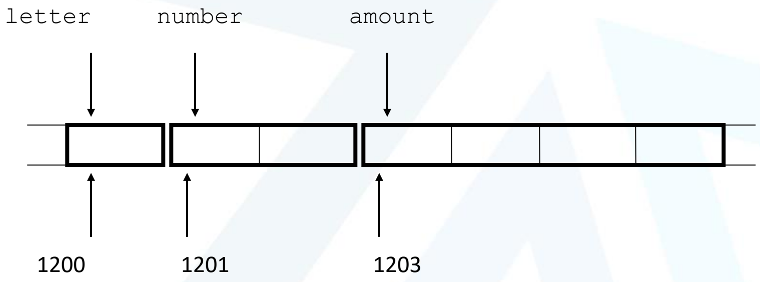
Chapter 9 – Pointers

9.1 Getting the address of a Variable

- The address operator (&) returns the memory address of a variable.

Figure 9-1



Program 9-1

```
// This program uses the & operator to determine a variable’s
// address and the sizeof operator to determine its size.

#include <iostream.h>

void main(void)
{
 int x = 25;
 cout << "The address of x is " << &x << endl;
 cout << "The size of x is " << sizeof(x) << " bytes\n";
 cout << "The value in x is " << x << endl;
}
```

Program Output

```
The address of x is 0x8f05
The size of x is 2 bytes
The value in x is 25
```

Pointer Variables

- Pointer variables, which are often just called pointers, are designed to hold memory addresses. With pointer variables you can indirectly manipulate data stored in other variables.

Pointers are useful for the following:

- Working with memory locations that regular variables don’t give you access to
- Working with strings and arrays
- Creating new variables in memory while the program is running
- Creating arbitrarily-sized lists of values in memory

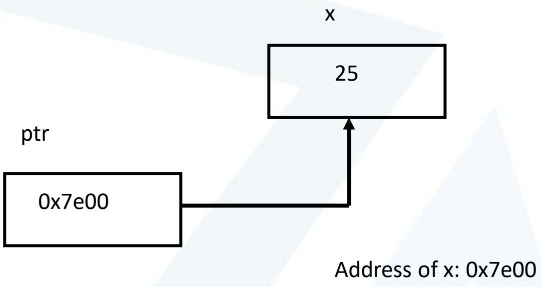
Program 9-2

```
// This program stores the address of a variable in a pointer.
#include <iostream.h>

void main(void)
{
 int x = 25;
 int *ptr;

 ptr = &x;   // Store the address of x in ptr
 cout << "The value in x is " << x << endl;
 cout << "The address of x is " << ptr << endl;
}
```

Program Output

```
The value in x is 25
The address of x is 0x7e00
```

Figure 9-2



Program 9-3

```
// This program demonstrates the use of the indirection
// operator.
#include <iostream.h>

void main(void)
{
 int x = 25;
 int *ptr;

 ptr = &x;   // Store the address of x in ptr
 cout << "Here is the value in x, printed twice:\n";
 cout << x << "  " << *ptr << endl;
 *ptr = 100;
 cout << "Once again, here is the value in x:\n";
 cout << x << "  " << *ptr << endl;
}
```

Program Output

```
Here is the value in x, printed twice:
25  25
Once again, here is the value in x:
100  100
```

Program 9-4

```
#include <iostream>

void main(void)
{
 int x = 25, y = 50, z = 75;
 int *ptr;
 cout << "Here are the values of x, y, and z:\n";
 cout << x << "   " << y << "   " << z << endl;
 ptr = &x;  // Store the address of x in ptr
 *ptr *= 2; // Multiply value in x by 2
 ptr = &y;  // Store the address of y in ptr
 *ptr *= 2;  // Multiply value in y by 2
 ptr = &z;   // Store the address of z in ptr
 *ptr *= 2;  // Multiply value in z by 2
 cout << "Once again, here are the values of x, y, and z:\n";
 cout << x << "   " << y << "   " << z << endl;
}
```

Program Output

```
Here are the values of x, y, and z:
25  50  75
Once again, here are the values of x, y , and z:
50  100  150
```

9.3 Relationship Between Arrays and Pointers

- array names can be used as pointers, and vice-versa.

