|  |
| --- |
| Title of Article [Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here.] |



**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <windows.h>

#include<stdbool.h>

#define sleep(x)Sleep(x\*1000)

#define SIZE 5

#define MAX\_COURSES 5

char queue[SIZE][256];

int front = -1, rear = -1;

//0 1 knapsack algo START

int \_max(int a, int b){

    if(a>=b)

        return a;

    else

        return b;

}

void knapsack01(int n, int C, int \*w, int \*v){

    int p[n+1][C+1], o[n];

    int k=0;

    for(int i=0; i<=n; i++){

        for(int j=0; j<=C; j++){

            if(i==0 || j==0){

                p[i][j]=0;

            }

            else if(j-w[i-1]>=0){

                p[i][j]=\_max(p[i-1][j], v[i-1]+p[i-1][j-w[i-1]]);

            }

            else if(j-w[i-1]<0){

                p[i][j]=\_max(p[i-1][j],p[i][j-1]);

            }

        }

    }

    printf("Max credit: %d\n", p[n][C]);

    for(int i=n, j=C; i>0; i--){

        if(p[i][j]!=p[i-1][j]){

            o[k]=i;

            j-=w[i-1];

            k++;

        }

    }

    printf("Courses included: ");

    for(int i=k-1; i>=0; i--)

        printf("%d ", o[i]);

}

//0 1 knapsack algo STOP

//Activity selection START

void activity\_selection(int n, int a[n][2]){

    int x;

    for(int i=0; i<n-1; i++){

        for(int j=0; j<n-i-1; j++){

            if(a[j][1]>a[j+1][1]){

                int tmp = a[j][1];

                a[j][1] = a[j+1][1];

                a[j+1][1] = tmp;

                tmp = a[j][0];

                a[j][0] = a[j+1][0];

                a[j+1][0] = tmp;

            }

        }

    }

    printf("Following activities are selected:\n");

    printf("(%d, %d)\n", a[0][0], a[0][1]);

    x=a[0][1];

    for(int i=1; i<n; i++){

        if(a[i][0]>x){

            x=a[i][1];

            printf("(%d, %d)", a[i][0], a[i][1]);

            printf("\n");

        }

    }

}

//Actvity selection STOP

// Greedy Method to avoid time-overlapping courses

void avoidTimeOverlappingCourses() {

    // Greedy method implementation

    int n;

    printf("Number of courses (from which we will determine maximum how many and which courses you can take without time overlap): ");

    scanf("%d", &n);

    int a[n][2];

    for(int i=0; i<n; i++){

        printf("Give the Start & End time of course %d: ", i+1);

        scanf("%d%d", &a[i][0], &a[i][1]);

    }

    activity\_selection(n, a);

}

// Structure to store student information

struct Student {

    char name[50];

    int id;

    char department[50];

    char courses[MAX\_COURSES][50]; // Assuming a student can take up to 5 courses

    int marks[MAX\_COURSES];

    char session[10];

    float cgpa;

};

//Merge sort START

void merge(struct Student \*arr,int l,int mid,int r){

int i=0,j=0,k=l;

int n1 = mid-l+1;

int n2 = r-mid;

struct Student left\_arr[n1];

struct Student right\_arr[n2];

for(int i=0; i<n1; i++){

    left\_arr[i]=arr[l+i];

}

for(int j=0; j<n2; j++){

    right\_arr[j]=arr[mid+1+j];

}

while(i<n1 && j<n2){

    if(left\_arr[i].id<=right\_arr[j].id){

    arr[k]=left\_arr[i];

    i++;}

    else{

      arr[k]=right\_arr[j];

    j++;

    }

    k++;

}

while(i<n1){

   arr[k]=left\_arr[i];

    i++;

    k++;

}

while(j<n2){

   arr[k]=right\_arr[j];

    j++;

    k++;

}

}

void merge\_sort(struct Student \*arr,int l,int r){

if(l<r){

    int mid = (l+r)/2;

    merge\_sort(arr,l,mid);

    merge\_sort(arr,mid+1,r);

    merge(arr,l,mid,r);

}

}

//Merge sort END

//Binary search START

void binary\_search(struct Student \*arr,int n){

  int search\_id,left=0,right=n-1,flag=0;

  printf("Enter ID : ");

  scanf("%d",&search\_id);

  while(left <= right){

    int mid=(left+right)/2;

    if(search\_id < arr[mid].id){

    right = mid-1;

  }

   else if(search\_id > arr[mid].id){

    left = mid+1;

