**DATA STRUCTURES AND ALGORITHM**

Salanatin, Nathaly Pearl F. Fernando Renegado

BSCS-NS-2AB 02/01/2023

Final Exam

|  |  |
| --- | --- |
| 1. Write a C or C++ function differentiate (Poly \*f) where f is a pointer to a polynomial function f(x) | void differentiate (Poly \*f)  {  Poly \*current = f;  Poly \*prev = NULL;    while(current != NULL)  {  if (current->coeff == 1 && prev != NULL)  {  prev->next = NULL;  break;  }  current->coeff = current->coeff \* current->exponent;  current->exponent -= 1;  prev = current;  current = current->next;  }  return;  } |
| 1. Write a C++ ClassMyLinkList with the following attributes and methods. Define themethods. | typedef struct node  {  string name;  struct node \*next;  } NODE;  Class MyLinkList  {  Private:  NODE\* L;  Public:  void makenull()  {  L = NULL  }  void insertSorted(string n)  {  NODE \*trav, \*last, \*new\_node;  last = trav = L;  new\_node = new Node;  new\_node->name = n;  while(last != NULL && last->name > new\_node->name)  {  trav = last;  last = last->next;  }  if (L\_is\_empty())  {  L = new\_node;  }  else  {  trav->next = new\_node;  }  new\_node->next = last;  }  void remove(string n)  {  NODE \*p, \*q;    p = q = L;    while(p != NULL && n != p->name)  {  q = p;  p = p->next;  }  if (p == NULL)  {  cout << "Not FOund";  }  else  {  if(L == p)  {  L = p->next;  }  else  {  q->next = p->next;  }  }  }  void display()  {  NODE \*display\_trav;  display\_trav = L;  while(display\_trav != NULL)  {  cout << display\_trav->name << endl;  display\_trav = display\_trav->next;  }  }  } |
| 1. Write problem #2 using Python class with method definitions. | class Node:  def \_\_init\_\_(self, dataval=None):  self.dataval = dataval  self.nextval = None  class SLinkedList:  def \_\_init\_\_(self):  self.headval = None  def listprint(self):  printval = self.headval  while printval is not None:  print (printval.dataval)  printval = printval.nextval  list = SLinkedList()  list.headval = Node("PEA")  e2 = Node("TSU")  e3 = Node("RYU")  list.headval.nextval = e2  e2.nextval = e3  list.listprint() |
| 1. Write the following recursive expression tree functions. 2. Prefix | void prefix(Node \*node)  {  if (node)  {  cout << node->data << " ";  prefix(node->left);  prefix(node->right);  }  } |
| 1. Infix | void infix(Node \*node)  {  if (node)  {  infix(node->left);  cout << node->data << " ";  infix(node->right);  }  } |
| 1. Postfix | void postfix(Node \*node)  {  if (node)  {  postfix(node->left);  postfix(node->right);  cout << node->data << " ";  }  } |
| 1. Evaluate (Evaluate the expression tree-return the value) | int evaluate(Node \*node)  {  if (!node)  {  return 0;  }  if (isdigit(node->data))  {  return node->data - '0';  }  int left = evaluate(node->left);  int right = evaluate(node->right);  switch (node->data)  {  case '+': return left + right;  case '-': return left - right;  case '\*': return left \* right;  case '/': return left / right;  default: return 0;  }  } |
| 1. Write the following SET ADT functions using the Bit Vector implementation 2. Difference (A,B) -set C contains the results | void difference(const vector<bool>& A, const vector<bool>& B, vector<bool>& C)  {  int size = min(A.size(), B.size());  C.resize(size, false);  for (int i = 0; i < size; i++)  {  C[i] = A[i] && !B[i];  }  } |
| 1. isMember (x, A) –check if x is a member of set A | bool isMember(int x, const vector<bool>& A)  {  if (x < 0 || x >= A.size())  {  return false;  }  return A[x];  } |
| 1. isSubset(A,B) –checks if set A is a subset of set B | if (x in set(M[i],M[j],block))  M[i][j]=z;  if (i!=9 and j!=9)  continue;  else  return (M)}; |
| 1. disjoint (A, B) –checks if sets are disjoint | bool areDisjoint(int set1[], int set2[], int m, int n)  {    for (int i=0; i<m; i++)  for (int j=0; j<n; j++)  if (set1[i] == set2[j])  return false;  return true;  } |
| 1. Explain how the Huffman algorithm works in text file compression and decompression. | * Huffman algorithm works by compressing by creating a tree using frequencies of the character and generate code. After the data is encoded, it will be decoded. Decoding is done using the same tree. * Huffman algorithm creates a Huffman tree based on the frequency of the characters in the text file and gives each character a distinct binary code, allowing for lossless compression and decompression of the text file. |