**A+ Computer Science  
Stacks M/C Test**

**Directions ::** On your answer sheet, mark the letter of the best answer to each question.

1. **Which of the following adds and removes items according to LIFO rules?**

|  |  |
| --- | --- |
| a. | stack |
| b. | queue |
| c. | priority queue |
| d. | binary tree |
| e. | array |

2. **What is output by the code below?**

Stack<Integer> a = new Stack<Integer>();

Queue<Integer> b;

b = new LinkedList<Integer>();

a.push(5);

a.push(7);

out.println(a); //line 1

|  |  |
| --- | --- |
| a. | [5] |
| b. | [7] |
| c. | [5,7] |
| d. | [7,5] |
| e. | [] |

3. **What is output by the code below?**

Stack<Integer> a = new Stack<Integer>();

Queue<Integer> b;

b = new LinkedList<Integer>();

a.push(5);

a.push(7);

b.add(a.pop());

b.add(a.pop());

out.println(b);

|  |  |
| --- | --- |
| a. | [5] |
| b. | [7] |
| c. | [5,7] |
| d. | [7,5] |
| e. | [] |

4. **What is output by the code below?**

Stack<Integer> f = new Stack<Integer>();

f.push(34);

f.push(76);

f.push(84);

out.println(f.pop());

|  |  |
| --- | --- |
| a. | 34 |
| b. | 76 |
| c. | 84 |
| d. | null |
| e. | 50 |

5. **What is ouptut by the code below?**

Stack<Integer> g;

g = new Stack<Integer>();

g.push(12);

g.pop();

g.push(36);

out.println(g);

|  |  |
| --- | --- |
| a. | [12,36] |
| b. | [36,12] |
| c. | [12] |
| d. | [36] |
| e. | [null] |

6. **What is output by the code below?**

Stack<Integer> h;

h = new Stack<Integer>();

h.push(12);

h.pop();

h.push(5);

h.pop();

h.push(7);

h.pop();

h.push(9);

out.println(h.pop());

|  |  |
| --- | --- |
| a. | 12 |
| b. | 5 |
| c. | 7 |
| d. | 9 |
| e. | 0 |

7. **What is output by the code below?**

Stack<Integer> j;

j = new Stack<Integer>();

j.push(34);

j.peek();

j.push(45);

out.print(j.pop()+" ");

j.push(87);

out.print(j.pop());

|  |  |
| --- | --- |
| a. | 45 87 |
| b. | 45 34 |
| c. | 45 |
| d. | 34 45 |
| e. | null |

8. **What is output by the code below?**

Stack<Double> k = new Stack<Double>();

k.push(3.4);

k.push(1.6);

out.println(k.pop());

|  |  |
| --- | --- |
| a. | 3.4 |
| b. | 1.6 |
| c. | 8.2 |
| d. | 5.2 |
| e. | 0.0 |

9. **What is output by the code below?**

Stack<Double> k = new Stack<Double>();

k.push(3.4);

k.push(1.6);

k.pop();

k.push(8.2);

out.println(k.pop());

|  |  |
| --- | --- |
| a. | 3.4 |
| b. | 1.6 |
| c. | 8.2 |
| d. | 5.2 |
| e. | 0.0 |

10. **What is output by the code below?**

Stack<Double> k;

k = new Stack<Double>();

k.push(3.4);

k.push(1.6);

k.pop();

k.push(8.2);

k.pop();

k.push(5.2);

out.println(k.pop());

|  |  |
| --- | --- |
| a. | 3.4 |
| b. | 1.6 |
| c. | 8.2 |
| d. | 5.2 |
| e. | 0.0 |

11. **Which of the following statements would correctly fill <\*1>?**

public class IntStack

{

private int[] stack;

private int top;

public IntStack(){

stack=new int[50];

top=-1;

}

public void push(int x){

stack[++top]=x;

