DASF004
Basic and Practice in Programming

Lecture 6

Array

Administration

Mid-term Exam

- Will be in the form of a big assignment
- Will cover most of the topics we covered, up to Lab 6
- Will be distributed on (Wednesday) 14th April 2020, 12:00 pm
- Due date: (Thursday) 15th April 2020, 18:00 pm
- You have about 30 hours to complete it
- No late submission
- Submit it on iCampus before the deadline
- Further information may also be posted on iCampus
- ***If you submit late, it will not be graded and you will receive a zero***

List of Topics in Mid Term Exam

1. Introduction
- Algorithm and Pseudo Code
- Compiling
- Syntax and semantic

2. Sequence and Control
- Variables
  - Declaration
  - Variable Type and Type Casting
  - Arithmetics, Assignment and Increment/Decrement
  - `if … else if` statement

3. Sequence and Control 2
- Repetition
  - `while` loop
  - `for` loop
  - `do … while` loop
- `switch` statement
- `break` and `continue` statement
- Nested loops

List of Topics in Mid Term Exam

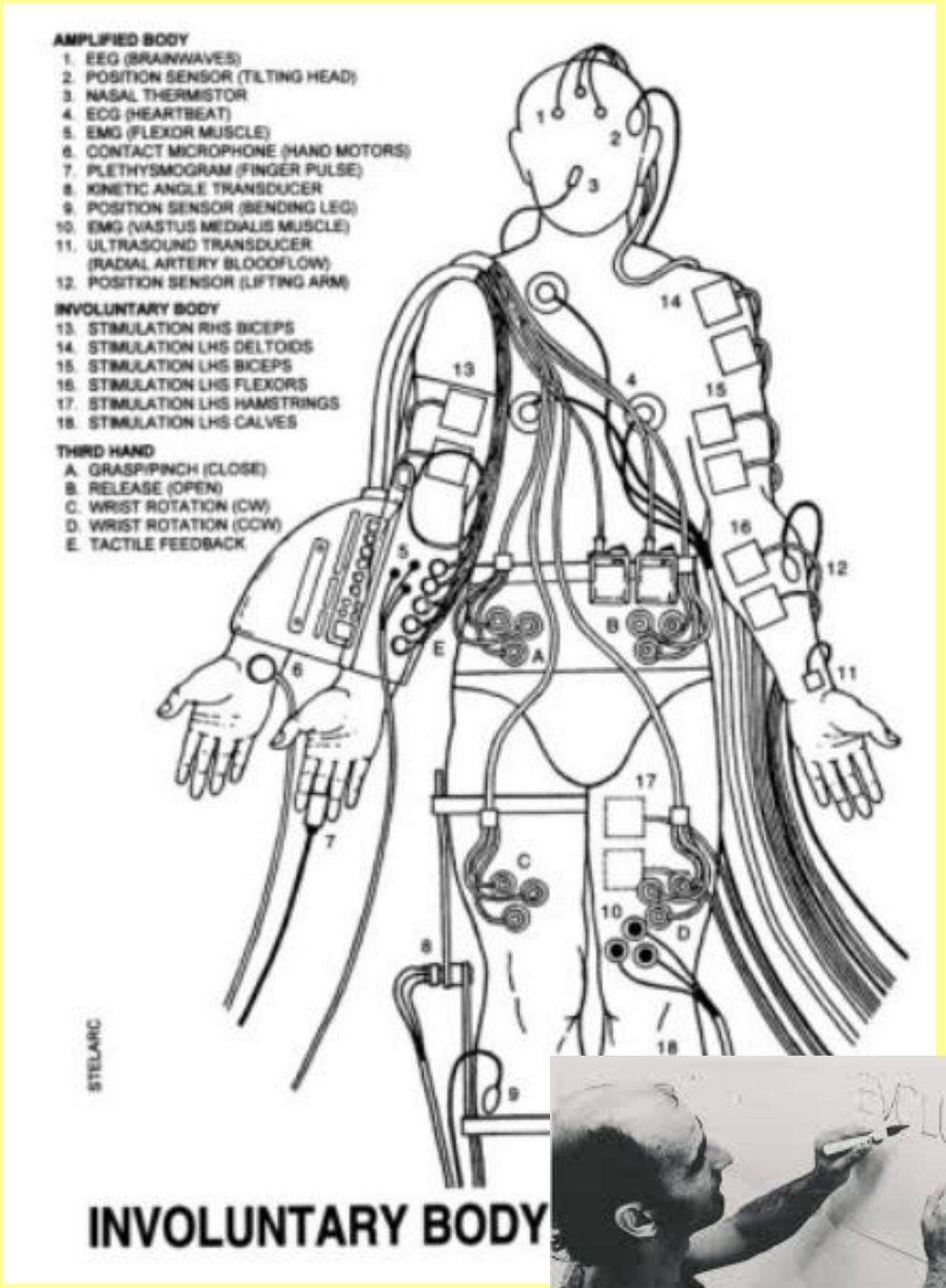
4. Function
- Function definition, Function Prototype, Function Calling
- Scope of Variables
- `Global` Variables and `Static` Variable