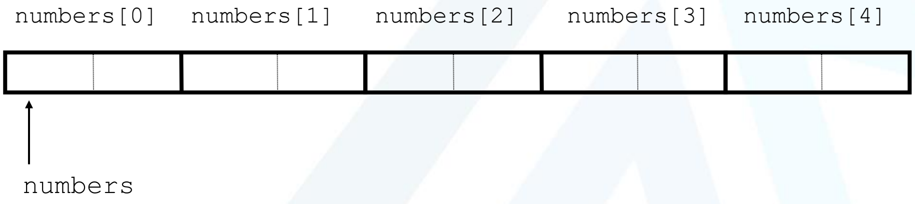
Program 9-5

```
// This program shows an array name being dereferenced
// with the * operator.

#include <iostream.h>

void main(void)
{
 short numbers[] = {10, 20, 30, 40, 50};

 cout << "The first element of the array is ";
 cout << *numbers << endl;
}
```

Program Output

```
The first element in the array is 10
```

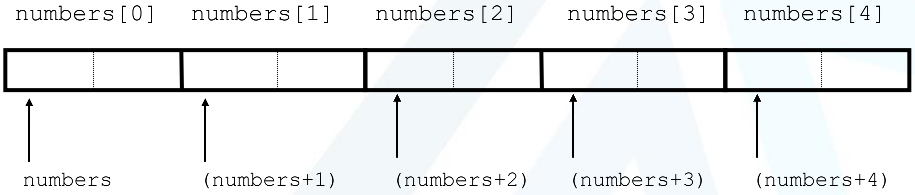
Figure 9-3



numbers[0] numbers[1] numbers[2] numbers[3] numbers[4]

numbers

Figure 9-4



numbers[0] numbers[1] numbers[2] numbers[3] numbers[4]

numbers (numbers+1) (numbers+2) (numbers+3) (numbers+4)

Program 9-6

```
// This program processes the contents of an array. Pointer
// notation is used.
#include <iostream.h>

void main(void)
{
 int numbers[5];

 cout << "Enter five numbers: ";
 for (int count = 0; count < 5; count++)
     cin >> *(numbers + count);
 cout << "Here are the numbers you entered:\n";
 for (int count = 0; count < 5; count++)
     cout << *(numbers + count)<< " ";
 cout << endl;
}
```

Program Output with Example Input

```
Enter five numbers: 5 10 15 20 25 [Enter]
Here are the numbers you entered:
5 10 15 20 25
```

Program 9-7

```
// This program uses subscript notation with a pointer and
// pointer notation with an array name.

#include <iostream.h>

void main(void)
{
 float coins[5] = {0.05, 0.1, 0.25, 0.5, 1.0};
 float *floatPtr;  // Pointer to a float
 int count;        // array index

 floatPtr = coins; // floatPtr now points to coins array
 cout.precision(2);
 cout << "Here are the values in the coins array:\n";
```

Program continues

```
 for (count = 0; count < 5; count++)
    cout << floatPtr[count] << " ";
 cout << "\nAnd here they are again:\n";
 for (count = 0; count < 5; count++)
    cout << *(coins + count) << " ";
 cout << endl;
}
```

Program Output

```
Here are the values in the coins array:
0.05 0.1 0.25 0.5 1
And here they are again:
0.05 0.1 0.25 0.5 1
```

Program 9-8

```
// This program uses the address of each element in
 the array.

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

void main(void)
{
 float coins[5] = {0.05, 0.1, 0.25, 0.5, 1.0};
 float *floatPtr; // Pointer to a float
 int count;       // array index
 cout.precision(2);
 cout << "Here are the values in the coins array:\n";
```

