AMITY
UNIVERSITY
Kolkata

Assignment

Name of the Student: Prateek Kumar

Enrolment No.- A910119824003

Program- B.Tech (AI)

Semester- 1st

Batch: 2024-28

Course Name: Introduction to Programming in C

Course Code: ES202

Date of Submission: 27

Name of the Faculty Coordinator: Mr. Nirmalya Tripathi

(1) WACP to check whether a number is prime or not

Algorithm:

1. Start.
2. Read the number **n**.
3. If **n <= 1**, print "Not Prime" and exit.
4. For **i** from 2 to **sqrt(n)**:
    - If **n % i == 0**, print "Not Prime" and exit.
5. Print "Prime".
6. End.

C Code:

```
#include <stdio.h>
#include <math.h>

int main() {
  int n, i, isPrime = 1;
  printf("Enter a number: ");
  scanf("%d", &n);

  if (n <= 1) {
    printf("%d is Not Prime\n", n);S
    return 0;
  }

  for (i = 2; i <= sqrt(n); i++) {
    if (n % i == 0) {
      isPrime = 0;
      break;
    }
  }

  if (isPrime)
```

```
        printf("%d is Prime\n", n);
    else
        printf("%d is Not Prime\n", n);


    return 0;
}
```

Output:

```
Enter a number: 7
7 is Prime
```

(2) Define storage class in C and describe the classification of it

Storage Classes in C: Storage classes in C define the scope (visibility) and lifetime of variables and functions. The four storage classes in C are:

1. **Automatic (auto)**: Default storage class for local variables. They are created when the block is entered and destroyed when it is exited.
2. **External (extern)**: Used to declare a global variable that is defined in another file or later in the same file.
3. **Static**: Used to declare a variable that retains its value even after it goes out of scope. It can be used for both local and global variables.
4. **Register**: Suggests to the compiler to store the variable in a register instead of RAM for faster access. It is also a local variable.

(3) Write a C program to add 2 matrices using 2D array

Algorithm:

1. Start.
2. Read the dimensions of the matrices (rows and columns).
3. Read the elements of the first matrix.
4. Read the elements of the second matrix.
5. Initialize a result matrix.
6. Add corresponding elements of both matrices.
7. Print the result matrix.
8. End.

C Code:

```
#include <stdio.h>

int main() {
    int rows, cols, i, j;
    printf("Enter number of rows and columns: ");
    scanf("%d %d", &rows, &cols);

    int a[rows][cols], b[rows][cols], sum[rows][cols];

    printf("Enter elements of first matrix:\n");
    for (i = 0; i < rows; i++)
        for (j = 0; j < cols; j++)
            scanf("%d", &a[i][j]);

    printf("Enter elements of second matrix:\n");
    for (i = 0; i < rows; i++)
        for (j = 0; j < cols; j++)
            scanf("%d", &b[i][j]);

    // Adding matrices
    for (i = 0; i < rows; i++)
        for (j = 0; j < cols; j++)
            sum[i][j] = a[i][j] + b[i][j];

    printf("Sum of the matrices:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++)
            printf("%d ", sum[i][j]);
        printf("\n");
    }
    return 0;
}
```

Output:

```
Enter number of rows and columns: 2 2
Enter elements of first matrix:
1 2
3 4
Enter elements of second matrix:
5 6
7 8
Sum of the matrices:
6 8
10 12
```

(4) Write to multiply 2 matrices using 2D array

Algorithm:

1. Start.
2. Read the dimensions of the matrices (rows and columns).
3. Read the elements of the first matrix.
4. Read the elements of the second matrix.
5. Initialize a result matrix.
6. Multiply the matrices.
7. Print the result matrix.
8. End.

C Code:

```
#include <stdio.h>

int main() {
    int r1, c1, r2, c2, i, j, k;

    printf("Enter rows and columns for first matrix: ");
```

```
scanf("%d %d", &r1, &c1);
printf("Enter rows and columns for second matrix: ");
scanf("%d %d", &r2, &c2);

// Check if multiplication is possible
if (c1 != r2) {
  printf("Matrix multiplication not possible.\n");
  return 0;
}

int a[r1][c1], b[r2][c2], product[r1][c2];

// Read first matrix
printf("Enter elements of first matrix:\n");
for (i = 0; i < r1; i++)
  for (j = 0; j < c1; j++)
    scanf("%d", &a[i][j]);

// Read second matrix
printf("Enter elements of second matrix:\n");
for (i = 0; i < r2; i++)
  for (j = 0; j < c2; j++)
    scanf("%d", &b[i][j]);

// Initialize product matrix to 0
for (i = 0; i < r1; i++)
  for (j = 0; j < c2; j++)
    product[i][j] = 0;

// Multiply matrices
for (i = 0; i < r1; i++) {
        for (j = 0; j < c2; j++) {
            for (k = 0; k < c1; k++) {
                product[i][j] += a[i][k] * b[k][j];
            }
        }
    }

    // Print product matrix
    printf("Product of the matrices:\n");
    for (i = 0; i < r1; i++) {
        for (j = 0; j < c2; j++)
            printf("%d ", product[i][j]);
        printf("\n");
    }

    return 0;
}
```

