Example

```c
/*
 * Separates a number into three parts: a sign (+, -, or blank),
 * a whole number magnitude, and a fractional part.
 */
void separate(double num, char *signp, int *wholep, double *fracp)
{
    double magnitude; /* local variable - magnitude of num */

    /* Determines sign of num */
    if (num < 0) *signp = '-';
    else if (num == 0) *signp = ' ';
    else *signp = '+';

    /* Separate num into whole (no sign) and fractional parts */
    magnitude = fabs(num);
    *wholep = floor(magnitude);
    *fracp = magnitude - *wholep;
}
```

Arrays

- Arrays are variables that allow storing multiple data that are related with each other.
- Arrays, similar to single value variables, can be of type int, char, double, etc.
- Let us represent temperature measurements taken every hour in a day:
  - \( T_0, T_1, \ldots, T_{23} \)
  - Array representation \( T[0], T[1], \ldots, T[23] \)
  - The numbers in brackets are called subscripts
  - The array has 24 elements, i.e. the array size is 24 (\( N = 24 \))
  - \( T_i \), where \( i \) goes from 0 to \( N - 1 \)
Declaring Arrays

- Like any other variable, arrays are declared in the beginning of the function they are used
  
  - **type** array_name[array_size];
  - **type** is the type of array elements (int, double, char, …)
  - **array_name** follows the same naming rules for the variables
  - **array_size** is an integer showing the number of elements in that array

  - double T[24];

Using Arrays

double x[8];

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0</td>
<td>2.1</td>
<td>5.2</td>
<td>9.3</td>
<td>2.5</td>
<td>5.5</td>
<td>1.9</td>
<td>-4.3</td>
</tr>
</tbody>
</table>

New x[3] = 7.5

i = 4; printf("%.1f", x[i]);
Prints 2.5

printf("%.1f + %.1f = %.1f", x[0], x[6], x[0]+x[6]);
Prints 10.0 + 1.9 = 11.9

i = 5; x[i] = x[i-1];

i = 2; printf("%.1f", x[--i]);
Prints 2.1. New i is 1.

i = 2; printf("%.1f", x[--i]);
Prints 5.2. New i is 1.
Example

/*
* This program allows the user to enter hourly temperature measurements for a day
* then calculates minimum, maximum, and average temperature for that day
*/
#include <stdio.h>

int main(void)
{
    double T[24]; /* Temperature array */
    int i; /* Array index */
    double Total = 0; /* The sum of all temperatures */
    double Average; /* The average temperature */
    double Minimum; /* The minimum temperature */
    double Maximum; /* The maximum temperature */

    /* Read hourly Temperature */
    for(i=0; i<24; i++)
    {
        printf("Please enter Temperature reading %2d>", i+1);
        scanf("%lf", &T[i]);
        printf("T[%d] = %f
", i, T[i]);
    }

    /* Calculate Average */
    for(i=0; i<24; i++) Total += T[i];
    Average = Total / 24;

    // Further code...
}
Example

/* Find minimum */
Minimum = T[0];
for(i=1;i<24;i++) if(T[i] < Minimum) Minimum = T[i];

/* Find maximum */
Maximum = T[0];
for(i=1;i<24;i++) if(T[i] > Maximum) Maximum = T[i];

/* Print Av, Min, and Max */
printf("The Average Temperature is:%4.1fn", Average);
printf("The Minimum Temperature is:%4.1fn", Minimum);
printf("The Maximum Temperature is:%4.1fn", Maximum);
}

Functions with Output Arguments

- Function with one input (non-array) and one output (as return)
  - Function Prototype
    - int ex1(int z);
  - Function Call
    - int x=0, y;
    - y = ex1(x);

- Function with one (non-array) input and one output (non-array)
  - Function Prototype
    - void ex1(int p, int *q);
  - Function Call
    - int x=0, y;
    - ex1(x, &y);
Array Elements as Arguments

- Function with one input (as array element) and one output (as return)
  - Function Prototype
    - int ex1(int z);
  - Function Call
    - int x[5], y;
    - x[3] = 0;
    - y = ex1(x[3]);

- Function with one input (non-array) and one output (as array element)
  - Function Prototype
    - void ex1(int p, int *q);
  - Function Call
    - int x=0, y[6];
    - ex1(x, &y[2]);

Arrays as Arguments

- Function with one input (as array) and one output (as return)
  - Function Prototype
    - int ex1(int z[]);
  - Function Call
    - int x[5], i;
    - for(i=0; i<5; i++) x[i] = i;
    - y = ex1(x);