Program continues

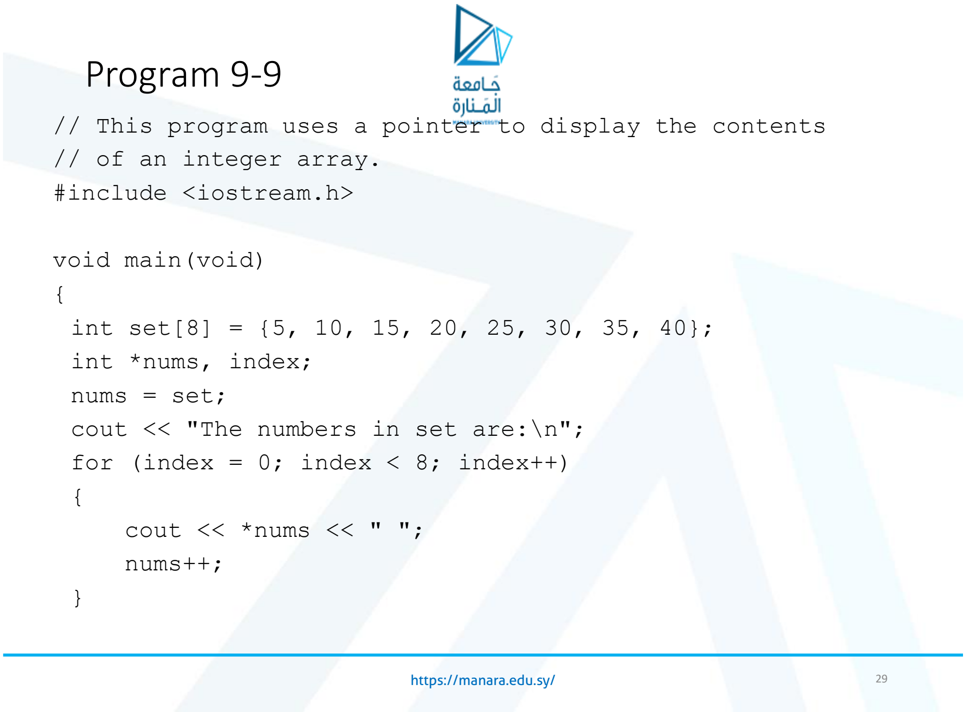
```
  for (count = 0; count < 5; count++)
  {
      floatPtr = &coins[count];
      cout << *floatPtr << " ";
  }
  cout << endl;
 }
```

Program Output

```
Here are the values in the coins array:
0.05 0.1 0.25 0.5 1
```

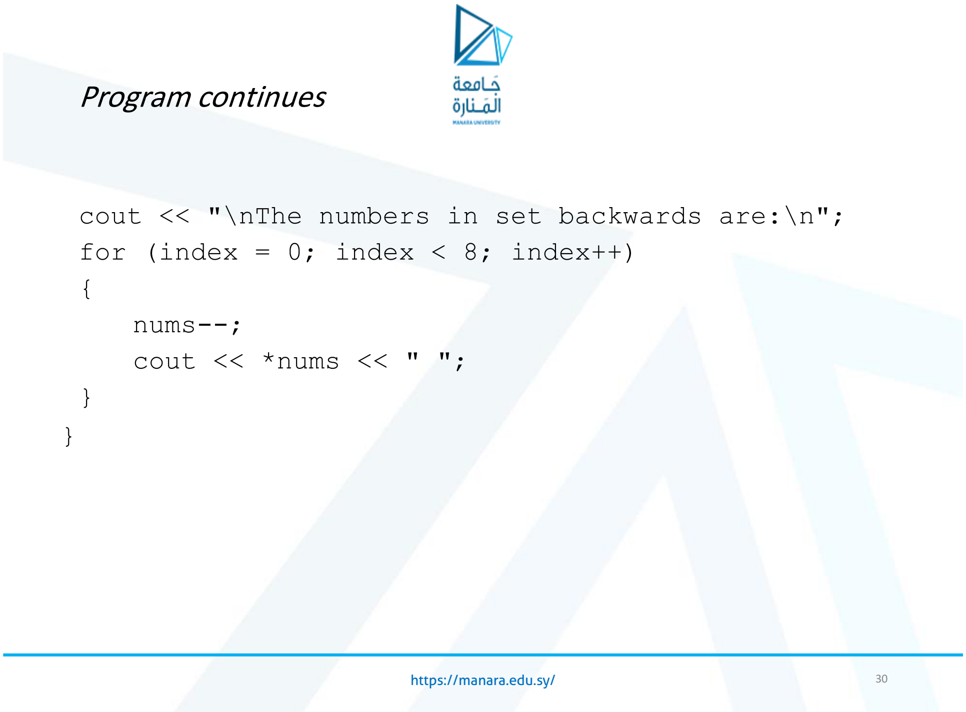
9.4 Pointer Arithmetic

- Some mathematical operations may be performed on pointers.
  - The ++ and – operators may be used to increment or decrement a pointer variable.
  - An integer may be added to or subtracted from a pointer variable. This may be performed with the +, - +=, or -= operators.
  - A pointer may be subtracted from another pointer.