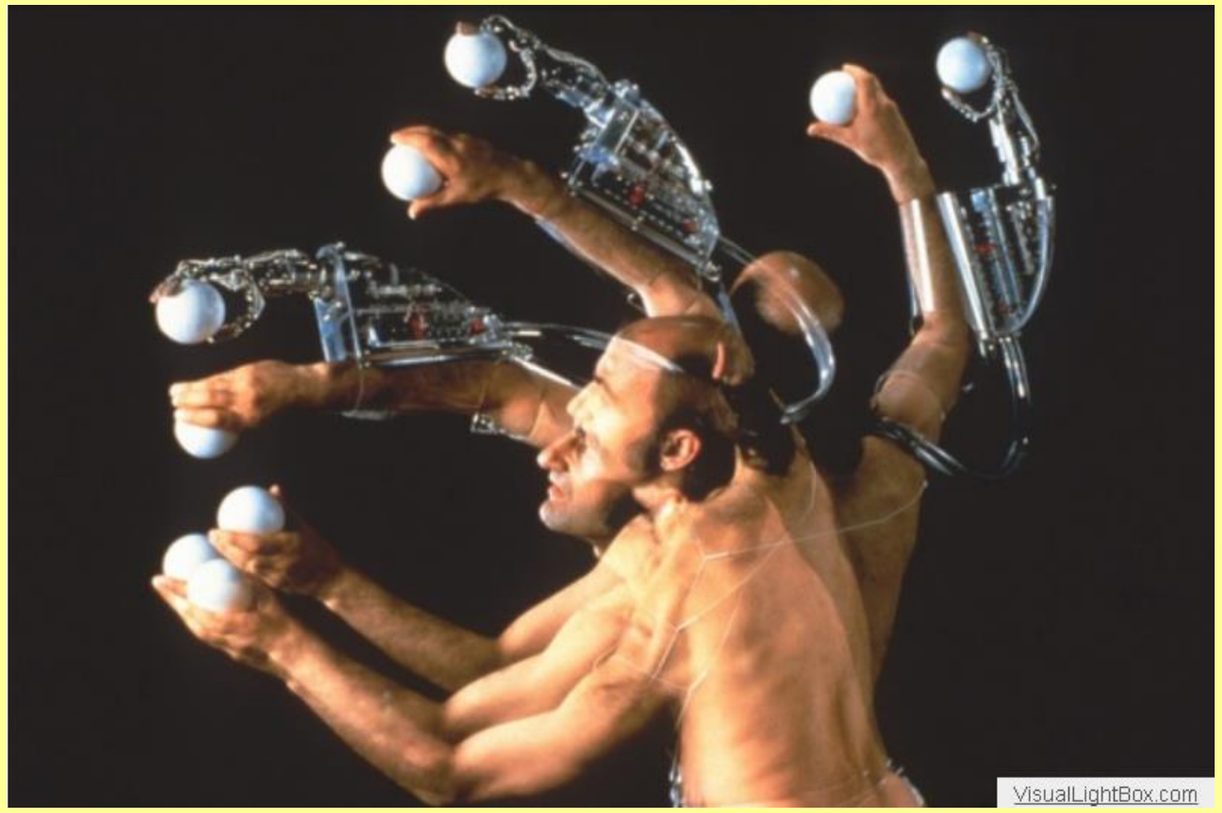
5. Function
- Argument Coercion
- Recursive Function
- Generating Random Value