  }

  else if(search\_id == arr[mid].id){

        printf("\nName: %s\n", arr[mid].name);

        printf("ID: %d\n", arr[mid].id);

        printf("Department: %s\n", arr[mid].department);

        printf("Courses:\n");

        for (int j = 0; j < 5; j++) {

            if (strcmp(arr[mid].courses[j], "END") == 0) {

                break;

            }

            printf("Courses %d :%s \n",j+1, arr[mid].courses[j]);

        }

        printf("Session: %s\n", arr[mid].session);

        printf("\n");

    flag=1;

    break;

  }

}

  if(flag != 1){

  printf("NOT FOUND!!");

  printf("\n");

  }

  }

  //Binary search END

// Function to add a new student to the system

void addStudent(struct Student \*\*studentList, int \*numStudents) {

    // Prompt the user for student information

    struct Student newStudent;

    printf("\nEnter student name: ");

    scanf(" %[^\n]s", newStudent.name);

    printf("Enter student ID: ");

    scanf("%d", &newStudent.id);

    printf("Enter student department: ");

    scanf("%s", newStudent.department);

    // Assuming up to 5 courses can be entered

    printf("Enter up to 5 courses (one per line, 'END' to finish):\n");

    for (int i = 0; i < 5; i++) {

        printf("Course %d: ", i + 1);

        scanf("%s", newStudent.courses[i]);

        if (strcmp(newStudent.courses[i], "END") == 0) {

            break;

        }

    }

    printf("Enter student session: ");

    scanf("%s", newStudent.session);

    // Increase the size of the student list

    (\*numStudents)++;

    \*studentList = realloc(\*studentList, sizeof(struct Student) \* (\*numStudents));

    // Add the new student to the list

    (\*studentList)[(\*numStudents) - 1] = newStudent;

    printf("\nStudent added successfully!\n\n");

}

// Function to display all students

void displayStudentsInfo(struct Student \*studentList, int numStudents) {

    if (numStudents == 0) {

        printf("\nNo students found.\n");

        return;

    }

    printf("List of Students:\n");

    for (int i = 0; i < numStudents; i++) {

        printf("\n\nStudent %d:\n", i + 1);

        printf("Name: %s\n", studentList[i].name);

        printf("ID: %d\n", studentList[i].id);

        printf("Department: %s\n", studentList[i].department);

        printf("Session: %s\n", studentList[i].session);}

}

void storeCourseMarks(struct Student \*studentList, int numStudents){

     if (numStudents == 0) {

        printf("No students found.\n");

        return;

    }

    printf("List of Students:\n");

    for (int i = 0; i < numStudents; i++) {

        float cg = 0;

        int total\_course=0;

        printf("Student %d:\n", i + 1);

        printf("Name: %s\n", studentList[i].name);

        printf("ID: %d\n", studentList[i].id);

        printf("Department: %s\n", studentList[i].department);

        printf("Session: %s\n", studentList[i].session);

        printf("Courses and Marks:\n");

        for (int j = 0; j < MAX\_COURSES; j++) {

            if (strcmp(studentList[i].courses[j], "END") == 0) {

                break;

            }

           // printf("  %s: \n", studentList[i].courses[j]);

             printf("Enter marks for %s: ",  studentList[i].courses[j]);

             scanf("%d", &studentList[i].marks[j]);

             total\_course++;

             if (studentList[i].marks[j] >= 80) {

             cg = 4.00 + cg;

            } else if (studentList[i].marks[j] >= 75) {

             cg = 3.75 + cg;

             } else if (studentList[i].marks[j] >= 70) {

             cg = 3.50 + cg;

            } else if (studentList[i].marks[j] >= 65) {

             cg = 3.24 + cg;

             } else if (studentList[i].marks[j] >= 60) {

              cg = 3.00 + cg;}

            else if (studentList[i].marks[j] >= 55) {

           cg = 2.75 + cg;

          } else if (studentList[i].marks[j] >= 50) {

            cg = 2.50 + cg;

            } else if (studentList[i].marks[j] >= 45) {

              cg = 2.25 + cg;

            } else if (studentList[i].marks[j] >= 40) {

            cg = 2.00 + cg;}

            else

         {//printf("Grade: F\n");

          cg = 0.00;

         break;}

        }

        studentList[i].cgpa=cg/total\_course;

        printf("\nCGPA: %.2f\n", studentList[i].cgpa);

        printf("\n");

    }

}

void enQueue(char \*query) {

  if ((front == rear + 1) || (front == 0 && rear == SIZE - 1))

    printf("\nList of Academic Queries is full!! \n");

  else {

    if (front == -1)

      front = 0;

    rear = (rear + 1) % SIZE;

    strcpy(queue[rear], query);

    printf("\nInserted Query:  %s\n", queue[rear]);

