Use Arrays, learn subscripting of arrays, assignment, and examination of contents.
What Are Arrays?

• A compound type composed of many items (variables), all of the same type and organised in a consecutive (contiguous) set of memory locations.

• Like a row of boxes, each box may contain a different value, all boxes of the same type.
A 5 Element Array

- arr[0]
- arr[1]
- arr[2]
- arr[3]
- arr[4]

Computer memory

-45
6
0
123
-2047
Name of array (Note that all elements of this array have the same name, $c$)

<table>
<thead>
<tr>
<th>Position number of the element within array $c$</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c[0]$</td>
<td>-45</td>
</tr>
<tr>
<td>$c[1]$</td>
<td>6</td>
</tr>
<tr>
<td>$c[2]$</td>
<td>0</td>
</tr>
<tr>
<td>$c[3]$</td>
<td>72</td>
</tr>
<tr>
<td>$c[4]$</td>
<td>1543</td>
</tr>
<tr>
<td>$c[5]$</td>
<td>-89</td>
</tr>
<tr>
<td>$c[6]$</td>
<td>0</td>
</tr>
<tr>
<td>$c[7]$</td>
<td>62</td>
</tr>
<tr>
<td>$c[8]$</td>
<td>-3</td>
</tr>
<tr>
<td>$c[9]$</td>
<td>1</td>
</tr>
<tr>
<td>$c[10]$</td>
<td>6453</td>
</tr>
</tbody>
</table>
Array Syntax

code diagram:
Declaration Syntax

type variable identifier [ int expression ] ;
Sample Code - Array Initialisation I

```cpp
#include <iostream.h>
#include <iomanip.h>

int main()
{
    int i, n[10];
    for ( i = 0; i < 10; i++ )  // initialize array
        n[ i ] = 0;
    cout << "Element" << setw( 13 ) << "Value" << endl;
    for ( i = 0; i < 10; i++ )  // print array
        cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;
    return 0;
}
```
Sample Code - Array Initialisation

```cpp
#include <iostream.h>
#include <iomanip.h>

int main()
{
    int n[10] = {32, 27, 64, 18, 95, 14, 90, 70, 60, 37};

    cout << "Element" << setw(13) << "Value" << endl;
    for (int i = 0; i < 10; i++)
        cout << setw(7) << i << setw(13) << n[i] << endl;

    return 0;
}
```

Same thing achieved within the declaration of the array
Programming Tips

• Remember that the first element is subscripted at zero i.e. arr[0] is first and arr[1] is second!

• Initialise your arrays before using them

• `int arr[5] = {32, 27, 64, 18, 95, 14}` will produce a compile time error

• if you declare
  – `int arr[] = {32, 27, 64, 18, 95, 14}`
  – then the size of the array will be 6 i.e. arr[5] (yes!)
Placing Values in Arrays

// Initialize array s to the even integers from 2 to 20. Fig. 4.5
#include <iostream.h>
#include <iomanip.h>
int main()
{
    const int arraySize = 10; // note here, const, cannot modify this later!
    int j, s[ arraySize ];
    for ( j = 0; j < arraySize; j++ ) // set the values
        s[ j ] = 2 + 2 * j;
    cout << "Element" << setw( 13 ) << "Value" << endl;
    for ( j = 0; j < arraySize; j++ ) // print the values
        cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;
    return 0;
}
# Getting at Values in Arrays