Program 9-9

```
// This program uses a pointer to display the contents
// of an integer array.
#include <iostream.h>

void main(void)
{
 int set[8] = {5, 10, 15, 20, 25, 30, 35, 40};
 int *nums, index;
 nums = set;
 cout << "The numbers in set are:\n";
 for (index = 0; index < 8; index++)
 {
     cout << *nums << " ";
     nums++;
 }
```



Program continues

```
 cout << "\nThe numbers in set backwards are:\n";
 for (index = 0; index < 8; index++)
 {
     nums--;
     cout << *nums << " ";
 }
}
```

Program Output

```
The numbers in set are:
5 10 15 20 25 30 35 40
The numbers in set backwards are:
40 35 30 25 20 15 10 5
```

9.5 Initializing Pointers

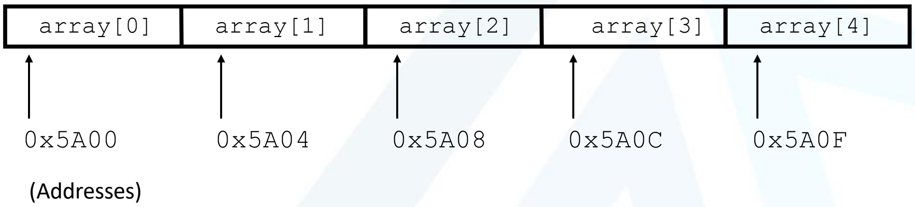
- Pointers may be initialized with the address of an existing object.

9.6 Comparing Pointers

- If one address comes before another address in memory, the first address is considered "less than" the second. C++'s relational operators maybe used to compare pointer values.