}

public int pop(){

**<\*1>**

}

public int peek(){

**<\*2>**

}

public int topIndex(){

**<\*3>**

}

public boolean isEmpty(){

**<\*4>**

}

}

|  |  |
| --- | --- |
| a. | return top; |
| b. | return stack[top]; |
| c. | return stack[top-1]; |
| d. | return stack[top--]; |
| e. | return stack[stack-1]; |

12. **Assuming <\*1> is filled correctly, which of the following statements would correctly fill <\*2>?**

public class IntStack

{

private int[] stack;

private int top;

public IntStack(){

stack=new int[50];

top=-1;

}

public void push(int x){

stack[++top]=x;

}

public int pop(){

**<\*1>**

}

public int peek(){

**<\*2>**

}

public int topIndex(){

**<\*3>**

}

public boolean isEmpty(){

**<\*4>**

}

}

|  |  |
| --- | --- |
| a. | return top; |
| b. | return stack[top]; |
| c. | return top[top-1]; |
| d. | return stack[top-1]; |
| e. | return stack[top--]; |

13. **Assuming <\*1> and <\*2> are filled correctly, which of the following statements would correctly fill <\*3>?**

public class IntStack

{

private int[] stack;

private int top;

public IntStack(){

stack=new int[50];

top=-1;

}

public void push(int x){

stack[++top]=x;

}

public int pop(){

**<\*1>**

}

public int peek(){

**<\*2>**

}

public int topIndex(){

**<\*3>**

}

public boolean isEmpty(){

**<\*4>**

}

}

|  |  |
| --- | --- |
| a. | return top-stack; |
| b. | return top+1; |
| c. | return top-1; |
| d. | return stack; |
| e. | return top; |

14. **Assuming <\*1>, <\*2> and <\*3> are filled correctly, which of the following statements would correctly fill <\*4>?**

public class IntStack

{

private int[] stack;

private int top;

public IntStack(){

stack=new int[50];

top=-1;

}

public void push(int x){

stack[++top]=x;

}

public int pop(){

**<\*1>**

}

public int peek(){

**<\*2>**

}

public int topIndex(){

**<\*3>**

}

public boolean isEmpty(){

**<\*4>**

}

}

|  |  |
| --- | --- |
| a. | return top==0; |
| b. | return top==-1; |
| c. | return top==50; |
| d. | return top; |
| e. | return top==stack.length; |

15. **Attempting to pop from an empty stack results in?**

|  |  |
| --- | --- |
| a. | type mismatch |
| b. | underflow |
| c. | segmentation fault |
| d. | overflow |

16. **Assuming all the methods in this stack are properly implemented what is the running time of empty?**

public class Stack<E> {

public boolean empty();

public E peek();

public E pop();

public E push(E item);

}

|  |  |
| --- | --- |
| a. | O(1) |
| b. | O(N) |
| c. | O(N^2) |
| d. | O(2^N) |
| e. | O(N!) |

17. **Assuming all the methods in this stack are properly implemented what is the running time of peek?**

public class Stack<E> {

public boolean empty();

public E peek();

public E pop();

public E push(E item);

}

|  |  |
| --- | --- |
| a. | O(1) |
| b. | O(N) |
| c. | O(N^2) |
| d. | O(2^N) |
| e. | O(N!) |

18. **Assuming all the methods in this stack are properly implemented what is the running time of pop?**

public class Stack<E> {

public boolean empty();

public E peek();

public E pop();

public E push(E item);

}

|  |  |
| --- | --- |
| a. | O(1) |
| b. | O(N) |
| c. | O(N^2) |
| d. | O(2^N) |
| e. | O(N!) |

19. **Assuming all the methods in this stack are properly implemented what is the running time of push?**

public class Stack<E> {

public boolean empty();

public E peek();

public E pop();

public E push(E item);

}

|  |  |
| --- | --- |
| a. | O(1) |
| b. | O(N) |
| c. | O(N^2) |
| d. | O(2^N) |
| e. | O(N!) |

