Name: Chad Huntebrinker

Answer the following Questions

**Q1: [5 points]** Multithreading is supposed to reduce the time of execution. Practically, when multithreading is applied on a small data set, utilizing more threads increase the execution time instead of reducing it, why?

|  |
| --- |
| Use this space to answer the question  Because multithreading works best with a large amount of data. Also, if your multithreading is utilizing a shared variable (which, by default, variables are shared) each thread will have to “wait their turn” to use the variable. This, in turn, will increase the time. In conclusion when multithreading is applied on a small data set, it will increase the execution time because of two reasons: (1) multithreading works best with a large set of data rather than a small set of data and (2) if the variable(s) the different threads are using are shared, it will increase the time because of the need for the different threads to wait until the variable is “open” and ready to use. |

**Q2: [6 points]** write 2 C++ methods to compute the dot product of two vectors. the first method will use a regular iteration process to complete the calculation, the second method will use the parallelism concept using openMP. Compile and run your project and provide the results.

|  |
| --- |
| Use this space to answer the question  //Chad Huntebrinker  //CS 303  //Part 1: regular iteration  #include <iostream>  #include <vector>  using namespace std;  int main() {  vector<int> vector1, vector2;  int total = 0;  //You could change the value and the number of elements in the vector.  //For this example, I have one vector that has the elements (3, 8)  //and the second vector has the elements (9, 13)  vector1.push\_back(3);  vector1.push\_back(8);  vector2.push\_back(9);  vector2.push\_back(13);  //To calculate the dot product, you multiply the two elements of the same spot in the vector by  //each other and add that to the total (the total starts out as 0)  //In this example, the x quantities of the vector (in spot number 0) are multiplied together and then  //the y quantities (in spot number 1) are multiplied together and added to the total.  //3 \* 9 + 13 \* 8 = 131  for (int i = 0; i < vector1.size(); ++i) {  total = vector1.at(i) \* vector2.at(i) + total;  }  cout << "The total of the dot product of the two vectors is: " << total << endl;  return 0;  }  Output: The total of the dot product of the two vectors is: 131  //Chad Huntebrinker  //CS 303  //Part 2: OpenMP  #include <iostream>  #include <vector>  #include "omp.h"  using namespace std;  int main() {  vector<int> vector1, vector2;  int total = 0;  //You could change the value and the number of elements in the vector.  //For this example, I have one vector that has the elements (3, 8)  //and the second vector has the elements (9, 13)  vector1.push\_back(3);  vector1.push\_back(8);  vector2.push\_back(9);  vector2.push\_back(13);  //To calculate the dot product, you multiply the two elements of the same spot in the vector by  //each other and add that to the total (the total starts out as 0)  //In this example, the x quantities of the vector (in spot number 0) are multiplied together and then  //the y quantities (in spot number 1) are multiplied together and added to the total.  //3 \* 9 + 13 \* 8 = 131  omp\_set\_num\_threads(2);  #pragma omp parallel for  for (int i = 0; i < vector1.size(); ++i) {  total = vector1.at(i) \* vector2.at(i) + total;  }  cout << "The total of the dot product of the two vectors is: " << total << endl;  return 0;  }  Output: The total of the dot product of the two vectors is: 131 |

**Q3: [6 points]** Check for balanced parentheses in an expression using **a stack**. Given the expression string below, examine whether the pairs and the orders of “{“,”}”,”(“,”)”,”[“,”]” are correct. Show the stack each time it is modified.

a\*(b-c)/{[3-d]\*(4+9)

|  |
| --- |
| Use this space to answer the question   1. Stack: ( 2. Stack: ( , )   Stack:   1. Stack: { 2. Stack: { , [ 3. Stack: { , [ , ]   Stack: {   1. Stack: { , ( 2. Stack: { , ( , )   Stack: {  In conclusion, the stack’s parentheses is not properly balanced. |

**Q4: [5 points]** Trace the evaluation of the following expression. Show the operand **stack** each time it is modified

20 12 \* 51 + 16 32 45 \* + /

|  |
| --- |
| Use this space to answer the question   1. Stack: 20 2. Stack: 20 , 12 3. Stack: 240 (because 20 \* 12) 4. Stack: 240 , 51 5. Stack: 291 (because 240 + 51) 6. Stack: 291 , 16 7. Stack: 291 , 16 , 32 8. Stack: 291 , 16 , 32 , 45 9. Stack: 291 , 16 , 1440 (because 32 \* 45) 10. Stack: 291, 1456 (because 1440 + 16) 11. Stack: 0.1997 (because 291 / 1456) |

**Q5: [3 points]** by using the STL, complete the following tasks:

1. By using a stack, write a method to convert a decimal number to binary presentation (in a form of string).
2. By using a stack, write a method to calculate the final value of postfix arithmetic expression given in a form of string.
3. By using a stack, write a function to insert an element in the middle of a stack.

