ESc 101: Fundamentals of Computing

Lecture 22

Feb 22, 2010

Summary So far

Concepts learned so far:

- Statements: assignment, conditional, loop
- Arrays
- Functions
- Strings
- Pointers

This is nearly all of C, except for a couple of concepts more.

Summary So far

Concepts learned so far:

- Statements: assignment, conditional, loop
- Arrays
- Functions
- Strings
- Pointers

This is nearly all of C, except for a couple of concepts more.

Summary So far

Concepts learned so far:

- Statements: assignment, conditional, loop
- Arrays
- Functions
- Strings
- Pointers

This is nearly all of C, except for a couple of concepts more.

Summary So far

Concepts learned so far:

- Statements: assignment, conditional, loop
- Arrays
- Functions
- Strings
- Pointers

This is nearly all of C, except for a couple of concepts more.

Summary So far

Concepts learned so far:

- Statements: assignment, conditional, loop
- Arrays
- Functions
- Strings
- Pointers

This is nearly all of C, except for a couple of concepts more.

Summary So far

Concepts learned so far:

- Statements: assignment, conditional, loop
- Arrays
- Functions
- Strings
- Pointers

This is nearly all of C, except for a couple of concepts more.

Outline

1. More on Arrays

Storing Matrices

We use two dimensional arrays for this:

```
#define SIZE 100

int matrix[SIZE][SIZE]; /* matrix of size 100 x 100 */
```

Function for Reading a Matrix

```
/* Reads a matrix */
void read_matrix(int matrix[][SIZE])
{
    int i;
    int j;

    for (i = 0; i < SIZE; i++) /* read i-th row */
        for (j = 0; j < SIZE; j++)
            /* read j-th element of i-th row */
            scanf(" %d", &matrix[i][j]);
}
```

Two Dimensional Arrays

- `matrix[SIZE][SIZE]` is a two dimensional array:
  - `matrix[0]`, `matrix[1]`, ..., `matrix[99]` are each arrays of size SIZE each.
  - These are also called single dimensional arrays.
- `matrix[0][0]`, ..., `matrix[99][99]` are names of memory locations each storing an integer.

Two Dimensional Arrays

- `matrix[SIZE][SIZE]` is a two dimensional array:
  - `matrix[0]`, `matrix[1]`, ..., `matrix[99]` are each arrays of size `SIZE` each.
  - These are also called single dimensional arrays.
- `matrix[0][0]`, ..., `matrix[99][99]` are names of memory locations each storing an integer.

Two Dimensional Arrays

- `matrix[SIZE][SIZE]` is a two dimensional array:
  - `matrix[0]`, `matrix[1]`, ..., `matrix[99]` are each arrays of size `SIZE` each.
  - These are also called single dimensional arrays.
- `matrix[0][0]`, ..., `matrix[99][99]` are names of memory locations each storing an integer.

Two Dimensional Arrays

- `matrix[SIZE][SIZE]` is a two dimensional array:
  - `matrix[0]`, `matrix[1]`, ..., `matrix[99]` are each arrays of size `SIZE` each.
  - These are also called single dimensional arrays.
- `matrix[0][0]`, ..., `matrix[99][99]` are names of memory locations each storing an integer.

Names Associated with Two Dimensional Arrays

- `matrix[i][j]` stores a single integer value.
- `matrix[i]` stores the address of `matrix[i][0]`.
- `matrix` stores the address of `matrix[0][0]`, same as `matrix[0]`.

Names Associated with Two Dimensional Arrays

- `matrix[i][j]` stores a single integer value.
- `matrix[i]` stores the address of `matrix[i][0]`.
- `matrix` stores the address of `matrix[0][0]`, same as `matrix[0]`.

Names Associated with Two Dimensional Arrays

- `matrix[i][j]` stores a single integer value.
- `matrix[i]` stores the address of `matrix[i][0]`.
- `matrix` stores the address of `matrix[0][0]`, same as `matrix[0]`.

Passing Two Dimensional Array as Parameter

- We can pass the name of the matrix, a pointer to the first element, as parameter.
- In the declaration of the function, it must be specified either as `read_matrix(int matrix[][SIZE])`, or as `read_matrix(int *matrix[SIZE])`.
- In particular, it cannot be specified as `read_matrix(int matrix[][])`.
- This is because the function needs to be told that the pointer is to an array of what size.
- This allows correct calculation of `*(matrix+i)`.

Passing Two Dimensional Array as Parameter

- We can pass the name of the matrix, a pointer to the first element, as parameter.
- In the declaration of the function, it must be specified either as `read_matrix(int matrix[][SIZE])`, or as `read_matrix(int *matrix[SIZE])`.
- In particular, it cannot be specified as `read_matrix(int matrix[][])`.
- This is because the function needs to be told that the pointer is to an array of what size.
- This allows correct calculation of `*(matrix+i)`.

Passing Two Dimensional Array as Parameter

- We can pass the name of the matrix, a pointer to the first element, as parameter.
- In the declaration of the function, it must be specified either as `read_matrix(int matrix[][SIZE])`, or as `read_matrix(int *matrix[SIZE])`.
- In particular, it cannot be specified as `read_matrix(int matrix[][])`.
- This is because the function needs to be told that the pointer is to an array of what size.
- This allows correct calculation of `*(matrix+i)`.

Passing Two Dimensional Array as Parameter

- We can pass the name of the matrix, a pointer to the first element, as parameter.
- In the declaration of the function, it must be specified either as `read_matrix(int matrix[][SIZE])`, or as `read_matrix(int *matrix[SIZE])`.
- In particular, it cannot be specified as `read_matrix(int matrix[][])`.
- This is because the function needs to be told that the pointer is to an array of what size.
- This allows correct calculation of `*(matrix+i)`.

Passing Two Dimensional Array as Parameter

- We can pass the name of the matrix, a pointer to the first element, as parameter.
- In the declaration of the function, it must be specified either as `read_matrix(int matrix[][SIZE])`, or as `read_matrix(int *matrix[SIZE])`.
- In particular, it cannot be specified as `read_matrix(int matrix[][])`.
- This is because the function needs to be told that the pointer is to an array of what size.
- This allows correct calculation of `*(matrix+i)`.

Program for Multiplying Two Matrices

```
#define SIZE 100

main()
{
    int A[SIZE][SIZE]; /* input matrix */
    int B[SIZE[[SIZE]; /* input matrix */
    int C[SIZE][SIZE]; /* resulting matrix */

    read_matrix(A);
    read_matrix(B);
    multiply_matrix(A, B, C);
    output_matrix(C);
}
```

multiply_matrix()

```
/* Calculates C = A * B */
multiply_matrix(int A[][SIZE], int B[][SIZE], int C[][SIZE])
{
    for (int i = 0; i < SIZE; i++)
        for (int j = 0; j < SIZE; j++) {
            C[i][j] = 0; /* initialize */
            for (int k = 0; k < SIZE; k++)
                C[i][j] = C[i][j] + A[i][k] * B[k][j];
        }
}
```

output_matrix()

```
/* Outputs a matrix */
output_matrix(int A[][SIZE])
{
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++)
            printf("%d ", A[i][j]);
    printf("\n");
}
```