Declaring Arrays

• Instead of writing:
  ```c
  int x;
  ```
• we can write:
  ```c
  int x[10];
  ```
• the name x refers to the collection (or array) of integer values, which can contain up to 10 values.
Using An Array

• We can assign a value to any element in the array by specify the array by name and its index:
  \[ x[0] = 87; \]
  \[ x[1] = 90; \]
  \[ \ldots \ldots \ldots \]  
  \[ x[9] = 93; \]

Using An Array (continued)

• An index can be any integer or character literal, constant, variable or expression:
  \[ x[\text{Five}] = 34; \]
  \[ x[i+1] = x[i] + 3; \]

• This is really useful, because we do not want to have to write separate statement to assign values to each array element.
Using a Counting Loop To Set An Array

- Counting loops are really useful when manipulating arrays:
  
  ```c
  for (i = 0; i < 10; i++)
      printf("%d ", x[i]);
  ```

---

A Program To Find Class Average

```c
#include <stdio.h>

void getgrades(int grades[]);
int calcaverage(int grades[]);
void printresults(int grades[], int mean);
char lettergrade(int score);

#define numgrades 10
```
int main(void)
{
    int grades[numgrades], average;

    getgrades(grades);
    average = calcaverage(grades);
    printresults(grades, average);
    return(0);
}

void getgrades(int grades[])
{
    int count;
    for (count = 0; count < numgrades; count++) {
        printf("Enter a grade\t?");
        scanf("%d", &grades[count]);
    }
}

int calcaverage(int grades[])
{
    int count, sum = 0;
    for (count = 0; count < numgrades; count++)
    {
        sum = sum + grades[count];
    }
    return(sum/numgrades);
}
void printresults(int grades[], int mean)
{
  int i;

  printf("The grades are:\n");
  for (i = 0; i < 10; i++)
    printf("%d\n", grades[i]);
  printf("The average is %d", mean);
  printf(" corresponding to a grade of"  
          " %c\n", lettergrade(mean));
}

char lettergrade(int score)
{
  if (score >= 90)
    return('A');
  if (score >= 80)
    return('B');
  if (score >= 70)
    return('C');
  if (score >= 60)
    return('D');
  else
    return('F');
}
Selection Sorting

#include <stdio.h>

#define Size 5

void sort(int x[]);

/*
 * main() - A driver for the Selection Sort
 */
int main(void)
{
  int i, a[Size];

  for (i = 0; i < Size; i++) {
    printf("Enter a[%d]\t?", i);
    scanf("%d", &a[i]);
  }
  sort(a);

  for (i = 0; i < Size; i++)
    printf("a[%d] = %d\n", i, a[i]);
  return(0);
}
/*
 * sort() - Sort an array of numbers
 */
void sort(int x[])
{
    int i, j, small, index, temp;

    /*
    * Place the smallest number in the first
    * position
    * Place the second smallest in the second
    * position and so on.
    */
    for (i = 0; i < Size -1; i++) {
        small = 32767;
        index = -1;

        /*
        * Compare each number that is not in
        * its proper place to the smallest so
        * far
        */
        for (j = i; j < Size; j++)
            if (x[j] <small) {
                small = x[j];
                index = j;
            }
}
/*
 * Swap the ith smallest number into its
 * proper place
 */
temp = x[i];
x[i] = x[index];
x[index] = temp;
}
Example: Class Average on a Series of Exams

- Imagine that students in a class have taken four tests and their grades depend on the average score on this exam.
- Our data looks like this:

```
<table>
<thead>
<tr>
<th>test 1</th>
<th>test 2</th>
<th>test 3</th>
<th>test 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>grades</td>
<td>grades</td>
<td>grades</td>
<td>grades</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

### grades.c