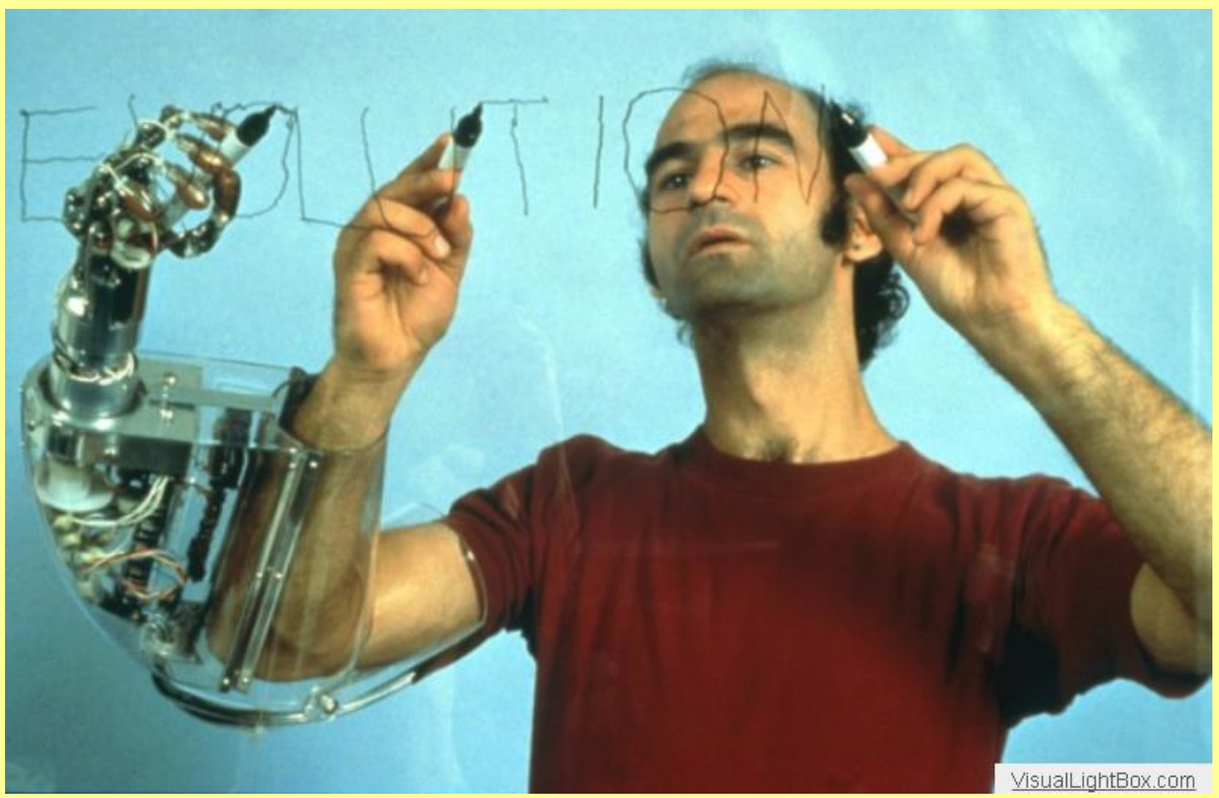
6. Array
- `#define` preprocessor directive
- Array
  - Declaration
  - Initialization
  - Character String
  - Multi-dimensional Array
  - Passing Array to Functions

