|  |
| --- |
| //Function finding gcd  int gdc(int a, int b){  int r;    while(a%b!=0){  r=a%b;  a = b;  b = r;  }    return b;  } |
| //ways to return 3 different ans from the function by using ans[3]  /\*\*  \* CS1010 AY2019/20 Semester 1 Lab4 Ex2  \* subsequence.c  \* This program reads in a list of integers and find the  \* k-interval subsequence with the largest sum.  \* <LIU YIFENG>  \* <C06>  \*/  #include <stdio.h>  //function prototype  int scan\_list(int []);  void sum\_subsequence(int [], int, int []);  int main(void) {  int list[10]={0}, size; // array and actual number of elements entered  int answers[3]={0}; // stores the required answers  size = scan\_list(list); //calling function 'scan\_list'  sum\_subsequence(list, size, answers); //calling function 'sum\_subsquence'  printf("Max sum %d of %d-interval subsequence starting at position %d.\n",  answers[0], answers[1], answers[2]);  return 0;  }  // Read in elements for the array arr and returns the  // number of elements read.  // Postcond: i > 0  int scan\_list(int arr[]) {  int num, i;  printf("Enter number of elements: ");  scanf("%d", &num);  printf("Enter %d element%s: ", num, (num>1)?"s":"");  for (i=0; i<num; i++) {  scanf("%d", &arr[i]);  }  return num;  }  // This function read in the array element in K-interval and  // finds the maximum sum  // ans[0] = maximum sum of solution subsequence  // ans[1] = interval k of the solution subsequence  // ans[2] = start position of the solution subsequence  void sum\_subsequence(int arr[], int size, int ans[]) {  int start, j, k, sum;  for(k=1; k<=size; k++){  for(start=0; start<k; start++){  sum=0;  for(j=start; j<size; j+=k){  sum = sum + arr[j];  }  if(sum>ans[0]){  ans[0]=sum;  ans[1]=k;  ans[2]=start;;  }  }  }  } |
|  |
|  |
|  |
|  |
|  |
|  |
| //Example of pointer function involving referencing and de-referencing   |  | | --- | | #include <stdio.h> | |  | #define DIM 12 | |  |  | |  | //Function Prototypes | |  | void scanBoard(int [][DIM]); | |  | int checkDiagonal(int [][DIM],int,int,int); | |  | int checkHorizontal(int [][DIM],int,int,int); | |  | int checkVertical(int [][DIM],int,int,int); | |  | int search(int[][DIM],int\*,int\*,int); | |  |  | |  | int main(void) { | |  | int board[DIM][DIM] = { {0} }; | |  | int search\_digit; | |  | int length = 0; // length of the longest sequence of search digit | |  | int bestRow, bestCol; // where the longest sequence of search digit starts in the board | |  | scanBoard(board); | |  | scanf("%d",&search\_digit); | |  | length = search(board,&bestRow,&bestCol,search\_digit); | |  | if (length > 0) { | |  | printf("Length of longest sequence = %d\n", length); | |  | printf("Start at (%d,%d)\n", bestRow, bestCol); | |  | } | |  | else | |  | printf("No such sequence.\n"); | |  |  | |  | return 0; | |  | } | |  |  | |  | /\*This function reads in the input for the board. | |  | \* Precond: Positive integers in the range of [1-9]. | |  | \*/ | |  | void scanBoard(int arr[][DIM]) { | |  | int row, col; | |  | for (row=0; row<DIM; row++){ | |  | for (col=0; col<DIM; col++) { | |  | scanf("%d", &arr[row][col]); | |  | } | |  | } | |  | } | |  |  | |  | /\*This function searches the board for the longest sequence of the search digit. | |  | \* Input: Board's array and search digit. | |  | \* Output: Starting postion of longest sequence and length. | |  | \*/ | |  | int search(int arr[][DIM],int \*bestRow, int \*bestCol,int search\_digit) { | |  | int row,col,counter = 0,length = 0 ; | |  | for(row = 0; row < DIM; row++){ | |  | for(col = 0; col < DIM; col++){ | |  | if(arr[row][col] == search\_digit){ //if found a search digit, check vertically. | |  | counter = checkVertical(arr,row,col,search\_digit); | |  | if(counter > length){ //If the