ESc 101: Fundamentals of Computing

Lecture 35

Apr 7, 2010

Redefining Type Matrix

Define Matrix as:

```
typedef struct matrix Matrix;
```

Or, do it directly as:

```
typedef struct matrix {
    int rows; // number of rows
    int cols; // number of columns
    float **elements; // pointer to elements
} Matrix;
```

Defining Type Vector

```
typedef struct {
    int dim; // dimension of the vector
    float *elements; // pointer to elements
} Vector;
```

Rewriting Matrix Multiplication

```
int multiply_matrix(Matrix A, Matrix B, Matrix C)
{
    Matrix D; // temp storage

    if (A.cols != B.rows) // error: cannot multiply
        return -1; // return a negative number denoting error

    D = allocate_matrix(A.rows, B.cols);
```

Rewriting Matrix Multiplication

```
    for (int i = 0; i < A.rows; i++)
        for (int j = 0; i < B.cols; j++) {
            D.elements[i][j] = 0; // initialize
            for (int k = 0; k < A.cols; k++)
                D.elements[i][j] +=
                    A.elements[i][k] * B.elements[k][j];
        }

        copy_matrix(C, 0, D, 0, D.rows, D.cols);
        C.rows = D.rows;
        C.cols = D.cols;
        return 1;
}
```

This Does Not Work!

- First two fields of matrix `C` are assigned inside the function.
- Observe that `C.elements` does not change!
- After exiting the function, the changed value for these two fields will not be available.
- To get around this, we adopt the standard method: instead of matrix, pass a pointer to it.

Rewriting Matrix Multiplication II

```
int multiply_matrix(Matrix A, Matrix B, Matrix *C)
{
    Matrix D; // temp storage

    if (A.cols != B.rows) // error: cannot multiply
        return -1; // return a negative number denoting error

    D = allocate_matrix(A.rows, B.cols);
```

Rewriting Matrix Multiplication II

```
    for (int i = 0; i < A.rows; i++)
        for (int j = 0; i < B.cols; j++) {
            D.elements[i][j] = 0; // initialize
            for (int k = 0; k < A.cols; k++)
                D.elements[i][j] +=
                    A.elements[i][k] * B.elements[k][j];
        }

        copy_matrix(*C, 0, D, 0, D.rows, D.cols);
        (*C).rows = D.rows;
        (*C).cols = D.cols;
        return 1;
}
```

A Shorthand

- Expression of the form `(*<name>).<field>` is very common when working with structures.
- C provides an alternative way of writing it: `<name>-><field>`.
- For example, `(*C).cols` can be written as `C->cols`.

Rewriting Matrix Multiplication III

```
int multiply_matrix(Matrix A, Matrix B, Matrix *C)
{
    Matrix D; // temp storage

    if (A.cols != B.rows) // error: cannot multiply
        return -1; // return a negative number denoting error

    D = allocate_matrix(A.rows, B.cols);
```

Rewriting Matrix Multiplication III

```
  for (int i = 0; i < A.rows; i++)
    for (int j = 0; i < B.cols; j++) {
      D.elements[i][j] = 0; // initialize
      for (int k = 0; k < A.cols; k++)
        D.elements[i][j] +=
          A.elements[i][k] * B.elements[k][j];
    }

    copy_matrix(*C, 0, D, 0, D.rows, D.cols);
    C->rows = D.rows;
    C->cols = D.cols;
    return 1;
}
```

Freeing Up Space

- Matrix variable `D` is allocated space in the `multiply_matrix()` function.
- This should be freed up when we exit the function.
- Otherwise, the space will remain assigned to the program until it exits.
- This was not the case when `D` was a two-dimensional array, as compiler makes sure that the space for local variables is freed up when the function is over.
- So we will have the strange situation where space allocated to variable `D` is freed up (this space is for storing the three fields) but the space allocated via `malloc()` is not freed!

Function free_matrix()

```
void free_matrix(Matrix A)
{
    for (int i = 0; i < A.rows; i++)
        free(A.elements[i]);

    free(A.elements);
}
```

Rewriting Matrix Multiplication IV

```
int multiply_matrix(Matrix A, Matrix B, Matrix *C)
{
    Matrix D; // temp storage

    if (A.cols != B.rows) // error: cannot multiply
        return -1; // return a negative number denoting error

    D = allocate_matrix(A.rows, B.cols);
```

Rewriting Matrix Multiplication IV

```
  for (int i = 0; i < A.rows; i++)
    for (int j = 0; i < B.cols; j++) {
      D.elements[i][j] = 0; // initialize
      for (int k = 0; k < A.cols; k++)
        D.elements[i][j] +=
          A.elements[i][k] * B.elements[k][j];
    }

    copy_matrix(*C, 0, D, 0, D.rows, D.cols);
    C->rows = D.rows;
    C->cols = D.cols;

    free_matrix(D); // free up space
    return 1;
}
```