  }

}

void deQueue() {

  char \*query;

  if (front == -1) {

    printf("\nThere is no academic query!! \n");

  } else {

    query = queue[front];

    if (front == rear) {

      front = -1;

      rear = -1;

    } else {

      front = (front + 1) % SIZE;

    }

    printf("\nDeleted Query: %s \n", query);

  }

}

// Display the queue

void display() {

  char \*query;

  int i;

 if (front == -1)

    printf(" \nThere is no academic query!!\n");

  else {

    int j =1;

    printf("\nAcademic Query list:  \n");

    for (i = front; i != rear; i = (i + 1) % SIZE) {

      printf("%d.%s \n", j,queue[i]);

     j++;

    }

    printf("%d.%s\n",j, queue[i]);

  }

}

void Academic\_query(){

 int op;

while(op !=3){

printf("\nTown Hall Meeting Academic Query System:\n");

printf("1. Enqueue Query\n");

printf("2. Dequeue Query\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &op);

switch (op) {

case 1: {

char query[256];

 printf("Enter query: ");

 scanf(" %[^\n]s", query);

 enQueue(query);

 break;

 }

 case 2:

 deQueue();

 break;

 case 3:

 break;

 default:

 printf("Invalid choice. Please try again.\n");}

 } ;

}

// Dynamic Programming to maximize courses with limited credits

void maximizeCoursesWithLimitedCredits() {

    printf("Your maximum capacity of contact hour is 8h\n");

    printf("Enter number of courses (from which to determine maximum how many courses can you take with max credit): ");

    int n, C=8;

    scanf("%d", &n);

    int w[n], v[n];

    for(int i=0; i<n; i++){

        printf("Enter %dth courses credit and contact hour: ", i+1);

        scanf("%d%d", &v[i],&w[i]);

    }

    knapsack01(n, C, w, v);

}

//Floyd warshal algorithm START

void floyd\_warshall(){

  int n;

  printf("Enter number of vatices: ");

  scanf("%d", &n);

  int a[n][n];

  printf("Enter graph of course-career map as adjecency matrix(>=100 for Infinity): \n");

  for(int i=0; i<n; i++){

    for(int j=0; j<n; j++){

      scanf("%d", &a[i][j]);

    }

  }

  for(int k=0; k<n; k++){

    for(int i=0; i<n; i++){

      for(int j=0; j<n; j++){

        if(a[i][j]>(a[i][k]+a[k][j])){

          a[i][j]=a[i][k]+a[k][j];

        }

      }

    }

  }

  printf("Optimized graph: All-pairs least cost path:\n");

  for(int i=0; i<n; i++){

    for(int j=0; j<n; j++){

      printf("%d\t", a[i][j]);

    }

    printf("\n");

  }

  printf("\n");

}

//Floyd warshal algorithm END

int main() {

    struct Student \*studentList = NULL;

    int numStudents = 0;

    int choice;

    do {

        SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_RED | FOREGROUND\_INTENSITY);

        printf("\t\t\*\*School Management System\*\*\n\n");

        sleep(1);

        printf("1. Add Student\n");

        printf("2. Display Students\n");

        printf("3. Map Courses and Career Paths\n");

        printf("4. Sort student by ID\n");

        printf("5. Search student by ID\n");

        printf("6. Store Course marks & Show result\n");

        printf("7. Take Academic Queries on Town Hall Meeting.\n");

        printf("8. List of Academic Queries on Town Hall Meeting.\n");

        printf("9. Maximum obtainable credit.\n");

        printf("10. Avoid taking time overlapping courses.\n");

        printf("0. Exit\n");

        SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE),  FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

        printf("\nEnter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                addStudent(&studentList, &numStudents);

                break;

            case 2:

                displayStudentsInfo(studentList, numStudents);

                break;

            case 3:

                floyd\_warshall();

                break;

            case 4:

               merge\_sort(studentList, 0, numStudents - 1);

                break;

            case 5:

                merge\_sort(studentList, 0, numStudents - 1);

                binary\_search(studentList,numStudents);

                break;

            case 6:

                storeCourseMarks(studentList, numStudents);

                break;

            case 7:

                Academic\_query();

                break;

            case 8:

                display();

                break;

            case 9:

                maximizeCoursesWithLimitedCredits();

                break;

            case 10:

                avoidTimeOverlappingCourses();

            case 0:

                free(studentList);

                exit(0);

            default:

                printf("Invalid choice. Please try again.\n");

        }

    } while(1);

    return 0;

    }