Stelarc: An Australian Performance Artist

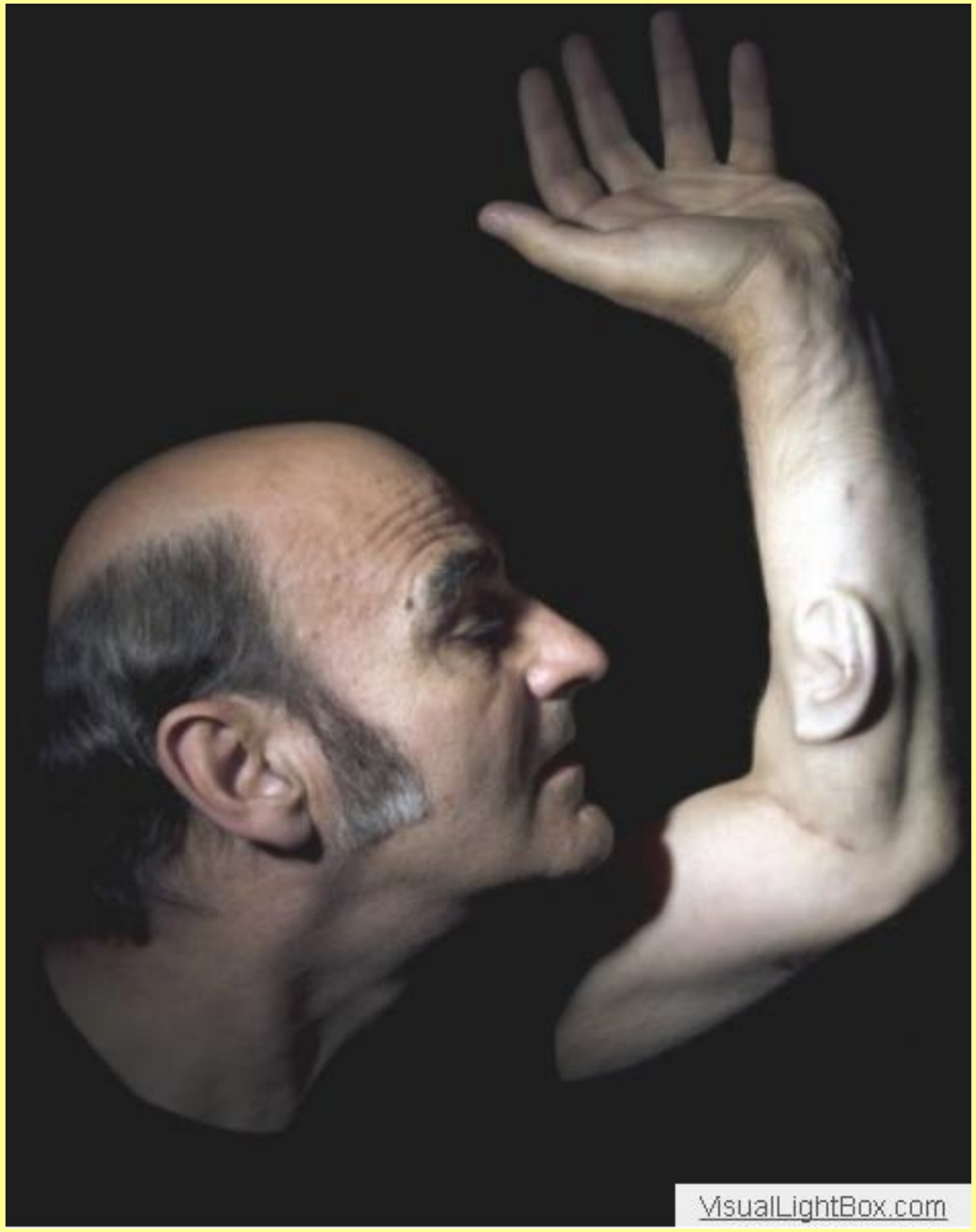
- stelarc.org







How about a third ear?



Q&A

```
#include <stdio.h>

int main(void)
{
   unsigned int counter = 1;

   while (counter++ <= 10)
   {
      printf("%u\n", counter);
   }

   printf("%u\n", counter); //12

   return 0;
}
```

Fig. 4.1

Agenda

`#define` preprocessor directive

Introduction to Data Structure

Array

- Declaration
- Initialization
- Character String
- Multi-dimensional Array
- Passing Array to Functions

`#define` Preprocessor Directive

- The #define preprocessor directive can be used to define a constant, for example:

```
#include <stdio.h>
#define PI 3.1416

int main(void)
{ int radius = 10;
  float area = PI * radius * radius;  // Area = PI x R x R
  printf("Area: %f\n",area);
}
```

- Another example:

```
#include <stdio.h>
#define CLASS_SIZE 80

int main(void)
{ int sum = 0;
  for(int i=1; i<=CLASS_SIZE; i++)
  { int score;
    printf("Enter score for student %d: \n",i);
    scanf("%d",&score);
    sum = sum + score;
  }
  printf("Class Average: %f\n", (float) sum/CLASS_SIZE);
}
```

Data Structure

- Data Structure
  - To organize data into a way such that it can be used more efficiently

- For example:
  - How do you store these values into variables:

	Math	Phy	Bio	Chem	Eng
John	80	75	96	78	69
Mary	90	74	78	84	98
Joe	86	79	78	78	74
Peter	94	82	87	81	67
Paul	94	82	91	83	72
Jane	89	87	76	85	95

```
int JohnMath = 80;    int MaryMath = 90;
int JohnPhy = 75;     int MaryPhy = 74;
int JoeMath = 86;     int JoePhy = 79;
…
```

- Is there a better way to organize the variables?
- What if the programmer don’t know how many students in the class?

Array

- Arrays are data structures consisting of related data items of the same type.

- An array is a group of *contiguous* memory locations that all have the ***same type***.
  - For example, an array of integer, is a group of integers

- We can declare an array of 5 integers as follow:

```
int JohnScore[5];
JohnScore[0] = 80;
JohnScore[1] = 75;
JohnScore[2] = 96;
JohnScore[3] = 78;
JohnScore[4] = 69;
```

- For an array with size `n`, The **index number** starts with `0`, and ends with `n-1`

Defining arrays

- Arrays occupy space in memory
  - *contiguous* memory locations
- You specify the type of each element and the number of elements each array requires so that the computer may reserve the appropriate amount of memory.
- The following definition reserves 12 elements for integer array `c`, which has subscripts in the range 0-11.
  - `int c[ 12 ];`