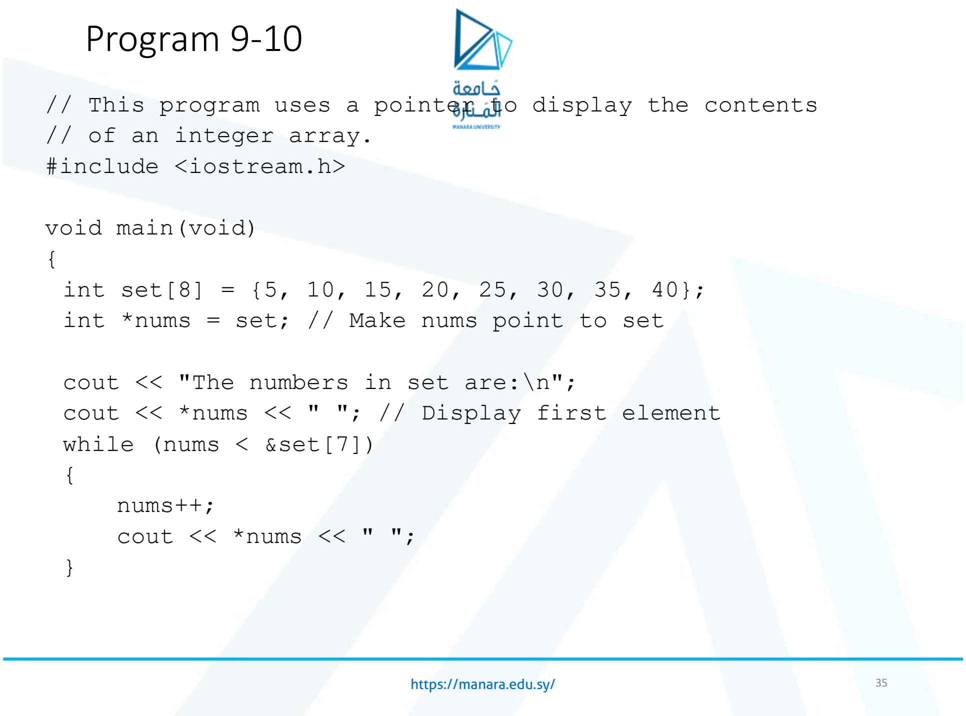
Figure 9-5

An array of five integers



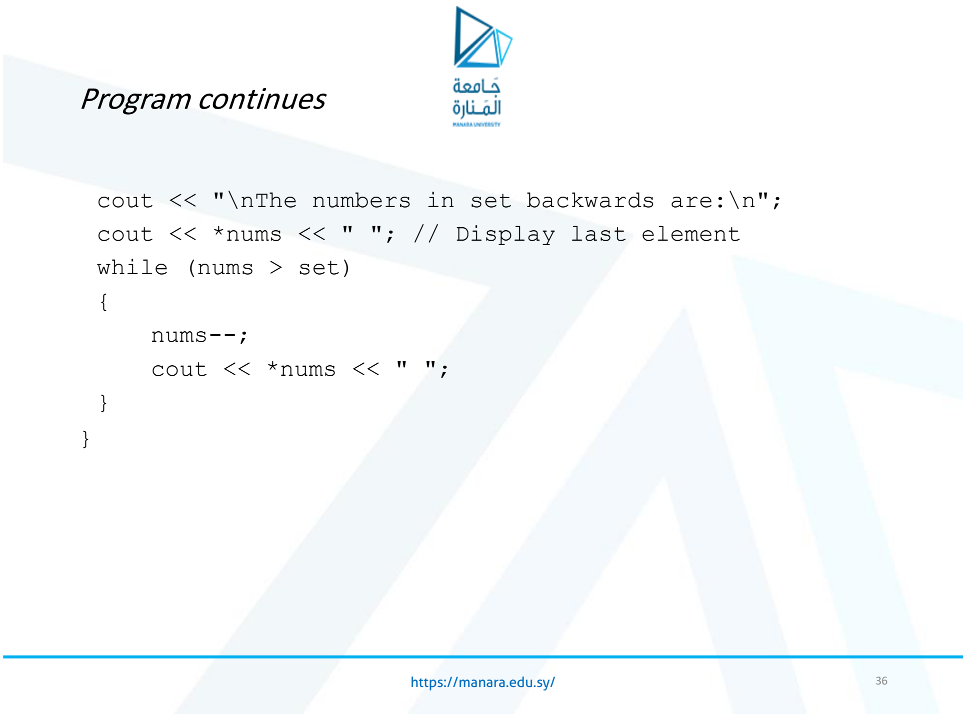
array[0]	array[1]	array[2]	array[3]	array[4]
0x5A00	0x5A04	0x5A08	0x5A0C	0x5A0F

(Addresses)



Program 9-10

```
// This program uses a pointer to display the contents
// of an integer array.
#include <iostream.h>

void main(void)
{
 int set[8] = {5, 10, 15, 20, 25, 30, 35, 40};
 int *nums = set; // Make nums point to set

 cout << "The numbers in set are:\n";
 cout << *nums << " "; // Display first element
 while (nums < &set[7])
 {
     nums++;
     cout << *nums << " ";
 }
```



Program continues

```
 cout << "\nThe numbers in set backwards are:\n";
 cout << *nums << " "; // Display last element
 while (nums > set)
 {
     nums--;
     cout << *nums << " ";
 }
}
```

Program Output

```
The numbers in set are:
5 10 15 20 25 30 35 40
The numbers in set backwards are:
40 35 30 25 20 15 10 5
```

9.7 Pointers as Function Parameters

- A pointer can be used as a function parameter. It gives the function access to the original argument, much like a reference parameter does.

Program 9-11

```
// This program uses two functions that accept addresses of
// variables as arguments.
#include <iostream.h>

// Function prototypes
void getNumber(int *);
void doubleValue(int *);

void main(void)
{
 int number;
 getNumber(&number) // Pass address of number to getNumber
 doubleValue(&number); // and doubleValue.
 cout << "That value doubled is " << number << endl;
}
```

Program continues

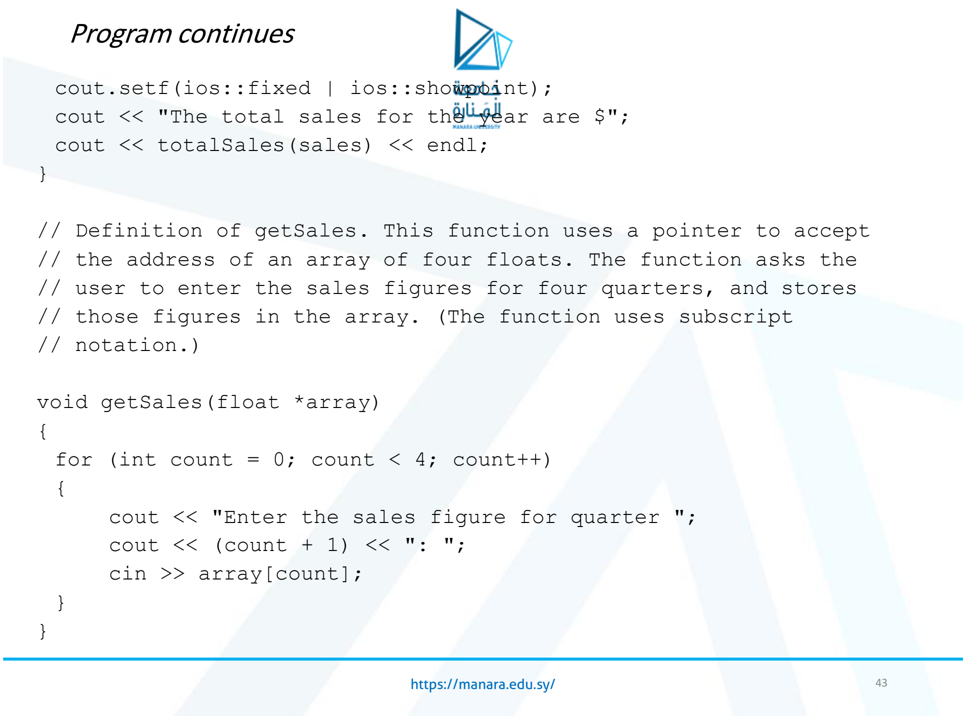
```
// Definition of getNumber. The parameter, Input, is a pointer.
// This function asks the user for a number. The value entered
// is stored in the variable pointed to by Input.

void getNumber(int *input)
{
 cout << "Enter an integer number: ";
 cin >> *input;
}

// Definition of doubleValue. The parameter, val, is a pointer.
// This function multiplies the variable pointed to by val by
// two.

void doubleValue(int *val)
{
 *val *= 2;
}
```