- Function with one input (non-array) and one output (as array)
  - Function Prototype
    - void ex1(int p, int q[]);
  - Function Call
    - int x=0, y[6];
    - ex1(x, y);
Arrays as Arguments

- Function with one input (as array) and one output (as array)
  - Function Prototype
    - int ex1(int p[], int q[]);
  - Function Call
    - int x[5], y[6];
    - for(i=0; i<5; i++) x[i] = i;
    - ex1(x, y);

- Function with one input (array) and two outputs (one array one return)
  - Function Prototype
    - int ex1(int p[], int q[], double z);  
  - Function Call
    - int x[5], y[6], z;
    - for(i=0; i<5; i++) x[i] = i;
    - z = ex1(x, y); 

Since x and y are arrays, the call refers to the address of the first elements &x[0] and &y[0].

- p[ ] and q[ ] are pointers to the beginning of arrays - *p and *q

Example with Functions

```c
#include <stdio.h>
define NUM_TEMP 24  /* Number of Temperature Readings */

void ReadTemp(double x[], int N);  /* Reads N Temperature values to array */
double CalcAve(double x[], double *Ave, int N);  /* Calculates the average of N temperatures */
double FindMin(double x[], double *Min, int N);  /* Finds min of N temperatures */
double FindMax(double x[], double *Max, int N);  /* Finds max of N temperatures */

int main(void)
{
    double T[NUM_TEMP];  /* Temperature array */
    double Average;  /* The average temperature */
    double Minimum;  /* The minimum temperature */
    double Maximum;  /* The maximum temperature */

    // Read temperature values...
    ReadTemp(T, NUM_TEMP);

    // Calculate average...
    double Ave = CalcAve(T, &Average, NUM_TEMP);

    // Find minimum...
    double Min = FindMin(T, &Minimum, NUM_TEMP);

    // Find maximum...
    double Max = FindMax(T, &Maximum, NUM_TEMP);
}
```
/* Read hourly Temperature */
ReadTemp(T, NUM_TEMP);
/* Send the array to be filled and size of array */
/* T is the same as &T[0], it is an address */
/* Calculate Average */
CalcAve(T, &Average, NUM_TEMP);
/* Send array, variable to be calculated and size of array */
/* T is the same as &T[0], it is an address */
/* Find Minimum */
FindMin(T, &Minimum, NUM_TEMP);
/* Send array, variable to be found and size of array */
/* T is the same as &T[0], it is an address */
/* Find Maximum */
FindMax(T, &Maximum, NUM_TEMP);
/* Send array, variable to be found and size of array */
/* T is the same as &T[0], it is an address */
/* Print Av, Min, and Max */
printf("The Average Temperature is:%4.1f\n", Average);
printf("The Minimum Temperature is:%4.1f\n", Minimum);
printf("The Maximum Temperature is:%4.1f\n", Maximum);
return(0);

Dr. Sadık Eşmelioğlu
CENG 114 – Spring 2009-10
95

Example with Functions

/* Reads Temperature values to array */
void ReadTemp(double x[], int N) {
  int i;
  for(i=0; i<N; i++) {
    printf("Please enter Temperature reading %2d> ", i+1);
    scanf("%lf", &x[i]);
  }
}
/* Calculates the average temperature */
void CalcAve(double x[], double *Ave, int N) {
  int i;
  double Total = 0;
  for(i=0; i<N; i++) Total += x[i];
  *Ave = Total / N;
}
Example with Functions

/* Finds min temperature */
void FindMin (double x[], double *Min, int N) /* x[] is the same as *x, it is a pointer */
{
    int i;
    *Min = x[0];
    for(i=1;i<N;i++)  if(x[i] < *Min) *Min = x[i];
}

/* Finds max temperature */
void FindMax(double x[], double *Max, int N) /* x[] is the same as *x, it is a pointer */
{
    int i;
    *Max = x[0];
    for(i=1;i<N;i++)  if(x[i] > *Max) *Max = x[i];
}

Value – Address – Pointer

Main Function

<table>
<thead>
<tr>
<th>Name</th>
<th>Addr.</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>T[0]</td>
<td>7000</td>
<td>0</td>
</tr>
<tr>
<td>T[1]</td>
<td>7001</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T[23]</td>
<td>7023</td>
<td>46</td>
</tr>
</tbody>
</table>

ReadTemp Function

<table>
<thead>
<tr>
<th>Name</th>
<th>Addr.</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[]</td>
<td>7900</td>
<td>7000</td>
</tr>
</tbody>
</table>

ReadTemp(T, 24);

T = &T[0]

ReadTemp(double x[], int N);

x[] = *(x+0) = *(7000) = 0
x[0] = *(x+1) = *(7001) = 2
If Function not to modify Array