// Compute the sum of the elements of the array // Fig. 4.8
#include <iostream.h>
int main()
{
    const int arraySize = 12;
    int a[ arraySize ] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
    int total = 0;
    for ( int i = 0; i < arraySize ; i++ )
        total += a[ i ];
    cout << "Total of array element values is " << total << endl;
    return 0;
}
Figure this out!... Use Debugger

```cpp
#include <iostream.h>   // fig 4.9
#include <iomanip.h>
int main()
{
    const int responseSize = 40, frequencySize = 11;
    int responses[ responseSize ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10, 1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6, 5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };;
    int frequency[ frequencySize ] = { 0 };
    for ( int answer = 0; answer < responseSize; answer++ )
        ++frequency[ responses[answer] ];
    cout << "Rating" << setw( 17 ) << "Frequency" << endl;
    for ( int rating = 1; rating < frequencySize; rating++ )
        cout << setw( 6 ) << rating
            << setw( 17 ) << frequency[ rating ] << endl;
    return 0;
}
See pp249, 250
1 // Fig. 4_12: fig04_12.cpp
2 // Treating character arrays as strings
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 int main()
10 {
11    char string1[20], string2[] = "string literal";
12
13    cout << "Enter a string: ";
14    cin >> string1;
15    cout << "string1 is: " << string1
16         << "\nstring2 is: " << string2
17         << "\nstring1 with spaces between characters is:\n";
18
19    for (int i = 0; string1[i] != '\0'; i++)
20       cout << string1[i] << ' ';
21
22    cin >> string1; // reads "there"
23    cout << "\nstring1 is: " << string1 << endl;
24
25    cout << endl;
26    return 0;
27 }

Enter a string: Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
Hello
string1 is: there

- 1. Initialize strings
- 2. Print strings
  - 2.1 Define loop
  - 2.2 Print characters individually
- Program Output
More Programming Tips for Arrays

• Validate correctness of input values, computed values for array subscripts, ensure not out of bounds

• Referencing elements outside array bounds gives rise to serious effects (crash !)

• Never reference below 0 and never reference greater than one less than max. subscript.

• Ensure loop terminating conditions obey rules
Multi-dimensional Arrays

• So far we have used an array as a linear collection of things of the same type, like rows or columns

• May now use arrays which have both rows and columns.

• These may be 2D, 3D or nD arrays

• Month is a 2D array, while week is a 1D array
## Example Multi-dimensional Array

<table>
<thead>
<tr>
<th>Row</th>
<th>Column 0</th>
<th>Column 1</th>
<th>Column 2</th>
</tr>
</thead>
</table>
| Row 0| Arr[0][0]| Arr[0][1]| Arr[0][2]| *(Array Name)*
| Row 1| Arr[1][0]| Arr[1][1]| Arr[1][2]| *(Row Subscript)*
# include <iostream.h>
int main()
{
    int myarr1[2][3]= {{1,2,3},{4,5,6}};
    int myarr2[2][3]= {{1,2,3}} ;
    int myarr3[2][3]= {{1}, {2,3}};
    int i,j;
    cout << "Values in array1 by row are:" << endl;
    for (i = 0; i < 2; i++ )
    {
        for (j = 0; j < 3; j++ )
            cout << myarr1[ i ][ j ] << ' ';  
        cout << endl;
    }

    cout << "Values in array2 by row are:" << endl;
    for (i = 0; i < 2; i++ )
    {
        for (j = 0; j < 3; j++ )
            cout << myarr2[ i ][ j ] << ' '; 
        cout << endl;
    }

    cout << "Values in array3 by row are:" << endl;
    for (i = 0; i < 2; i++ )
    {
        for (j = 0; j < 3; j++ )
            cout << myarr3[ i ][ j ] << ' '; 
        cout << endl;
    }
    return 0;
}

See 2Darr.cpp
Array: Exercises

• Build an array (1D), dimension 15, of integers in the range of 0..20
  – print a bar chart with a series of ‘+’ to represent the contents of the array
    • +++ is 3, +++++++ is 7

• Build an array of char and reverse it
  – do it using 2 arrays, then do it using 1 array

• Check arrays of char for palindromes