```c
#include <stdio.h>

#define numstudents 30
#define numexams 4

void readgrades(int grades[][numexams]);
void findaverages(int averages[],
                   int grades[][numexams]);
void writegrades(int grades[][numexams],
                 int averages[]);
```

Required: tells how many columns there are
/*
 * CalcAverages() - Calculate the term
 * averages for a class
 * The average is based on
 * four exams
 */

int main(void)
{
    int grades[numstudents][numexams];
    int averages[numstudents];

    /*
    * Get the grades, find the averages and
    * print them
    */

    readgrades(grades);
    findaverages(averages, grades);
    writegrades(grades, averages);

    return(0);
}

/*  
* readgrades() - Read the complete set of grades  
*/  
void readgrades(int grades[][numexams])  
{  
    int i, j;  
    // Get each student's grade  
    for (i = 0; i < numstudents; i++) {  
        // Get the next grade for this student  
        for (j = 0; j < numexams; j++) {  
            printf("Grade on test \#%d for"  
                   " for student \# %d\t?",  
                   j, i);  
            scanf("%d", &grades[i][j]);  
        }  
        // Skip one line for clarity  
        printf("\n");  
    }  
}
// FindAverages() — Find the average for each student

void findaverages(int averages[],
                   int grades[][numexams])
{
    int i, j, sum;

    for (i = 0;  i < numstudents;  i++) {
        sum = 0;
        for (j = 0;  j < numexams;  j++)
            sum += grades[i][j];
        averages[i] = sum/numexams;
    }
}

// WriteAverage() — Output the grades and average for each student

void writegrades(int grades[][numexams],
                 int averages[])
{
    int i, j;

    // Print a heading
    printf("Student Exam1\tExam2\tExam3\tExam4\tAverage\n");

    for (i = 0;  i < numstudents;  i++) {
        // Number each line, then print the grades and the average for the next student
        printf("%d", i);
for (j = 0; j < numexams; j++)  
    printf("%4d", grades[i][j]);
}
printf("%d\n", averages[i]);
}

Limitations of grades.c

- The program has a major limitation: we must know exactly how many students there are.
- It would be better if we can safely guess our upper limit and count the exact number.
Matrices

- A matrix is a two-dimensional array of numbers, used in many types of mathematical problems.
- Adding and subtracting matrices is easy:
  - $a_{i,j} + b_{i,j} = c_{i,j}$
  - $a_{i,j} - b_{i,j} = c_{i,j}$
- Multiplying matrices is much harder:
  $$c_{i,j} = \sum_{k=1}^{n} a_{i,k} \times b_{k,j}$$

multmat.c