- If we do not want the called function to modify an array that is passed to it as argument, put the reserved word `const` in front of the argument in the formal parameter list

**PROTOTYPE:**
```c
void FindMin(const double x[], double *Min, int N);
```

**CALL:**
```c
FindMin(T, &Minimum, 24);
```

**DEFINITION:**
```c
void FindMin(const double x[], double *Min, int N) {
    int i;
    *Min = x[0];
    for(i=1;i<N; i++) if(x[i] < *Min) *Min = x[i];
}
```

Declaring Arrays with values

- **type array_name[] = {initialization_list};**
  - `type` is the type of array elements (int, double, char, …)
  - `array_name` follows the same naming rules for the variables
  - `initialization_list` is a comma separated list of values for each element
  - `array_size` is not needed since the number of elements in the list determines the size

- `char Suit[] = {'H', 'S', 'D', 'C'};`
  - array size is 4
  - Suit[0] is 'H'
  - Suit[1] is 'S'
  - Suit[2] is 'D'
  - Suit[3] is 'C'
**Example**

Redo the GetNumber function in BlackJack example (Page 111)

```c
/*
 * Given an integer from 0 – 12 returns the card name
 * 0 is 1, 1 is 2, ..., 9 is T, 10 is J, 11 is Q, 12 is K
 */
char GetNumber(int number)
{
    char Number[ ] = {'1', '2', '3', '4', '5', '6', '7', '8', '9', 'T', 'J', 'Q', 'K', '?'};
    if (number >=0 && number <= 12) return(Number[number]);
    else return(Number[13]);
}
```

**Initialize an Array**

**Prototype:**

```c
void InitArr(double x[ ], int N, double FillVal);
```

**Call:**

```c
double Arr[6] ;
double InitVal = 0.0;
InitArr(Arr, 6, InitVal);
```

**Function Definition:**

```c
void InitArr(double x[ ], int N, double FillVal)
{
    int i;
    for(i = 0; i <= N - 1; i++) x[i] = FillVal;
}
Increment an Array

Prototype:
void IncArr(double x[], int N);

Call:
double Arr[] = {2.1, 5.3, -4.5, 0.5, -3.0, 2.2};
IncArr(Arr, 6);

Function Definition:
void IncArr(double x[], int N)
{
    int i;
    for(i = 0; i <= N - 1; i++) x[i]++;
}

Add two Arrays

Prototype:
void Add2Arrs(const double x[], const double y[], double z[], int N);

Call:
double Arr1[] = {2.1, 5.3, -4.5, 0.5, -3.0, 2.2};
double Arr2[] = {1.2, 3.3, 5.1, -0.5, -3.3, 1.3};
double Arr3[6];
Add2Arrs(Arr1, Arr2, Arr3, 6);

Function Definition:
void Add2Arrs(const double x[], const double y[], double z[], int N)
{
    int i;
    for(i = 0; i <= N - 1; i++) z[i] = x[i] + y[i];
}
### Multiply two Arrays

**Prototype:**

```c
void Mult2Arrs(const double x[], const double y[], double z[], int N);
```

**Call:**

```c
double Arr1[] = {2.1, 5.3, -4.5, 0.5, -3.0, 2.2};
double Arr2[] = {1.2, 3.3, 5.1, -0.5, -3.3, 1.3};
double Arr3[6];

Mult2Arrs(Arr1, Arr2, Arr3, 6);
```

**Function Definition:**

```c
void Mult2Arrs(const double x[], const double y[], double z[], int N)
{
    int i;
    for(i = 0; i <= N - 1; i++) z[i] = x[i] * y[i];
}
```

### Find Min. of an Array w/return

**Prototype:**

```c
double FindMin(const double x[], int N);
```

**Call:**

```c
double Arr1[] = {2.1, 5.3, -4.5, 0.5, -3.0, 2.2};
double MinVal;