found sequence is longer then current length, swap length and update position. | |  | length = counter; | |  | \*bestRow = row; | |  | \*bestCol = col; | |  | } | |  | counter = checkHorizontal(arr,row,col,search\_digit); //if found a search digit, check horizontally. | |  | if(counter > length){ //If the found sequence is longer then current length, swap length and update position. | |  | length = counter; | |  | \*bestRow = row; | |  | \*bestCol = col; | |  | } | |  | counter = checkDiagonal(arr,row,col,search\_digit); //if found a search digit, check diagonally. | |  | if(counter > length){ //If the found sequence is longer then current length, swap length and update position. | |  | length = counter; | |  | \*bestRow = row; | |  | \*bestCol = col; | |  | } | |  | } | |  | } | |  | } | |  | return length; | |  | } | |  |  | |  | /\*This functions checks vertically if the number below contains the search digit and counts how long it is. | |  | \* Input: Board's array, current position and search digit. | |  | \* Output: Return length of the consecutive search digit. | |  | \*/ | |  | int checkVertical(int arr[][DIM],int row, int col,int search\_digit){ | |  | int counter = 0; | |  | while(arr[row][col] == search\_digit){ //If it equals search digit, continue searching in that direction | |  | row++; | |  | counter++; | |  | } | |  | return counter; | |  | } | |  |  | |  | /\*This functions checks diagonally if the number diagonally contains the search digit and counts how long it is. | |  | \* Input: Board's array, current position and search digit. | |  | \* Output: Return length of the consecutive search digit. | |  | \*/ | |  | int checkDiagonal(int arr[][DIM],int row, int col,int search\_digit){ | |  | int counter = 0; | |  | while(arr[row][col] == search\_digit){ //If it equals search digit, continue searching in that direction | |  | row++; | |  | col++; | |  | counter++; | |  | } | |  | return counter; | |  | } | |  |  | |  | /\*This functions checks horizontally if the number beside it contains the search digit and counts how long it is. | |  | \* Input: Board's array, current position and search digit. | |  | \* Output: Return length of the consecutive search digit. | |  | \*/ | |  | int checkHorizontal(int arr[][DIM],int row, int col,int search\_digit){ | |  | int counter = 0; | |  | while(arr[row][col] == search\_digit){ //If it equals search digit, continue searching in that direction | |  | col++; | |  | counter++; | |  | } | |  | return counter; | |  | } | |
| //Example of a hash-map method   |  | | --- | | #include <stdio.h> | |  | #define MAXSIZE 9 | |  | //Functions Prototype | |  | int scanSquare(int [][MAXSIZE]); | |  | int isSemiMagic(int[][MAXSIZE], int); | |  |  | |  | int main(void) { | |  | int size, square[MAXSIZE][MAXSIZE]; | |  | size = scanSquare(square); | |  | if(isSemiMagic(square,size)) | |  | printf("It is a semi-magic square.\n"); | |  | else | |  | printf("It is not a semi-magic square.\n"); | |  |  | |  | return 0; | |  | } | |  | /\*This function checks if it is a semi-magic square by: | |  | \* 1)all rows and column sum are the same | |  | \* 2)All integers from 1 to size\*size is included. | |  | \* input:2D array of sqaure and size | |  | \* output:1 for semi magic square and 0 for not semi magic square. | |  | \*/ | |  | int isSemiMagic(int arr[][MAXSIZE], int size){ | |  | int i,j, firstSum = 0, rowSum,colSum, num[MAXSIZE\*MAXSIZE] ={0}; | |  | for(i = 0; i < size; i++){ //Compute the sum of a row for comparison | |  | firstSum += arr[0][i]; | |  | } | |  | for(i = 0; i<size; i++){ //Loop through each row/column and compute sum | |  | rowSum = 0; | |  | colSum = 0; | |  | for(j = 0; j <size; j++){ | |  | rowSum += arr[i][j]; //i and j are flipped for row and