithm without understanding it**
- **Unable to reproduce the algorithms if there is a small change in the specification**

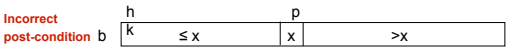
Result is the ability to memorize, not the ability to solve problem!

39

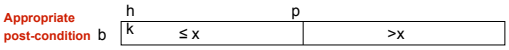
Common mistake #5

Binary Search: Given a sorted (in ascending order) array segment b[h..k-1] and a value x. Store in p an integer that satisfies:
R: b[h..p] <= x < b[p+1..k-1]

Incorrect post-condition



Appropriate post-condition



40

Algorithms

- Binary Search
- Dutch National Flag
- Insertion Sort
- Selection Sort
- Partition

41

Question 4 (20 points). (a) Draw the invariants of the loops that perform Insertion and Selection sort algorithms.

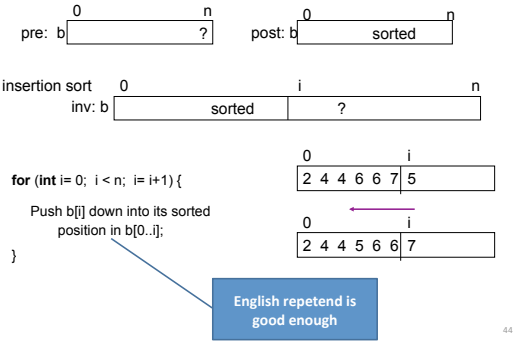
(b) Write the loop for Selection Sort. The repetend should be written in English.

42

Insertion sort

43

Insertion Sort



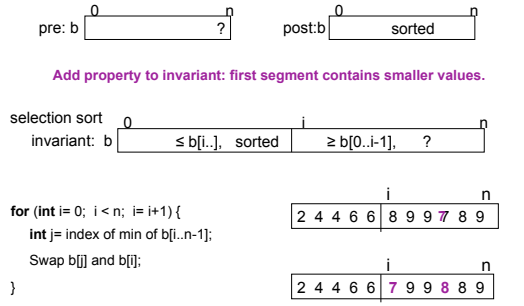
44

Algorithms

- Binary Search
- Dutch National Flag
- Insertion Sort
- Selection Sort
- Partition

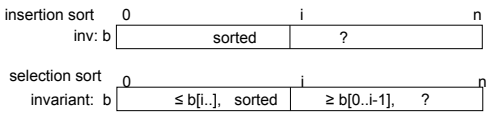
45

Selection Sort



46

Question 4 (20 points). a) Draw the invariants for the loops in the Insertion and Selection sort algorithms.



b) Write the loop for Selection Sort. The repetend should be written in English.

```
for (int i= 0; i < n; i= i+1) {
    int j= index of min of b[i..n-1];
    Swap b[j] and b[i];

}
```

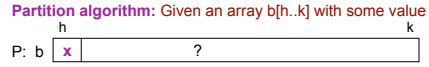
47

Algorithms

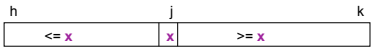
- Binary Search
- Dutch National Flag
- Insertion Sort
- Selection Sort
- Partition

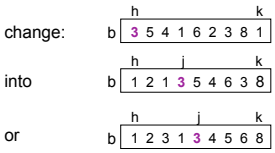
48

Partition algorithm: Given an array b[h..k] with some value x in b[h]:



Swap elements of b[h..k] and store in j to truthify P:





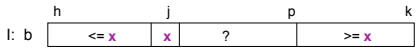
Algorithm is an important piece of the most famous sorting algorithm, quicksort

x is called the pivot value.

x is not a program variable; x just denotes the value initially in b[h].

Partition algorithm: Given an array b[h..k] with some value x in b[h]:

Invariant




```
j= h;   p= k;
while (j < p) {
        if ( b[j+1] <= b[j]){
                swap b[j] and b[j+1];
                j= j+1;
        } else {
                swap b[j+1] and b[p];
                p= p-1
        }
}
```


Summary

- Today we discussed
  - Arrays
  - For loops
  - While loops
  - Algorithms