MinVal = FindMin(Arr1, 6);
```

**Function Definition:**

```c
double FindMin(const double x[], int N)
{
    int i;
    double Min = x[0];
    for(i = 1; i <= N - 1; i++)
    {
        if(x[i] < Min)
            Min = x[i];
    }
    return(Min);
}
```
Find Min. of an Array w/pointer

Prototype:
double FindMin(const double x[], double *Min, int N);

Call:
double Arr1[] = {2.1, 5.3, -4.5, 0.5, -3.0, 2.2};
double MinVal;
FindMin(Arr1, &MinVal, 6);

Function Definition:

void FindMin(const double x[], double *Min, int N)
{
    int i;
    double *Min = x[0];
    for(i = 1; i <= N - 1; i++)
        if(x[i] < *Min)
            *Min = x[i];
}

Sort Array Example

1. Problem
   - We have a file of Student Numbers with their Midterm, Lab, and Final Grades as shown:

<table>
<thead>
<tr>
<th>Student Number</th>
<th>Midterm</th>
<th>Lab</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>201011017</td>
<td>60</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>201111054</td>
<td>77</td>
<td>67</td>
<td>88</td>
</tr>
</tbody>
</table>

   - We would like a program that sorts this file, with respect to Student Number from the smallest to the largest
   - Assume maximum number of students is 100
Sort Array Example

2. Analysis
   - Many ways to sort an array
   - Let’s use the technique that requires n-1 iterations
   - The first iteration finds the row where the minimum is and exchanges row 1 with that row. Now row 1 has minimum
   - The second iteration starts with row 2, and finds the row where the next minimum is and exchanges row 2 with that row
   - This process continues until the last two rows are ordered.

x[5] = \{8 9 1 7 2\}
- Find the minimum Rows 0-4 → Row 2 is min
- Replace x[0] and x[2] → x[ ] = \{1 9 8 7 2\}
- Find the minimum Rows 1-4 → Row 4 is min
- Replace x[1] and x[4] → x[ ] = \{1 2 8 7 9\}
- Find the minimum Rows 2-4 → Row 3 is min
- Replace x[2] and x[3] → x[ ] = \{1 2 7 8 9\}
- Find the minimum Rows 3-4 → Row 3 is min
- Replace x[3] and x[3] → x[ ] = \{1 2 7 8 9\}
Sort Array Example

3. Design
   1. Read the file into four arrays, StudentNo[], MidGrd[], LabGrd[], and FinGrd[]. We can keep count of how many actual students we have read \( \rightarrow N \)
   2. Find minimum of rows i thru N where i goes from 0 to N-2 (Until the last two elements left)
   3. Replace row[minimum] with row[i]
   4. Increment i and go back to (2)
   5. Print the ordered arrays into an output file
Sort Array Example

4. Coding
/*
 * This Program sorts the contents of a file
 * according to the StudentNo
 * The file is assumed to have four integers in each row
 * StudentNo MidGrd LabGrd FinGrd
 */

#include <stdio.h>
define NOofSTUDENTS 100

int FillArrays(FILE *inpf, int w[], int x[], int y[], int z[]);
int FindMinRow(int x[], int k, int N);
int ReplaceRows(int k, int l, int w[], int x[], int y[], int z[]);
int PrintFile(FILE *outpf, int w[], int x[], int y[], int z[], int N);

int main (void)
{
    FILE  *inpf, *outpf;

    int StudentNo[NOofSTUDENTS],
        MidGrd[NOofSTUDENTS],
        LabGrd[NOofSTUDENTS],
        FinGrd[NOofSTUDENTS];

    int i, k, N;

    inpf = fopen("ClassGrades.txt", "r");
    outpf = fopen("ClassGradesS.txt", "w");
Sort Array Example

N = FillArrays(inpf, StudentNo, MidGrd, LabGrd, FinGrd);

for(i = 0; i < N-1; i++) {
    k = FindMinRow(StudentNo, i, N);
    if(i != k)
        ReplaceRows(k, i, StudentNo, MidGrd, LabGrd, FinGrd);
}

PrintFile(outpf, StudentNo, MidGrd, LabGrd, FinGrd, N);

fclose(inpf);
fclose(outpf);

return(0);

Sort Array Example

int FillArrays(FILE *inpf, int w[], int x[], int y[], int z[]) {
    int ReadStat;
    int i = 0;

    ReadStat = fscanf(inpf, "%d %d %d %d", &w[i], &x[i], &y[i], &z[i]);

    while (ReadStat != EOF) {
        i++;
        ReadStat = fscanf(inpf, "%d %d %d %d", &w[i], &x[i], &y[i], &z[i]);
    }

    return(i);
}
Sort Array Example

```c
int FindMinRow(int x[], int k, int N)
{
    int i;
    int xMin = x[k];
    int iMin = k;

    for (i = k+1; i < N; i++) {
        if(x[i] < xMin) {
            xMin = x[i];
            iMin = i;
        }
    }
    return(iMin);
}
```

Dr. Sadık Eşmelioğlu
CENG 114 – Spring 2009-10

Sort Array Example

```c
int ReplaceRows(int k, int l, int w[], int x[], int y[], int z[])
{
    int temp;

    temp = w[k];
    w[k] = w[l];
    w[l] = temp;

    temp = x[k];
    x[k] = x[l];
    x[l] = temp;

    temp = y[k];
    y[k] = y[l];
    y[l] = temp;

    temp = z[k];
    z[k] = z[l];
    z[l] = temp;

    return(0);
}
```

Dr. Sadık Eşmelioğlu
CENG 114 – Spring 2009-10
Sort Array Example