20. **Assuming all the methods in this stack are properly implemented what does the empty method do?**

public class Stack<E> {

public boolean empty();

public E peek();

public E pop();

public E push(E item);

}

|  |  |
| --- | --- |
| a. | tests if it’s the stack contains items |
| b. | removes all the items from the stack |
| c. | tests if the stack is empty |
| d. | tests if the stack is at maximum capacity |

21. **Assuming all the methods in this stack are properly implemented what does the peek method do?**

public class Stack<E> {

public boolean empty();

public E peek();

public E pop();

public E push(E item);

}

|  |  |
| --- | --- |
| a. | returns the object at the bottom of the stack without removing it |
| b. | removes and returns the object at the top of the stack |
| c. | returns the object at the top of the stack without removing it |
| d. | returns the object second from the top of the stack without removing it |

22. **Assuming all the methods in this stack are properly implemented what does the pop method do?**

public class Stack<E> {

public boolean empty();

public E peek();

public E pop();

public E push(E item);

}

|  |  |
| --- | --- |
| a. | returns the object at the bottom of the stack without removing it |
| b. | removes and returns the object at the top of the stack |
| c. | removes and returns the object at the bottom of the stack |
| d. | returns the object at the top of the stack without removing it |

23. **Assuming all the methods in this stack are properly implemented what does the push method do?**

public class Stack<E> {

public boolean empty();

public E peek();

public E pop();

public E push(E item);

}

|  |  |
| --- | --- |
| a. | replaces the object at the top of the stack with the new object |
| b. | pushes the top object off of the stack |
| c. | pushes an object onto the bottom of the stack |
| d. | pushes an object onto the top of the stack |

24. **Last-in, first-out is referring to what in a stack?**

|  |  |
| --- | --- |
| a. | the element popped is the one most recently pushed |
| b. | the element popped is the one that has been in the stack the longest |
| c. | the most recently pushed element is at the bottom of the stack |
| d. | the runtime of the push and pop operations |

25. **What is output by the code below?**

public String fun (String str) {

Stack<Character> a = new Stack<Character>();

String res = "";

for (char b: str.toCharArray()) {

a.push(b);

}

while (!a.empty()) {

res += a.pop();

}

return res;

}

**// client code**

out.println(fun("Hello, World!"));

|  |  |
| --- | --- |
| a. | Hello, World! |
| b. | World, Hello! |
| c. | !World ,Hello |
| d. | !dlroW ,olleH |
| e. | !olleH ,dlroW |

26. **What is output by the code below?**

public String fun (String str) {

Stack<Character> a = new Stack<Character>();

String res = "";

for (char b: str.toCharArray()) {

a.push(b);

}

while (!a.empty()) {

res += a.pop();

}

return res;

}

**// client code**

out.println(fun("foobar"));

|  |  |
| --- | --- |
| a. | foobar |
| b. | raboof |
| c. | abfoor |
| d. | roofba |
| e. | barfoo |

27. **What is output by the code below?**

public String fun (String str) {

Stack<Character> a = new Stack<Character>();

String res = "";

for (char b: str.toCharArray()) {

a.push(b);

}

while (!a.empty()) {

res += a.pop();

}

return res;

}

**// client code**

out.println(fun("racecar"));

|  |  |
| --- | --- |
| a. | racecar |
| b. | carrace |
| c. | race |
| d. | aaccerr |
| e. | rreccaa |

28. **What does method fun do?**

public String fun (String str) {

Stack<Character> a = new Stack<Character>();

String res = "";

for (char b: str.toCharArray()) {

a.push(b);

}

while (!a.empty()) {

res += a.pop();

}

return res;

}

|  |  |
| --- | --- |
| a. | Sorts the string |
| b. | Reverses the order of words in a string |
| c. | Reverses the order of the characters in a string |
| d. | Makes a copy of a string |
| e. | Doubles all the characters in a string |