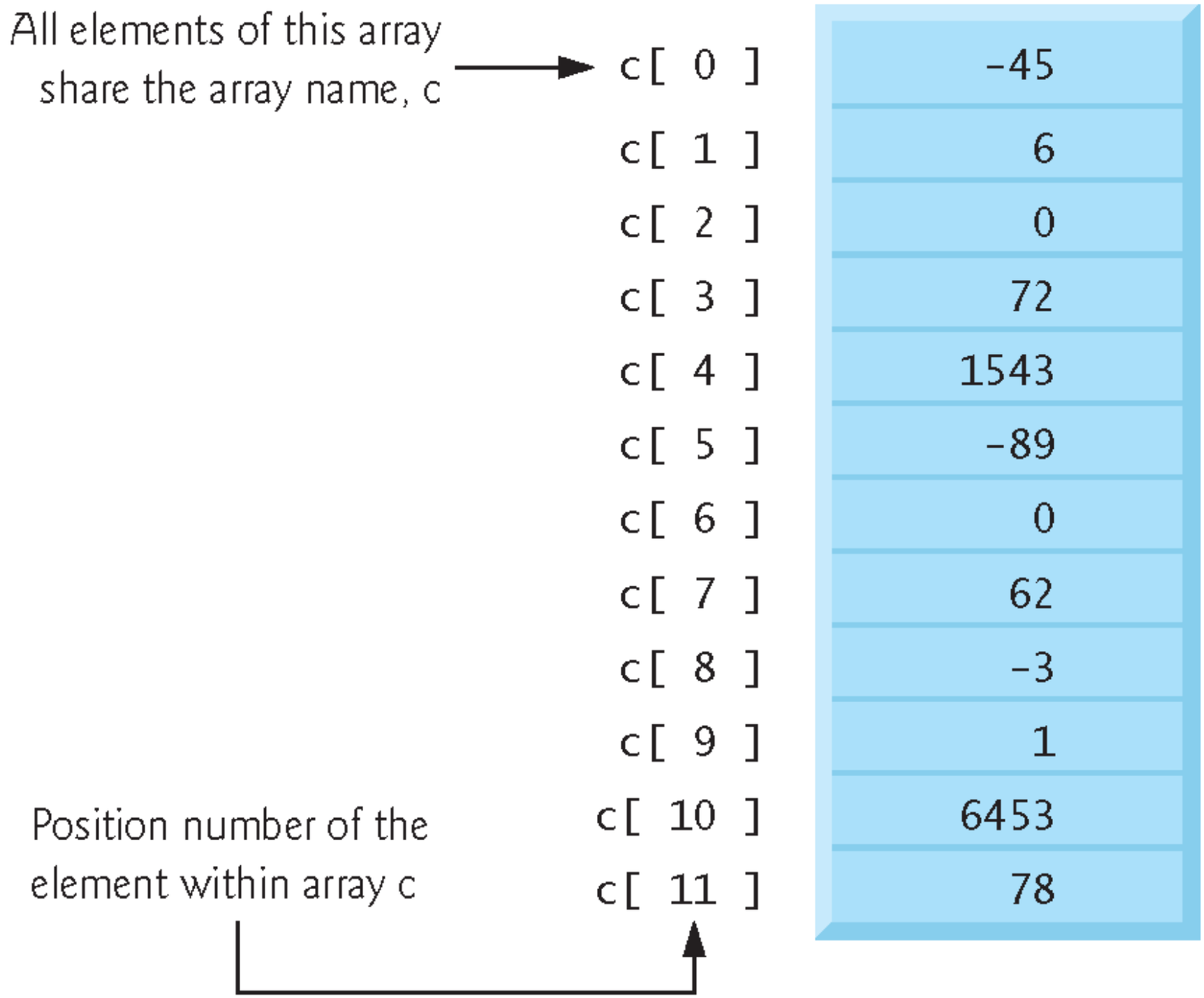


Fig. 6.1 | 12-element array.

```c
// Fig. 6.3: fig06_03.c
// Initializing the elements of an array to zeros.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   int n[ 10 ]; // n is an array of 10 integers
   size_t i; // counter

   // initialize elements of array n to 0
   for ( i = 0; i < 10; ++i ) {
      n[ i ] = 0; // set element at location i to 0
   } // end for

   printf( "%s%13s\n", "Element", "Value" );

   // output contents of array n in tabular format
   for ( i = 0; i < 10; ++i ) {
      printf( "%7u%13d\n", i, n[ i ] );
   } // end for
} // end main
```

Fig. 6.3 | Initializing the elements of an array to zeros. (Part 1 of 2.)

```
Element         Value
      0             0
      1             0
      2             0
      3             0
      4             0
      5             0
      6             0
      7             0
      8             0
      9             0
```

Fig. 6.3 | Initializing the elements of an array to zeros. (Part 2 of 2.)

```c
// Fig. 6.4: fig06_04.c
// Initializing the elements of an array with an initializer list.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   // use initializer list to initialize array n
   int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
   size_t i; // counter

   printf( "%s%13s\n", "Element", "Value" );

   // output contents of array in tabular format
   for ( i = 0; i < 10; ++i ) {
      printf( "%7u%13d\n", i, n[ i ] );
   } // end for
} // end main
```

Fig. 6.4 | Initializing the elements of an array with an initializer list. (Part 1 of 2.)

```
Element         Value
      0            32
      1            27
      2            64
      3            18
      4            95
      5            14
      6            90
      7            70
      8            60
      9            37
```

Fig. 6.4 | Initializing the elements of an array with an initializer list. (Part 2 of 2.)

Array examples

- If there are *fewer* initializers than elements in the array, the remaining elements are initialized to zero.
- For example, the elements of the array `n` in Fig. 6.3 could have been initialized to zero as follows:

```
// initializes entire array to zeros
int n[ 10 ] = { 0 };
```

- This *explicitly* initializes the first element to zero and initializes the remaining nine elements to zero because there are fewer initializers than there are elements in the array
- It's important to remember that **arrays are not automatically initialized to zero**.
- You must at least initialize the first element to zero for the remaining elements to be automatically zeroed. .