column respectively | |  | colSum += arr[j][i]; | |  | num[arr[i][j]-1] = 1; //Assign 1 to array if integer is present, checking if 1 to n^2 integer are included | |  | } | |  | if(rowSum != firstSum || colSum != firstSum) //If any of the sum is not equals to first row return 0 | |  | return 0; | |  | } | |  | for(i = 0; i < size\*size; i++){ //Loop through stored array of included integer. If not present return 0. | |  |  | |  | if(num[i] == 0) | |  | return 0; | |  | } | |  | return 1; | |  | } | |  |  | |  | /\*This function reads in the input for the magic square. | |  | \* Precond: Positive integers | |  | \* output: Returns size of square. | |  | \*/ | |  | int scanSquare(int arr[][MAXSIZE]) { | |  | int i,j,size; | |  | printf("Enter size of square: "); | |  | scanf("%d",&size); | |  | printf("Enter values in the square: \n"); | |  | for(i = 0; i < size; i++){ | |  | for(j = 0; j< size; j++){ | |  | scanf("%d",&arr[i][j]); | |  | } | |  | } | |  | return size; | |  | } | |
|  |
|  |
| /\*\*  \*S1010 AY2019/20 Semester 1 Lab5 Ex1  \*Square.c  \*Given the size of a square and the value in  \*the square, this program determines whether  \*the given square is a semi-magic square.  \*Name: LIU YIFENG  \*Group C06  \*/  #include <stdio.h>  #define MAXSIZE 9  //function prototype  int scanSquare(int [][MAXSIZE]);  int isSemiMagic(int [][MAXSIZE], int);  int main(void) {  int square[9][9];//declare int array  int size;//declare variable  size = scanSquare(square);//calling function scanSquare    //callin function isSemiMagic  if(isSemiMagic(square, size)){  printf("It is a semi-magic square.\n");  }  else{  printf("It is not a semi-magic square.\n");  }  return 0;  }  // Read in size of square and values in the square.  // Return the size of square.  int scanSquare(int square[][MAXSIZE]) {  int r, c, size;  printf("Enter size of square: ");  scanf("%d", &size);  printf("Enter values in the square: \n");  for (r=0; r<size; r++) {  for (c=0; c<size; c++) {  scanf("%d", &square[r][c]);  }  }  return size;  }  //Check if a array is SemiMagic  int isSemiMagic(int arr[][MAXSIZE], int count){  int i, j, colSum, rowSum, sumCheck=0, firstSum=0, fixSum=0; //declare variables  sumCheck = ((count\*count)\*((count\*count)+1))/2;//finding the total sum of 1-n\*n    //finding the total sum of the array elements  for(i=0; i<count; i++){  for(j=0; j<count; j++){  fixSum += arr[i][j];  }  }  //setting a sum to check against sums for all rows and columns  for(i=0; i<count; i++){  firstSum += arr[i][0];  }  //if the sum of all elements in the array is not equal to the total sum of 1-n\*n  //the array is not SemiMagic  if(fixSum != sumCheck){  return 0;  }  //find all the rowSum and colSum and check against with the reference sum  for(i=0; i<count; i++){  rowSum=0;  colSum=0;  for(j=0; j<count; j++){  rowSum += arr[i][j];  colSum += arr[j][i];  }  if(rowSum!=firstSum || colSum!=firstSum){  return 0;  }  }  return 1;  } |
| /\*  \* CS1010 AY2019/20 Semester 1 Lab4 Ex3  \* Frog.c  \* This program allows the user to play a game by moving frogs from the left to the right and vice versa  \* <LIU YIFENG>  \* <C06>  \*/  #include <stdio.h>  //Function Prototypes  int jump(int, int, int[]);  void printBridge(int, int[]);  int checkGameState(int, int[]);  int main(void) {  int bridge[1000];  int i, len\_Bridge, position, validity;  int numMoves = 0;  int gameState = 1;  printf("Please enter the length of the bridge: ");  scanf("%d", &len\_Bridge);  for (i=0; i<len\_Bridge; i++) {  int frogType;  printf("Please insert frogs at position %d: ", i);  scanf("%d", &frogType);  bridge[i] = frogType;  }  printf("Input completed\n");  printBridge(len\_Bridge, bridge);  printf("Please start moving the frogs\n");  while(gameState == 1){ //Valid moves are still left to be made  printf("Move the frog at position: ");  scanf("%d",&position);  validity = jump(position,len\_Bridge,bridge);  while(validity == 0){  printf("Invalid move!