29. **What is output by the code below?**

Stack<Integer> s = new Stack<Integer>();

s.push(70);

s.push(25);

s.push(42);

out.println(s.search(70));

|  |  |
| --- | --- |
| a. | 1 |
| b. | 2 |
| c. | 3 |
| d. | 4 |
| e. | -1 |

30. **What is output by the code below?**

Stack<Integer> s = new Stack<Integer>();

s.push(70);

s.push(25);

s.push(42);

out.println(s.search(42));

|  |  |
| --- | --- |
| a. | 1 |
| b. | 2 |
| c. | 3 |
| d. | 4 |
| e. | -1 |

31. **What is output by the code below?**

Stack<Integer> s = new Stack<Integer>();

s.push(70);

s.push(25);

s.push(42);

out.println(s.search(20));

|  |  |
| --- | --- |
| a. | 1 |
| b. | 2 |
| c. | 3 |
| d. | 4 |
| e. | -1 |

32. **What is output by the code below?**

public static Stack<Integer> fun (Stack<Integer> aplus)

{

Stack<Integer> comp = new Stack<Integer>();

Stack<Integer> ans = new Stack<Integer>();

while (!aplus.empty())

{

int m = aplus.peek();

while (!aplus.empty())

{

if (aplus.peek() < m)

m = aplus.peek();

comp.push(aplus.pop());

}

while (!comp.empty())

{

if (comp.peek() == m)

ans.push(comp.pop());

else

aplus.push(comp.pop());

}

}

return ans;

}

**// client code**

Stack<Integer> s = new Stack<Integer>();

s.push(51);

s.push(35);

s.push(51);

s.push(2);

out.println(fun(s));

|  |  |
| --- | --- |
| a. | [51, 51, 35, 2] |
| b. | [51, 35, 51, 2] |
| c. | [51, 35, 2] |
| d. | [2, 35, 51, 51] |
| e. | [2, 35, 51] |

33. **What is output by the code below?**

public static Stack<Integer> fun (Stack<Integer> aplus)

{

Stack<Integer> comp = new Stack<Integer>();

Stack<Integer> ans = new Stack<Integer>();

while (!aplus.empty())

{

int m = aplus.peek();

while (!aplus.empty())

{

if (aplus.peek() < m)

m = aplus.peek();

comp.push(aplus.pop());

}

while (!comp.empty())

{

if (comp.peek() == m)

ans.push(comp.pop());

else

aplus.push(comp.pop());

}

}

return ans;

}

**// client code**

Stack<Integer> s = new Stack<Integer>();

int[] x = {-6,20,111,-87,300,5};

for( int item : x )

s.add( item );

System.out.println(fun(s));

|  |  |
| --- | --- |
| a. | [-87, -6, 5, 20, 111, 300] |
| b. | [-6, 5, 20, 111, 300, -87] |
| c. | [5, 20, 111, 300, -6, -87] |
| d. | [20, -87, -6, 5, 111, 300] |
| e. | [300, 111, 20, 5, -6, -87] |

34. **What does method fun do?**

public static Stack<Integer> fun (Stack<Integer> aplus)

{

Stack<Integer> comp = new Stack<Integer>();

Stack<Integer> ans = new Stack<Integer>();

while (!aplus.empty())

{

int m = aplus.peek();

while (!aplus.empty())

{

if (aplus.peek() < m)

m = aplus.peek();

comp.push(aplus.pop());

}

while (!comp.empty())

{

if (comp.peek() == m)

ans.push(comp.pop());

else

aplus.push(comp.pop());

}

}

return ans;

}

|  |  |
| --- | --- |
| a. | Removes duplicates from the stack |
| b. | Reverses the stack |
| c. | Sums the values in the stack |
| d. | Sorts the stack in descending order |
| e. | Sorts the stack in ascending order |

35. **What is output by the code below?**

public static int fun (String str)

{

String aplus = "{[()]}";

int res = 0, center = aplus.length() / 2;