Array examples

The array definition

```
int n[ 5 ] = { 32, 27, 64, 18, 95, 14 };
```

causes a syntax error because there are six initializers and *only* five array elements.

If the array size is *omitted* from a definition with an initializer list, the number of elements in the array will be the number of elements in the initializer list.

For example,

```
int n[] = { 1, 2, 3, 4, 5 };
```

would create a five-element array initialized with the indicated values.

```c
// Fig. 6.5: fig06_05.c
// Initializing the elements of array s to the even integers from 2 to 20.
#include <stdio.h>
#define SIZE 10 // maximum size of array

// function main begins program execution
int main( void )
{
   // symbolic constant SIZE can be used to specify array size
   int s[ SIZE ]; // array s has SIZE elements
   size_t j; // counter

   for ( j = 0; j < SIZE; ++j ) { // set the values
      s[ j ] = 2 + 2 * j;
   } // end for

   printf( "%s%13s\n", "Element", "Value" );

   // output contents of array s in tabular format
   for ( j = 0; j < SIZE; ++j ) {
      printf( "%7u%13d\n", j, s[ j ] );
   } // end for
} // end main
```

Fig. 6.5 | Initialize the elements of array s to the even integers from 2 to 20. (Part 1 of 2.)

```
Element         Value
      0             2
      1             4
      2             6
      3             8
      4            10
      5            12
      6            14
      7            16
      8            18
      9            20
```

Fig. 6.5 | Initialize the elements of array s to the even integers from 2 to 20. (Part 2 of 2.)

Array examples

Summing the Elements of an Array

Figure 6.6 sums the values contained in the 12-element integer array `a`.

The `for` statement's body (line 16) does the totaling.

```c
// Fig. 6.6: fig06_06.c
// Computing the sum of the elements of an array.
#include <stdio.h>
#define SIZE 12

// function main begins program execution
int main( void )
{
   // use an initializer list to initialize the array
   int a[ SIZE ] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
   size_t i; // counter
   int total = 0; // sum of array

   // sum contents of array a
   for ( i = 0; i < SIZE; ++i ) {
      total += a[ i ];
   } // end for

   printf( "Total of array element values is %d\n", total );
} // end main
```

```
Total of array element values is 383
```

Fig. 6.6 | Computing the sum of the elements of an array.

Array examples

Using Arrays to Summarize Survey Results

Our next example uses arrays to summarize the results of data collected in a survey.

Consider the problem statement.

> Forty students were asked to rate the quality of the food in the student cafeteria on a scale of 1 to 10 (1 means awful and 10 means excellent). Place the 40 responses in an integer array and summarize the results of the poll.

This is a typical array application (see Fig. 6.7).

We wish to summarize the number of responses of each type (i.e., 1 through 10).

```c
// Fig. 6.7: fig06_07.c
// Analyzing a student poll.
#include <stdio.h>
#define RESPONSES_SIZE 40 // define array sizes
#define FREQUENCY_SIZE 11

// function main begins program execution
int main( void )
{
   size_t answer; // counter to loop through 40 responses
   size_t rating; // counter to loop through frequencies 1-10

   // initialize frequency counters to 0
   int frequency[ FREQUENCY_SIZE ] = { 0 };

   // place the survey responses in the responses array
   int responses[ RESPONSES_SIZE ] = { 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 };

```

Fig. 6.7 | Analyzing a student poll. (Part 1 of 3.)

```
21      // for each answer, select value of an element of array responses
22      // and use that value as subscript in array frequency to
23      // determine element to increment
24      for ( answer = 0; answer < RESPONSES_SIZE; ++answer ) {
25         ++frequency[ responses [ answer ] ];
26      } // end for
27
28      // display results
29      printf( "%s%17s\n", "Rating", "Frequency" );
30
31      // output the frequencies in a tabular format
32      for ( rating = 1; rating < FREQUENCY_SIZE; ++rating ) {
33         printf( "%6d%17d\n", rating, frequency[ rating ] );
34      } // end for
35   } // end main
```

Fig. 6.7 | Analyzing a student poll. (Part 2 of 3.)

```
Rating        Frequency
     1                2
     2                2
     3                2
     4                2
     5                5
     6               11
     7                5
     8                7
     9                1
    10                3
```

Fig. 6.7 | Analyzing a student poll. (Part 3 of 3.)

Array examples

Graphing Array Element Values with Histograms

Our next example (Fig. 6.8) reads numbers from an array and graphs the information in the form of a bar chart or histogram—each number is printed, then a bar consisting of that many asterisks is printed beside the number.

The nested `for` statement (line 20) draws the bars.