\n");  printf("Move the frog at position: ");  scanf("%d",&position);  validity = jump(position,len\_Bridge,bridge);  }  printBridge(len\_Bridge, bridge);  numMoves++;  gameState = checkGameState(len\_Bridge,bridge);  }  if(gameState == 0) //No valid Moves  printf("Oh no! It seems like the two clans of frogs are stuck!");  else if(gameState == 2) //Game finished  printf("Congratulations! The frogs took %d jumps to cross the bridge!\n",numMoves);  /\*\* Insert your code below \*\*/  return 0;  }  //Prints out the current state on the bridge  void printBridge(int size, int bridge[]) {  int i;  printf("Position: ");  for (i=0; i<size; i++) {  printf("%2d ", i);  }  printf("\n");  printf("Bridge: ");  for (i=0; i<size; i++) {  printf("%2d ", bridge[i]);  }  printf("\n");  }  /\* This function "jumps" the frog based on its position and returns 1 if valid.  \* Returns 0 if no valid move.  \*/  int jump(int position, int size, int bridge[]){  int validMove = 0;  switch(bridge[position]){  case 1:  if(bridge[position+1] == 0 ){ //if it is empty infront of frog, move forward.  bridge[position+1] = 1;  bridge[position] = 0; //previous position becomes 0.  validMove++;  }  else if(bridge[position+2] == 0 ){ //if it is empty infront of frog, move forward.  bridge[position+2] = 1;  bridge[position] = 0; //previous position becomes 0.  validMove++;  }  else  validMove = 0; //not able to move  break;  case -1:  if(bridge[position-1] == 0 ){ //if it is empty infront of frog, move forward.  bridge[position-1] = -1;  bridge[position] = 0; //previous position becomes 0.  validMove++;  }  else if(bridge[position-2] == 0 ){ //if it is empty infront of frog, move forward.  bridge[position-2] = -1;  bridge[position] = 0; //previous position becomes 0.)  validMove++;  }  else  validMove = 0; //not able to move  break;  case 0:  validMove = 0; //blank space  }  return validMove;  }  /\*  \* This function checks the following conditions:  \* 1)There is valid moves left to be made. Returns 1.  \* 2)There is no more valid moves to be made. Returns 0.  \* 3)End of game state. Returns 2.  \*/  int checkGameState(int size, int bridge[]){  int i,j,k, flag = 0;  for(i = 0; i < size; i++){  if(bridge[i] == 1){ //If position holds value "1" frog, and there is still a valid move for it, return 1.  if(bridge[i+1] == 0 || bridge[i+2] == 0 ){  flag = 1;  return flag;  }  }  else if(bridge[i] == -1){ //If position holds value "-1" frog, and there is still a valid move for it, return 1.  if(bridge[i - 1] == 0 || bridge[i - 2] == 0) {  flag = 1;  return flag;  }  }  else if(bridge[i] == 0){ //If position is 0 then to check end game state  for(j = i+1; j < size; j++){  if(bridge[j] == -1)  break;  else if(bridge[j] == 1 && j == size-1){  for(k = i-1; j > 0;k--){  if(bridge[k] == 1)  break;  else{  flag =2; //the game has been completed.  return flag;  }  }  }  }  }  }  flag = 0; //no possible move  printf("Flag value: %d\n",flag);  return flag;  } |
| /\*  \* CS1010 AY2019/20 Semester 1 Lab4 Ex1  \* nanotable1.c  \* This program takes in student ID and score to store as a table  \* LIU YIFENG  \* C06  \*/  #include <stdio.h>  #include <string.h>  #include <math.h>  #define TRUE 1  #define MAX\_COMMAND\_LENGTH 100  /\* Function prototypes \*/  void print\_help();  void insert\_table(int[][100], int);  int sum(int[][100]);  int ave(int[][100], int);  void rank(int[][100], int);  int main(void) {  char input[MAX\_COMMAND\_LENGTH];  int table[2][100]={}; //initialising 2D array  int i, j, average, total=0, count=0; //declaring variables  // print welcome information  printf("Welcome to Nanotable!