You are asked to post the code and provide the results of running your project for each method

|  |
| --- |
| Use this space to answer the question  //Chad Huntebrinker  //CS 303  //Problem #1  #include <iostream>  #include <stack>  #include <string>  using namespace std;  int main() {  int userInput;  stack<char> stack;  char userChoice;  do {  cout << "Enter a integer number to convert to binary: ";  cin >> userInput;    //To change the number to binary, the program divides the inputted number by 2. If there is no remainder,  //then the program puts 0 into the stack. If there is a remainder of 1 (the only possible number the  //remainder can be is 1) then the program puts 1 in the stack.  //example: 8 / 2 = 4 / 2 = 2 / 2 = 1 / 2 = 0  // 8 % 2 = 0 | 4 % 2 = 0 | 2 % 2 = 0 | 1 % 2 = 1  // Stack: 0, 0, 0, 1  do {  if (userInput % 2 == 0) {  stack.push('0');  }  else {  stack.push('1');  }  userInput = userInput / 2;  } while (userInput != 0);  cout << endl << endl;  //The program now takes the 1's and 0's in stack and outputs them the way they are in a stack.  //After they are outputted, they are deleted so that the next one can be outputted.  //Example: 8 = 0, 0, 0, 1  //Output: 1000  do {  cout << stack.top();  stack.pop();  } while (!stack.empty());    cout << endl << endl;  cout << "Enter 'Q' to quit or enter 'C' to continue: ";  cin >> userChoice;  cout << endl << endl;  userChoice = toupper(userChoice);  } while (userChoice != 'Q');  return 0;  }  Input: 114, C, 8, Q  Output: 1110010, 1000  //Chad Huntebrinker  //CS 303  //Problem #2  #include <iostream>  #include <stack>  #include <string>  using namespace std;  //This code takes a string written as a postfix expression and solves it using a stack.  int main() {  string userInput;  stack<int> stack;  int temp1, temp2, i = 0;  char symbol;  cout << "Please enter your postfix expression: ";  cin >> userInput;  do {  //This if statement finds if the character i is currently at is an operation (+, -, \*, or /).  //If it is, then the top number and the number before the top number have that operation  //done to it.  if (userInput[i] == '+' || userInput[i] == '-' || userInput[i] == '\*' || userInput[i] == '/') {  symbol = userInput[i];  temp1 = stack.top();  stack.pop();  temp2 = stack.top();  stack.pop();  if (symbol == '+') {  temp1 = temp2 + temp1;  }  else if (symbol == '-') {  temp1 = temp2 - temp1;  }  else if (symbol == '\*') {  temp1 = temp2 \* temp1;  }  else {  temp1 = temp2 / temp1;  }  stack.push(temp1);  }  //In order to convert the character from the string to an integer value, I subtracted the  //ASCII value of the number from the ASCII value of '0'. This will give you the correct value.  //Example: '4' = 52, '0' = 48: 52 - 48 = 4.  else {  symbol = userInput[i];  temp1 = symbol - '0';  stack.push(temp1);  }  ++i;  //The loop stops once i is the same value of the string the user inputted.  } while (i != userInput.size());  cout << endl << endl;  cout << stack.top();  cout << endl << endl;  stack.pop();  return 0;  }  Input: 47\*  Output: 28  Input: 347\*2/+  Output: 17  //Chad Huntebrinker  //CS 303  //Problem #3  #include <iostream>  #include <stack>  #include <string>  #include <vector>  using namespace std;  int main() {  stack<int> origStack, temp;  int stackPosition, data, i;  char userInput;  vector<int> vector;  do {  i = 0;  //The menu for what the user wants to do with the stack  cout << "If you want to insert integer into a stack regularly, enter R." << endl;  cout << "If you want to insert integer at a certain point, enter P." << endl;  cout << "If you want to quit, enter Q." << endl;  cin >> userInput;  userInput = toupper(userInput);  //If the user entered 'R' or 'P', then the user will enter in what integer number he wants to enter in.  //If the user entered 'P', then the user will enter where he/she wants to put the number in the stack  //starting with 0. If it is 'R', it uses the 'push' command as usual.  if (userInput == 'R' || userInput == 'P') {  cout << "Please enter your integer: ";  cin >> data;  if (userInput == 'P') {  cout << "Please enter the position you want to insert your integer (starts at 0): ";  cin >> stackPosition;  //The program puts all the data before the desired position in a different stack and then  //deletes it from the original stack. If it is at the desired position, the program  //puts the user's number in the different stack at that spot. i is the position the original  //stack is at currently.  do {  if (i == stackPosition) {  temp.push(data);  }  else {  temp.push(origStack.top());  origStack.pop();  }  ++i;  } while (!origStack.empty());  //This loop returns the stacks data to the original position and stack.  do {  origStack.push(temp.top());  temp.pop();  } while (!temp.empty());  }  else {  origStack.push(data);  }  }  //This loop outputs the stack, puts the top element in a vector, and then deletes the top element.  cout << "Stack: ";  do {  cout << origStack.top() << " ";  vector.push\_back(origStack.top());  origStack.pop();  } while (!origStack.empty());  //i is now the largest position  i = vector.size() - 1;    //This loop returns the data from the vector back to the original stack.  do {  origStack.push(vector.at(i));  --i;  } while (i != -1);  vector.clear();  cout << endl << endl;  } while (userInput != 'Q');  return 0;  }  Input: p, 1, p, 2, p, 3, p, 4, r, 8, 3,  Output: Stack: 4 3 2 8 1 |