```c
int PrintFile(FILE *outpf, int w[], int x[], int y[], int z[], int N)
{
    int i;

    for(i = 0; i < N; i++) {
        fprintf(outpf, "%d\t%d\t%d\n", w[i], x[i], y[i], z[i]);
    }

    return(0);
}
```

5. Test
   a. Create an input file, make sure it is not sorted properly, then run the program. Observe that the output file is sorted properly.
   b. Create an input file, make sure it is sorted properly, then run the program. Observe that the output file is still sorted properly.
   c. Create an input file, make sure it is sorted from max to min, then run the program. Observe that the output file is sorted properly.

6. Maintain
   - Add an option to sort in ascending or descending order
   - Add an option to sort wrt Midterm, Lab, and Final grades
Multidimensional Arrays

- A multidimensional array is an array of two or more dimensions
- All the elements of the array have to be of the same type
- A two dimensional arrays is represented by two indices:

\[
\text{int } x[3][4]
\]

<table>
<thead>
<tr>
<th>row 0</th>
<th>column 0</th>
<th>column 1</th>
<th>column 2</th>
<th>column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 0</td>
<td>5</td>
<td>12</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>row 1</td>
<td>0</td>
<td>5</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>row 2</td>
<td>16</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Declaring Multidimensional Arrays

- \textbf{type} array\_name[size\_1] [size\_2] \ldots [size\_n];
  - \textbf{type} is the type of array elements (int, double, char, ...)
  - array\_name follows the same naming rules for the variables
  - \textbf{size} is an integer showing the number of elements in that dimension

- \textbf{int} Stdnt[24][5]; /* No, Mid, Lab, Fin, Grd*/

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>98765</td>
<td>78</td>
<td>89</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>78998</td>
<td>84</td>
<td>88</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>88723</td>
<td>65</td>
<td>77</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Grades

/*
 * Reads grades from an input file - Grades.txt
 * Each line contains Student No and 3 grades, Midterm 30%, Lab 30%, and Final 40%
 * Calculates Term Grades in Number
 * the output is placed in another file called GradesOut.txt
 */

#include <stdio.h> /* Defines fopen, fclose, fscanf, fprintf, and EOF */
#define MID_PERC 0.3 /* Midterm Percentage */
#define LAB_PERC 0.3 /* Lab Percentage */
#define FIN_PERC 0.4 /* Final Percentage */
#define MAX_STDNTS 100 /* Max Number of Students */

int main(void)
{
    FILE *gradesin, *gradesout; /* Input file pointer */
    /* Output file pointer */
    int Std[MAX_STD][5]; /* No, Mid, Lab, Fin, Grd */
    int ClsSz = 0; /* Size of Class */
    int ReadStat, i, j; /* Return value for fscanf */
    /* Loop indices */

    /* Open Files to be Used */
    gradesin = fopen("Grades.txt", "r");
    gradesout = fopen("GradesOut.txt", "w");

    /* Read First Line */
    ReadStat = fscanf(gradesin, "%d %d %d %d", &Std[0][0], &Std[0][1], &Std[0][2], &Std[0][3]);

    return 0;
}
Grades

while (ReadStat != EOF) {
    /* Exit when EOF is reached */

    /* Calculate End Term Grade */
    Std[ClsSz][4] = (int) ((Std[ClsSz][1] * MID_PERC + Std[ClsSz][2] * LAB_PERC + Std[ClsSz][3] * FIN_PERC) + 0.5);

    /* Read Next Line */
    ClsSz++;
    ReadStat = fscanf(gradesin, "%d %d %d %d", &Std[ClsSz][0], &Std[ClsSz][1], &Std[ClsSz][2], &Std[ClsSz][3]);
}

/* Print the output */
for (i = 0; i < ClsSz; i++) {
    for (j = 0; j < 4; j++) fprintf(gradesout, "%d\t", Std[i][j]);
    fprintf(gradesout, "%d\n", Std[i][4]);
}

Initialization of MDAs

- Multidimensional Arrays can be initialized when they are declared.
- The values are grouped by rows

```c
char Trngl[4][7] = { {' ', ' ', ' ', '*', ' ', ' ', ' '},
                      {' ', ' ', '*', ' ', '*', ' ', ' '},
                      {' ', '*', ' ', ' ', ' ', '*', ' '},
                      {'*', '*', '*', '*', '*', '*', '*'};
```