\n");  printf("Type \"help\" for more information...\n");  while (TRUE) {  printf("Waiting for command...\n");  scanf("%s", input);  total = sum(table); //calling function 'sum'  if (strcmp(input,"help") == 0) {  print\_help();  } else if (strcmp(input,"exit") == 0){  printf("See you!\n");  break;  } else if (strcmp(input,"sum") == 0) {    if (total==0){ //determine whether the table is empty  printf("The table is empty\n");  } else {  printf("The sum of score is %d\n", total);  }  } else if (strcmp(input,"ave") == 0) {  if (total==0){ //determine whether the table is empty  printf("The table is empty\n");  } else {  average = ave(table, count); //calling function 'ave'  printf("The average of score is %d\n", average);  }  }  else if (strcmp(input,"insert") == 0) {  insert\_table(table, count); //calling function 'insert\_table'  count++;  } else if (strcmp(input,"init") == 0) { //to initialise the table  for(i=0; i<2; i++){  for(j=0; j<100; j++){  table[i][j]=0;  }  }  count=0;  printf("Initializing table...\n");  } else if (strcmp(input,"rank") == 0) {  if (total==0) {  printf("The table is empty\n");  } else {  rank(table,count); //calling funtion 'rank'  }  } else {  printf("No such command: %s, please input command again!\n", input);  }  }  return 0;  }  // This function prints the help information  void print\_help() {  printf("Commands available: insert, init, sum, ave, rank, help, exit\n");  }  //This function scans for student ID and Score to store in a 2D array  void insert\_table(int arr[][100], int i){  printf("Please input the student's ID...\n");  scanf("%d", &arr[0][i]);  printf("Please input the student's score...\n");  scanf("%d", &arr[1][i]);  }  //This function calculates the sum of all scores in the 2D array  int sum(int arr[][100]){  int sum=0;  int i;  for(i=0; i<100; i++){  sum += arr[1][i];  }  return sum;  }  //This funcion calculates the average of all scores in the 2D array  int ave(int arr[][100], int count){  int sum=0;  int i;  double average=0;  for(i=0; i<count; i++){  sum += arr[1][i];  }  average = (double)sum / count;  return round(average);  }  //This function ranks the ID and Scores accoringly  void rank(int arr[][100], int count){  int num, i, j, temp1, temp2, temp3;  printf("Please input the rank (1 - %d)...\n", count);  scanf("%d", &num);  while(num>count){  printf("Invalid rank: %d\n", num);  printf("Please input the rank (1 - %d)...\n", count);  scanf("%d", &num);  }  //to sort all data in ascending order and make sure ID and Score are paired  for(i=0; i<count; i++){  for(j=i+1; j<count; j++){  if(arr[1][i]>arr[1][j]){  temp1=arr[0][i];  temp2=arr[1][i];  arr[0][i]=arr[0][j];  arr[1][i]=arr[1][j];  arr[0][j]=temp1;  arr[1][j]=temp2;  }  //sort by ID if the scores are found to be the same  if(arr[1][i]==arr[1][j]){  if(arr[0][i]>arr[0][j]){  temp3=arr[0][i];  arr[0][i]=arr[0][j];  arr[0][j]=temp3;  }  }  }  }  printf("ID: %d, Score: %d\n", arr[0][num-1], arr[1][num-1]);  } |
| /\*CS1010 AY2019/20 Semester 1 Lab3 Ex2  \* square\_free.c  \* This program reads 4 integers in the following sequence:  \* lower1, upper1, lower2, upper2, compute the number of square-free  \* integers in two ranges [lower1, upper1] (both inclusive) and  \* [lower2, upper2] (both inclusive), compare and report which  \* range has more square-free integers.  \* LIU YIFENG  \* Group C06  \*/  #include <stdio.h>  #include <math.h>  //function prototyping  int compute\_squareFreeNum(int, int);  int main(void) {  //declear variables  int lower1, upper1, lower2, upper2, range1, range2;  printf("Enter four positive integers: ");  scanf("%d%d%d%d", &lower1, &upper1, &lower2, &upper2);  //calling function twice  range1 = compute\_squareFreeNum(lower1, upper1);  range2 = compute\_squareFreeNum(lower2, upper2);  //if condition statements to compare the number of square number for 2 sets of numbers  if(range1==range2){  printf("Both ranges have the same number of square-free numbers: %d\n", range1);  }  else if(range1<range2){  printf("Range [%d, %d] has more square-free numbers: %d\n", lower2, upper2, range2);  }  else if(range1>range2){  printf("Range [%d, %d] has more square-free numbers: %d\n", lower1, upper1, range1);  }  return 0;  }  // this function takes 2 integers and cumpute the number of square-free integers  int compute\_squareFreeNum(int a, int b){  //declear variables  int squareNum=0, squareFreeNum, num, i, n;  //condition for 'for loop' and initialising n = a  for(n=a; n<=b; n++){  //condition for 'for loop' and initialising i = 2  for(i=2; n/(pow(i,2))>=1; i++){  //condition for if statement and cast typing the pow to int type  if(n % ((int)(pow(i,2))) == 0){  squareNum ++;  break;  }  }  }  num = b - a + 1;  squareFreeNum = num - squareNum;  return squareFreeNum;  } |
| /\*CS1010 AY2019/20 Semester 1 Lab3 Ex1  \* candles.c  \* This program calculates the total number of candles burnt  \* by asking user to enter the initial number of candles and  \* number of residual wax needed to form a new candle.  \* Name: LIU YIFENG  \* Group: C06  \*/  #include <stdio.h>  //function prototype  int count\_candles(int, int);  int main(){  //declaring variables  int candleNum, residualNum, candlesBurn;  printf("Enter number of candles and \n");  printf("number of residuals to make a new candle: ");    scanf("%d%d", &candleNum, &residualNum);  //calling function "count\_candles"  candlesBurn = count\_candles(candleNum, residualNum);    printf("Total candles burnt = %d\n", candlesBurn);  return 0;  }  /\*  \*This function computes the number of candles burnt  \*from a given number of total candles and the number of  \*residual candles need to form a new candle  \*pre-condition for function, b>1  \*/  int count\_candles(int a, int b){    int reminder, set, sum; //declear variable  sum = a;  //conditions for 'for loop'  for(a=a; a/b != 0; a = set+reminder){  set = a/b;  reminder = a%b;  sum += set;  }  return sum;  } |
| /\*CS1010 AY2019/20 Semester 1 Lab3 Ex3  \* bisection.c  \* This program computes the root of a continuous  \* function by using the bisection method.  \* Name: LIU YIFENG  \* Group: C06  \*/  #include <stdio.h>  //function prototype  double polynomial(double, int, int, int, int);  int main(void) {  //declaring variables and contants  int c3, c2, c1, c0; // coefficients of polynomial  double a, b, // endpoints  pA, pB; // function values at endpoints  double m, pM; // midpoint and function value at midpoint  const double threshold = 0.0001; //constant variable for threshold  printf("Enter coefficients (c3,c2,c1,c0) of polynomial: ");  scanf("%d %d %d %d", &c3, &c2, &c1, &c0);    printf("Enter endpoints a and b: ");  scanf("%lf %lf", &a, &b);  //condition for while loop where the difference between a & b has to be larger than threshold  while ((b-a)>threshold){  m = (a+b)/2;  pA = polynomial(a, c3, c2, c1, c0); //calling function of p(a)  pB = polynomial(b, c3, c2, c1, c0); //calling function of p(b)  pM = polynomial(m, c3, c2, c1, c0); //calling function of p(m)  //when pM is zero, the vaule of m will be the root and while loop breaks  if(pM==0){  break;  }  else if(pA\*pM >0){  a=m;  }  else if(pB\*pM>0){  b=m;  }  }  m = (a+b)/2;  pM = polynomial (m, c3, c2, c1, c0); //calling function of p(m) which is p(root)  printf("root = %0.6lf\n", m);  printf("p(root) = %0.6lf\n", pM);  return 0;  }  //function polynomial which computes and returns a function value  double polynomial(double x, int a, int b, int c, int d){  return a\*x\*x\*x+b\*x\*x+c\*x+d;  } |