• NB char arrays may be null ‘\0’ terminated
Array: Graded Exercise

• Write a program to play knots & crosses, the program should be written in stages
  • print out the contents of a 2D array (3 x 3)
  • select a point (tell a user to select column, row)
  • place a 1 or a 0 in that position
  • print the contents again to view the board
  • check for a win (row, column, diagonal)
  • stop playing if board full or a win occurs
  • go to step 2
Functions with Array Parameters

Pass by Value
Pass by Reference
Notes

• Section 4.5 in Deitel & Deitel

• Prototype declaration appears as follows
  – \texttt{void modifyArray( int [], int );}

• The declaration has an array of \texttt{int} and one \texttt{int}

• Function call as follows
  – \texttt{modifyArray( a, arraySize );}

• Only \texttt{name} of array (no “[ ]”) and then an \texttt{int} passed
1 // Fig. 4.14: fig04_14.cpp
2 // Passing arrays and individual array elements to functions
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 void modifyArray( int [], int );  // appears strange
13 void modifyElement( int );
14
15 int main()
16 {
17    const int arraySize = 5;
18    int i, a[ arraySize ] = { 0, 1, 2, 3, 4 };
19
20    cout << "Effects of passing entire array call-by-reference:" << "
\nThe values of the original array are:\n";
21    for ( i = 0; i < arraySize; i++ )
22        cout << setw( 3 ) << a[ i ];
23    cout << endl;
24
25 // array a passed call-by-reference
26 modifyArray( a, arraySize );
27
28    cout << "The values of the modified array are:\n";
29    cout << "0 1 2 3 4 

The values of the original array are:
0 1 2 3 4 

The values of the modified array are:
0 2 4 6 8
2.1 Modify an element (call by value)

3. Print changes.

Parameter names required in function definition

Effects of passing array element call-by-value:

The value of a[3] is 6

Value in modifyElement is 12

The value of a[3] is 6
Effects of passing entire array call-by-reference:

The values of the original array are:
    0  1  2  3  4
The values of the modified array are:
    0  2  4  6  8

Effects of passing array element call-by-value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
Notes on Example I

• Available on
  – g:\public\bstone\n
• Size of array should be passed, as function only
told where array starts, not where it ends.

• Default is pass by Reference as we are passing a
pointer (memory location) variable

• Function has access to original array, so can
change values.

• Parameter names are optional on prototype.
Notes on Example II

• Program demonstrates passing an array Vs passing an individual element
• Passing the individual element and then operating on it does not modify the contents of the original array (pass by value)
• If you do not want to modify the contents of an array, then you must explicitly say so, see next program
// Demonstrating the const type qualifier
#include <iostream.h>
void tryToModifyArray( const int [] );
int main()
{
    int a[] = { 10, 20, 30 };
    tryToModifyArray( a );
    cout << a[ 0 ] << ' ' << a[ 1 ] << ' ' << a[ 2 ] << '
';
    return 0;
}
void tryToModifyArray( const int b[] )
{
    b[ 0 ] /= 2;    // error
    b[ 1 ] /= 2;    // error
    b[ 2 ] /= 2;    // error
Notes on \texttt{const} Qualifier

- The \texttt{const} qualifier prevents programmer from changing the contents of the argument / variable
- With other parameter types, pass by value is the default, with arrays the opposite holds, default is pass by reference, programmer must stipulate pass by value explicitly.
Self Study Exercise

- Check out bubble sort in Ex. 4.16
- Check out statistics example in Ex. 4.17
- Check out search algorithms in Ex. 4.19 and Ex. 4.20
- Make sure that you understand all constructs and code. Ask your tutor any questions that you have, bring them to tutorial.
W4.2

The `struct` User Defined Data Type
**struct** Definitions

- Aggregate data types built using elements of other types.
- Deitel & Deitel jump straight into OO, we will linger...