Note the use of `puts( "" )` to end each histogram bar (line 24).

```c
// Fig. 6.8: fig06_08.c
// Displaying a histogram.
#include <stdio.h>
#define SIZE 10

// function main begins program execution
int main( void )
{
   // use initializer list to initialize array n
   int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
   size_t i; // outer for counter for array elements
   int j; // inner for counter counts *s in each histogram bar

   printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );

   // for each element of array n, output a bar of the histogram
   for ( i = 0; i < SIZE; ++i ) {
      printf( "%7u%13d          ", i, n[ i ]) ;

      for ( j = 1; j <= n[ i ]; ++j ) { // print one bar
         printf( "%c", '*' );
      } // end inner for

      puts( "" ); // end a histogram bar
   } // end outer for
} // end main
```

```
Element         Value         Histogram
      0            19         ********************
      1             3         ***
      2            15         ***************
      3             7         *******
      4            11         ***********
      5             9         *********
      6            13         *************
      7             5         *****
      8            17         *****************
      9             1         *
```

Fig. 6.8 | Displaying a histogram. (Part 2 of 2.)

Character Array

```
char string1[15] = {“H”,”e”,”l”,”l”,”o”,”,”,” “,”W”,”o”,”r”,”l”,”d”,”!”,”\n”,”\0”};
char string1[] = {“H”,”e”,”l”,”l”,”o”,”,”,” “,”W”,”o”,”r”,”l”,”d”,”!”,”\n”,”\0”};
```

```
char string1[15] = "Hello, World!\n";
char string1[] = "Hello, World!\n";
```

- The character array `string1` contains 15 characters including the '\0' character which is automatically added by the compiler at the end of the string.

Character Array

Using Character Arrays to Store and Manipulate Strings

A character array can be initialized using a string literal.

For example,

```
char string1[] = "Hello, World!\n";
printf("%s",string1);
```

```c
// Fig. 6.10: fig06_10.c
// Treating character arrays as strings.
#include <stdio.h>
#define SIZE 20

// function main begins program execution
int main( void )
{
   char string1[ SIZE ]; // reserves 20 characters
   char string2[] = "string literal"; // reserves 15 characters
   size_t i; // counter

   // read string from user into array string1
   printf( "%s", "Enter a string (no longer than 19 characters): " );
   scanf( "%19s", string1 ); // input no more than 19 characters

   // output strings
   printf( "string1 is: %s\nstring2 is: %s\n"
           "string1 with spaces between characters is:\n",
           string1, string2 );

   // output characters until null character is reached
   for ( i = 0; i < SIZE && string1[ i ] != '\0'; ++i ) {
      printf( "%c ", string1[ i ] );
   } // end for

   puts( "" );
} // end main
```

```
Enter a string (no longer than 19 characters): Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```

Fig. 6.10 | Treating character arrays as strings. (Part 2 of 2.)

Multi-dimensional Arrays

You can declare arrays with multi-dimension, for example:

```
int StudentScore[3][3] = {{95, 92, 93},
                          {89, 98, 82},
                          {90, 87, 88}};
```

will declare a 3x3 array.

The index number of the above array is ranging from 0 to 2:

The value of `StudentScore[0][0]` is `95`.

The value of `StudentScore[0][1]` is `92`.

The value of `StudentScore[0][2]` is `93`.

The value of `StudentScore[1][0]` is `89`.

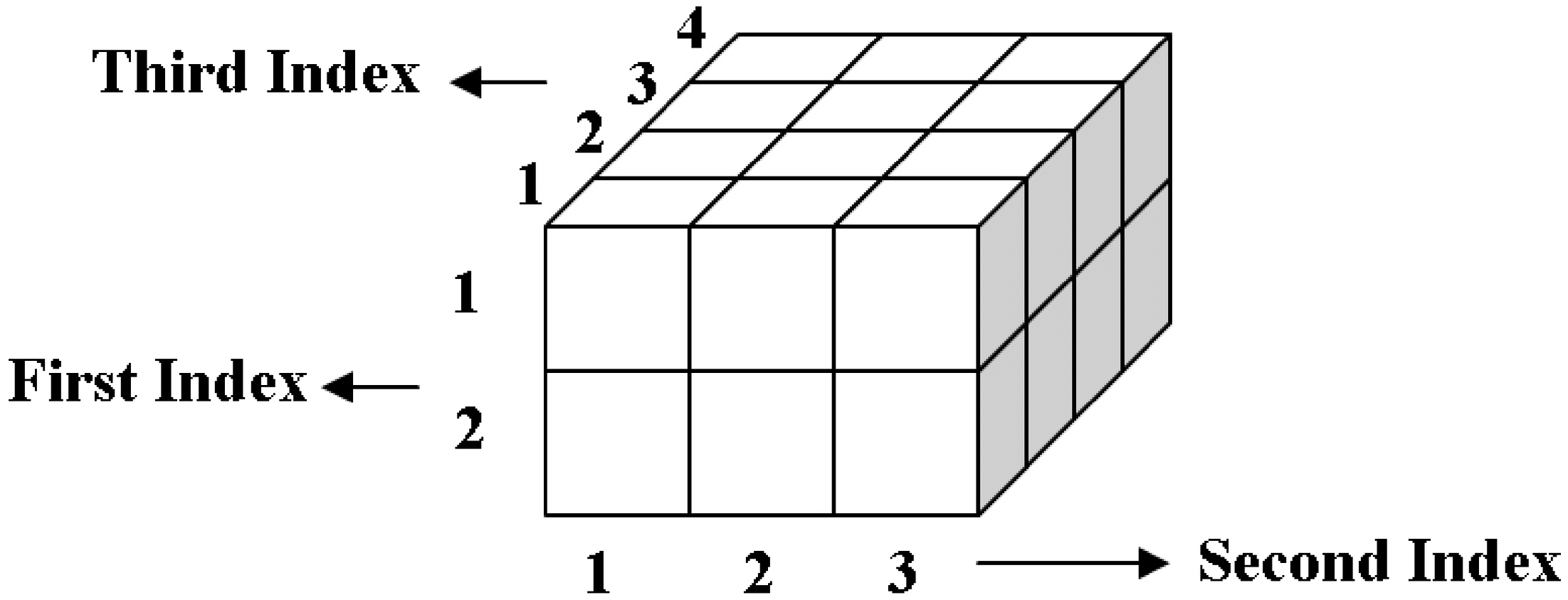
The value of `StudentScore[1][1]` is `98` ...

Multi-dimensional Arrays

Similarly, you can create array with higher dimension, for example:

```
int Array3D[2][2][2] = {
                                {
                                  {2, 3}, {3, 3},
                                },
                                  {
                                    {4, 3}, {3, 2}
                                  },
                               };
```

Be careful about matching the braces of multidimensional arrays.



Three-dimensional array with twenty four elements

Multi-dimensional Arrays

You can initialize the entire multidimensional array with the value 0 as follow:

```
int Array2D[3][3] = {0};
```

Declaration of multi-dimensional array must have bound. The following declaration will produce compiling error:

```
int Array1[3][];
int Array2[][];
```

Declaring Array with Length 0

```
int x[0];
```

Allowed in some compiler (Dev C++)

Disallowed in some compiler (Visual C++)

Declaring an Array Based on User Input

```
#include <stdio.h>

int main(void)
{ int ClassSize;

  printf("How many students are there in the class: ");
  scanf("%d",&ClassSize); // prompt user to enter class size

  int score[ClassSize];   // create an array based on class size

  for(int i=0;i<ClassSize;i++) // a loop for enter each score
  { printf("Enter score for student %d:",i+1);
    scanf("%d",&score[i]);
  }

  for(int i=0;i<ClassSize;i++) // a loop to display each score
  { printf("Student %d Score: %d\n",i+1,score[i]);
  }

  return 0;
}
```

C:\X\PortableApps\Dev-Cpp32\ConsolePauser.exe

```
How many students are there in the class: 6
Enter score for student 1:12
Enter score for student 2:23
Enter score for student 3:34
Enter score for student 4:45
Enter score for student 5:56
Enter score for student 6:67
Student 1 Score: 12
Student 2 Score: 23
Student 3 Score: 34
Student 4 Score: 45
Student 5 Score: 56
Student 6 Score: 67


Process exited normally.
Press any key to continue . . .
```

Declaring Array with size unknown

```
int x;
printf(“Input: “);
scanf(“%d”,&x);
Int ArrayX[x];
```

Allowed in some compiler (Dev C++)

Disallowed in some compiler (Visual C++)

Memory allocation

- The malloc operator (C Lang)
- The new operator (C++)
- Advance pointer operations

Array size cannot be changed after declaration

```
int y=5;
int x[y] = {1,2,3,4,5);
y=10;
```

Array is pointer

```
int x[5] = {1,2,3,4,5};
```

`x` is a pointer.

Value of `x` is an address.

Passing arrays to functions

You can pass an individual value in an array to a function as argument:

```
int score[4] = {1,2,3,4};
printf("score[2]: %i",score[2]);
```

This code segment would generate output:

```
score[2]: 3
```

When you pass an array to a function, its address is being pass, for example:

```
int score[4] = {1,2,3,4};

printf("Array: %p",score);
```

This code segment would generate output similar to:

```
Array: 0012ff78
```

`0012ff78` is the address location of the array `score`.

Note: The `%p` conversion specifier is for printing out pointer address in printf function

Passing arrays to functions

Addresses are represented in Hexadecimal value (base 16).

Most of the time, this address value is meaningless to you.

But when you see output similar to this in your program, you know that it is likely you are making a mistake in your program (i.e. you're printing out the address instead of the value).

Q&A?