Stack<Character> comp = new Stack<Character>();

for (char x: str.toCharArray())

{

if (aplus.contains("" + x))

{

if (aplus.indexOf(x) < 3)

{

comp.push(x);

if (comp.size() > res)

res = comp.size();

}

else

{

int ind = aplus.indexOf(x);

int ref = center - (ind - center + 1);

if (comp.empty() ||

comp.pop() != aplus.charAt(ref))

return -1;

}

}

}

return res;

}

**// client code**

out.println(fun("{()()} (([]{[]})) ()"));

|  |  |
| --- | --- |
| a. | 1 |
| b. | 2 |
| c. | 3 |
| d. | 4 |
| e. | -1 |

36. **What is output by the code below?**

public static int fun (String str)

{

String aplus = "{[()]}";

int res = 0, center = aplus.length() / 2;

Stack<Character> comp = new Stack<Character>();

for (char x: str.toCharArray())

{

if (aplus.contains("" + x))

{

if (aplus.indexOf(x) < 3)

{

comp.push(x);

if (comp.size() > res)

res = comp.size();

}

else

{

int ind = aplus.indexOf(x);

int ref = center - (ind - center + 1);

if (comp.empty() ||

comp.pop() != aplus.charAt(ref))

return -1;

}

}

}

return res;

}

**// client code**

out.println(fun("x = (1 + 2) \* (12 / (4 \* 3))"));

|  |  |
| --- | --- |
| a. | 1 |
| b. | 2 |
| c. | 3 |
| d. | 4 |
| e. | -1 |

37. **What is output by the code below?**

public static int fun (String str)

{

String aplus = "{[()]}";

int res = 0, center = aplus.length() / 2;

Stack<Character> comp = new Stack<Character>();

for (char x: str.toCharArray())

{

if (aplus.contains("" + x))

{

if (aplus.indexOf(x) < 3)

{

comp.push(x);

if (comp.size() > res)

res = comp.size();

}

else

{

int ind = aplus.indexOf(x);

int ref = center - (ind - center + 1);

if (comp.empty() ||

comp.pop() != aplus.charAt(ref))

return -1;

}

}

}

return res;

}

**// client code**

out.println(fun("}}}()()"));

|  |  |
| --- | --- |
| a. | 1 |
| b. | 2 |
| c. | 3 |
| d. | 4 |
| e. | -1 |

38. **What does method fun do?**

public static int fun (String str)

{

String aplus = "{[()]}";

int res = 0, center = aplus.length() / 2;

Stack<Character> comp = new Stack<Character>();

for (char x: str.toCharArray())

{

if (aplus.contains("" + x))

{

if (aplus.indexOf(x) < 3)

{

comp.push(x);

if (comp.size() > res)

res = comp.size();

}

else

{

int ind = aplus.indexOf(x);

int ref = center - (ind - center + 1);

if (comp.empty() ||

comp.pop() != aplus.charAt(ref))

return -1;

}

}

}

return res;

}

|  |  |
| --- | --- |
| a. | return the number of matching pairs in str |
| b. | test if str is balanced |
| c. | return the deepest level of nested pairs if balanced |
| d. | returns the result of a given equation |
| e. | return the count of characters in str that are also in foo |

39. **What is next value to be popped in the stack pseudocode shown below?**

push 4

push 5

push 6

pop X

push 3

push 2

pop X

push 1

push 7

pop X

push 9

pop X

|  |  |
| --- | --- |
| a. | 6 |
| b. | 5 |
| c. | 4 |
| d. | 3 |
| e. | 1 |

40. **What is the sum of all values popped off the stack?**

push 4

push 5

push 6

pop X

push 3

push 2

pop X

push 1

push 7

pop X

push 9

pop X

pop X

|  |  |
| --- | --- |
| a. | 25 |
| b. | 15 |
| c. | 22 |
| d. | 29 |
| e. | 19 |