```c
struct Time {   // structure definition
    int hour;   // 0-23
    int minute; // 0-59
    int second; // 0-59
};
```
struct Syntax

struct definition

struct identifier {
    variable declaration;
}

struct member variable

struct identifier.

variable
**struct Example**

- **Keyword** *struct* introduces definition
- **Time** tag establishes *Time* as a new type within the language
- Members declared within braces
- Three *int* declared within *struct*

```c
struct Time {    // structure definition
    int hour;  // 0-23
    int minute; // 0-59
    int second; // 0-59
};
```
Notes on Declaration

- Definition of **Time** contains three variables, may be of any type, must be uniquely named within this **struct**

- Other **structs** may have same names within **struct** but must have unique **identifiers**.

- No space is reserved in memory until an **instance** of type **Time** is created.
Accessing Members of Structures

• Member access operators:
  – Dot operator ( . ) for structures
  – Arrow operator ( -> ) for pointers
  – Print member hour of timeStruct:
    
    ```
    cout << timeStruct.hour;
    ```
    OR
    ```
    timePtr = &timeStruct;
    cout << timePtr->hour;
    ```
  – timePtr->hour is the same as ( *timePtr ).hour
  – Parentheses required: * has lower precedence than .
struct

Example Code

#include <iostream.h> // Fig. 6.1: fig06_01.cpp
#include <iostream.h> // Fig. 6.1: fig06_01.cpp

struct Time {    // structure definition
    int hour;   // 0-23
    int minute; // 0-59
    int second; // 0-59
};

void printMilitary( const Time & ); // prototype
void printStandard( const Time & ); // prototype

int main()
{
    Time dinnerTime;     // variable type Time
dinnerTime.hour = 18;
dinnerTime.minute = 30;
dinnerTime.second = 0;
cout << "Dinner will be held at ";
printMilitary( dinnerTime );
cout << " military time,\nwhich is ";
printStandard( dinnerTime );
cout << " standard time.\n";
dinnerTime.hour = 29; // invalid values
dinnerTime.minute = 73;
cout << "\nTime with invalid values: ";
printMilitary( dinnerTime );
cout << endl;
return 0;
}
// Print the time in military format
void printMilitary( const Time &t )
{
    cout << ( t.hour < 10 ? "0" : "" ) << t.hour << ":" 
        << ( t.minute < 10 ? "0" : "" ) << t.minute;
}

// Print the time in standard format
void printStandard( const Time &t )
{
    cout << ( ( t.hour == 0 || t.hour == 12 ) ? 
             12 : t.hour % 12 ) 
          << ":" << ( t.minute < 10 ? "0" : "" ) << t.minute 
          << ":" << ( t.second < 10 ? "0" : "" ) << t.second 
          << ( t.hour < 12 ? " AM" : " PM" );
}
Notes on Example I

- Time has three members
  - int hour
  - int minute
  - int second

- Dot operator (.) used, initialise values to 18:30:00

- Pass by reference (&) used, so overhead of copying individual elements avoided (runs faster).

- **const** also used so that functions will not alter parameters.
Notes on Example II

• Initialisation can be accomplished as with an array as follows…..
  – `Time dinnerTime = {12, 20, 10};`

• Programmer must explicitly assign, read and handle in every way elements of a struct.
**struct Within struct**

- Because a struct is an extension to the C++ types, it is allowed to embed a `struct` inside another `struct` as though it were an `int` or a `float`.
- See example code on public directory.

```cpp
struct Date {
    int Year;
    int Month;
    int Day;
};

struct Employee {
    char Fname[20];
    char Sname[20];
    Date Birth;
    Date Start;
};
```
Another Example

- Students sit examinations and record a series of results
- Struct contains many data types, as well as other structs
- See Public directory for student.cpp
Arrays of `struct`

- Because we have established a new type, we can use this type as a building block for arrays
  
  ```
  - Time DailySchedule[3];
  - DailySchedule[0].hour = 9;
  - DailySchedule[0].minute = 0;
  - DailySchedule[0].second = 0;
  ```

- Modify the code provided to handle all times within an array or three `structs` used to store mealtimes
struct In Memory

- Memory locations are contiguous for a single struct
**struct** As Function Parameters

- In the example, a struct was passed as a single parameter (by reference)
  - `void printStandard( const Time & );` // prototype

- It is as though `Time` were just a normal *type* of the language, indeed it is now!

```cpp
void printStandard( const Time & t )
{
    cout << ( ( t.hour == 0 || t.hour == 12 ) ?
                 12 : t.hour % 12 )
       << " : " << ( t.minute < 10 ? "0" : "" ) << t.minute
       << " : " << ( t.second < 10 ? "0" : "" ) << t.second
       << ( t.hour < 12 ? " AM" : " PM" );
}
```
Graded Exercises

• Not graded
  – See examples on my public directory
  – structex.cpp, struct2.cpp, struct3.cpp

• Graded
  – Rewrite the code in student.cpp and use an array of 3 students. Break the code up into sensible functions using “pass by value” or “pass by reference” as appropriate. Show off a little, but justify your design!