```c
#include <stdio.h>

#define numrows 4
#define numcolumns 4

void readmatrix(int matrix[][numcolumns]);
void multmatrix(int c[][numcolumns],
                int a[][numcolumns],
                int b[][numcolumns]);
void writematrix(int matrix[][numcolumns]);
```
int main(void)
{
    int a[numrows][numcolumns],
        b[numrows][numcolumns],
        c[numrows][numcolumns];

    printf("Enter matrix a\n");
    readmatrix(a);
    printf("Enter matrix b\n");
    readmatrix(b);
    multmatrix(c, a, b);
    printf("The product is:\n");
    writematrix(c);
    return(0);
}
/* 
  * readmatrix() - Read in a matrix 
  */
void readmatrix(int matrix[][numcolumns])
{
    int i, j;

    for (i = 0;  i < numrows; i++) {
        printf("Enter row #%d\t?", i+1);
        for (j = 0;  j < numcolumns;  j++)
            scanf("%d", &matrix[i][j]);
    }
}

/* 
  * multmatrix() – Multiply a x b to get c 
  */
void multmatrix(int c[][numcolumns],
                int a[][numcolumns],
                int b[][numcolumns])
{
    int i, j, k;

    for (i = 0; i < numrows; i++)
        for (j = 0;  j < numcolumns;  j++) {
            c[i][j] = 0;
            for (k = 0;  k < numrows;  k++)
                c[i][j] += a[i][k]*b[k][j];
        }
}
What is a Structure

- A structure is a heterogeneous collection of data.
- Even if the data type is the same, it may not belong in an array but in a structure.
Declaring A Structure

- A structure **containing** the rate of pay and hours worked might look like this:
  ```c
  struct {
    int rate;
    int hours;
    int gross;
  } worker;
  ```

Declaring A Structure (continued)

- Alternatively, we can write:
  ```c
  struct {
    int rate, hours, gross;
  } worker;
  ```
Declaring A Structure (continued)

- We can give the structure a name and then declare variables as structures of this type very easily:
  ```c
  struct workerstuff {
      int rate, hours, gross;
  };
  ...
  int main(void)
  {
      struct workerstuff worker;
  }
  ```

Using A Structure

- To use a field within the structure, you must specify both the structure and the field with a period “.” in between:
  ```c
  scanf("%d",&worker.rate);
  scanf("%d",&worker.hours);
  worker.gross = worker.rate * worker.hours;
  ```
A simple payroll program

#include <stdio.h>

/*
 * a simple structure for payroll
 */
struct workerstuff {
    char name[20];
    float rate;
    float hours;
    float gross;
};

/*
 * payroll.c - A simple payroll program
 */
int main(void)
{
    struct workerstuff worker;
    printf("What is the worker\'s rate per "
           "hour?\t");
    scanf("%f", &worker.rate);

    printf("How many hours did the worker work "
           "last week?\t");
    scanf("%f", &worker.hours);
```c
printf("What is the worker\'s name?\t");
scanf("%s", &worker.name);
printf("%s worked for out %3.1f hours at $\n" "%4.2f per hour.\n", worker.name, worker.hours, worker.rate);

return(0);
}
```

**Structures Containing Arrays**

- A structure can have an array as a field within it. Examples of this include character strings.
- Our Dean’s List program could use this to include the grades that comprised our students’ g.p.a.
typedef

• There are times when it is useful to define one's own data types. You can do this with the typedef statement.
• Syntax:
  ```
  typedef DataTypeDataTypeName;
  ```
• Examples
  ```
  typedef int IntegerType;
  typedef int *IntPtr;
  ```

->

• If we are working with a pointer to a structure and we wish to reference one of the fields, we can write:
  ```
  *(myStructPtr.myField)
  ```
  or
  ```
  myStructPtr -> myField
  ```
• This second form is consider far better form.
avggrade.cpp

#include <stdio.h>

#define namelen 15
#define numexams 4

typedef struct {
    char firstname[namelen], lastname[namelen];
    int exam[numexams];
} examrec;

void readstudent(examrec *student);
float findaverage(examrec student);
void writestudent(examrec student, float average);

/*
 * AvgGrade() - Averages the grades on n exams
 */
int main(void)
{
    examrec student;
    float average;

    /* Read the students name and test scores */
    readstudent(&student);

    /* Find the average */
    average = findaverage(student);
}
/* Print the results */
writestudent(student, average);
return(0);
}

/*
* ReadStudent() —       Read the input about the
*                      student
*/
void readstudent(examrec *student)
{
    int i;
    printf("First name\t?\n");
    scanf("%s", student -> firstname);
    printf("Last name\t?\n");
    scanf("%s", student -> lastname);
for (i = 0; i < numexams; i++) {
    printf("Enter grade for exam #\%d\t?", i+1);
    scanf("%d", &student->exam[i]);
}

float findaverage(examrec student) {
    int i, sum = 0;
    for (i = 0; i < numexams; i++)
        sum += student.exam[i];
    return((float) sum/numexams);
}
/**
 * WriteStudent() - Print the data about the
 * student including the
 * average
 */
void writestudent(examrec student, float average)
{
    int i;
    printf("%s %s scored :\n", student.firstname,
           student.lastname);

    for (i = 0; i < numexams; i++)
        printf("%d\t", student.exam[i]);

    printf("\n\twhich resulted in an average of"
           "%3.1f\n", average);
}