Output:

```
Enter rows and columns for first matrix: 2 3
Enter rows and columns for second matrix: 3 2
Enter elements of first matrix:
1 2 3
4 5 6
Enter elements of second matrix:
7 8
9 10
11 12
Product of the matrices:
58 64
```

139 154

(5) Write a program using a pointer array to sort an integer data

Algorithm:

1. Start.
2. Read the number of elements.
3. Allocate memory for an array of integers.
4. Read the elements into the array.
5. Create an array of pointers and point them to the elements of the array.
6. Sort the pointers based on the values they point to using a sorting algorithm (e.g., bubble sort).
7. Print the sorted values by dereferencing the pointers.
8. Free the allocated memory.
9. End.

C Code:

```
#include <stdio.h>
#include <stdlib.h>

void sort(int **arr, int n) {
  int *temp;
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (*arr[j] > *arr[j + 1]) {
        temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}
```

```
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int *data = (int *)malloc(n * sizeof(int));
    int **ptr = (int **)malloc(n * sizeof(int *));

    printf("Enter the elements:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &data[i]);
        ptr[i] = &data[i];
    }

    sort(ptr, n);

    printf("Sorted elements:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", *ptr[i]);
    }
    printf("\n");

    free(data);
    free(ptr);
    return 0;
}
```

Output:

```
Enter number of elements: 5
Enter the elements:
34 12 5 67 23
Sorted elements:
```

```
5 12 23 34 67
```

(6) Write a program to sort an array using insertion and binary sort

Insertion Sort Algorithm:

1. Start.
2. For each element in the array (starting from the second element):
    - Store the current element.
    - Compare it with the elements before it and insert it in the correct position.
3. End.

Binary Insertion Sort Algorithm:

1. Start.
2. For each element in the array (starting from the second element):
    - Use binary search to find the correct position for the current element in the sorted part of the array.
    - Shift elements to make space and insert the current element.
3. End.

C Code:

```
#include <stdio.h>


// Function to perform insertion sort
void insertionSort(int arr[], int n) {
  for (int i = 1; i < n; i++) {
    int key = arr[i];
    int j = i - 1;


    // Move elements of arr[0..i-1], that are greater than key,
    // to one position ahead of their current position
    while (j >= 0 && arr[j] > key) {
      arr[j + 1] = arr[j];
      j = j - 1;
    }
```

```
        arr[j + 1] = key;
    }
}

// Function to perform binary search
int binarySearch(int arr[], int item, int low, int high) {
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == item)
            return mid + 1; // Return the position after mid
        else if (arr[mid] < item)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return low; // Return the position to insert
}

// Function to perform binary insertion sort
void binaryInsertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        // Find location where key should be inserted
        int loc = binarySearch(arr, key, 0, j);

        // Move all elements after location to one position ahead
        while (j >= loc) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[loc] = key;
    }
}

int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter the elements:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Insertion Sort
    printf("Array before Insertion Sort:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    insertionSort(arr, n);
    printf("Array after Insertion Sort:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Resetting the array for Binary Insertion Sort
    printf("Enter the elements again for Binary Insertion Sort:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Binary Insertion Sort
    printf("Array before Binary Insertion Sort:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    binaryInsertionSort(arr, n);
    printf("Array after Binary Insertion Sort:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Output:

```
Enter number of elements: 5
Enter the elements:
34 12 5 67 23
Array before Insertion Sort:
34 12 5 67 23
Array after Insertion Sort:
5 12 23 34 67
Enter the elements again for Binary Insertion Sort:
34 12 5 67 23
```

```
Array before Binary Insertion Sort:

34 12 5 67 23

Array after Binary Insertion Sort:

5 12 23 34 67
```