Program Output with Example Input

```
Enter an integer number: 10 [Enter]
That value doubled is 20
```

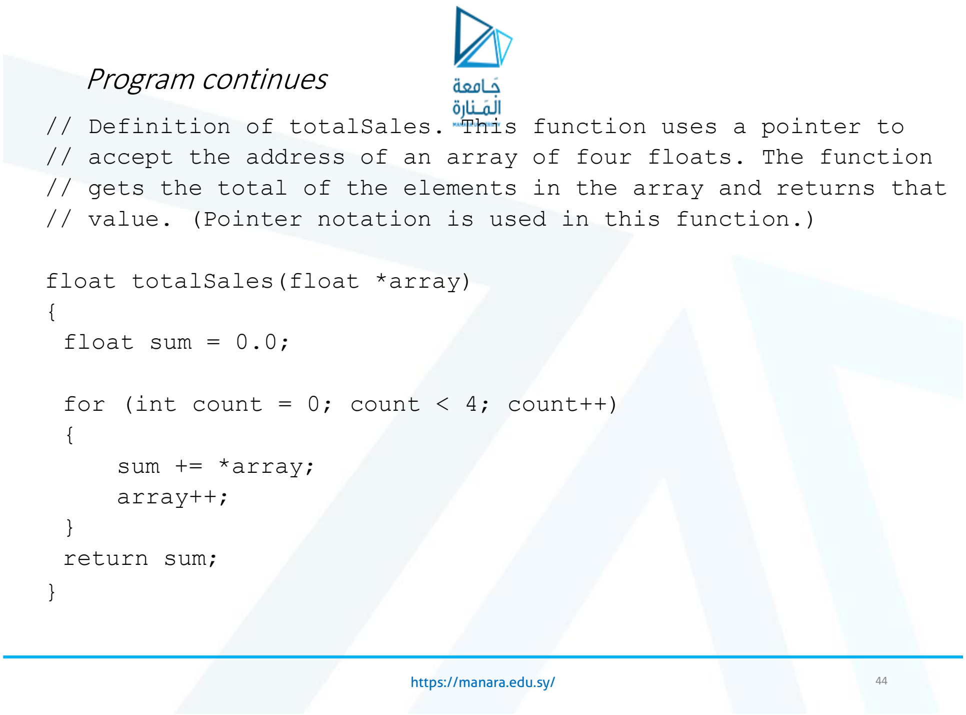
Program 9-12

```
// This program demonstrates that a pointer may be used as a
// parameter to accept the address of an array. Either subscript
// or pointer notation may be used.
#include <iostream.h>
#include <iomanip.h>

// Function prototypes
void getSales(float *);
float totalSales(float *);

void main(void)
{
  float sales[4];

  getSales(sales);
  cout.precision(2);
```



Program continues

```
  cout.setf(ios::fixed | ios::showpoint);
  cout << "The total sales for the year are $";
  cout << totalSales(sales) << endl;
}

// Definition of getSales. This function uses a pointer to accept
// the address of an array of four floats. The function asks the
// user to enter the sales figures for four quarters, and stores
// those figures in the array. (The function uses subscript
// notation.)

void getSales(float *array)
{
  for (int count = 0; count < 4; count++)
  {
      cout << "Enter the sales figure for quarter ";
      cout << (count + 1) << ": ";
      cin >> array[count];
  }
}
```



Program continues

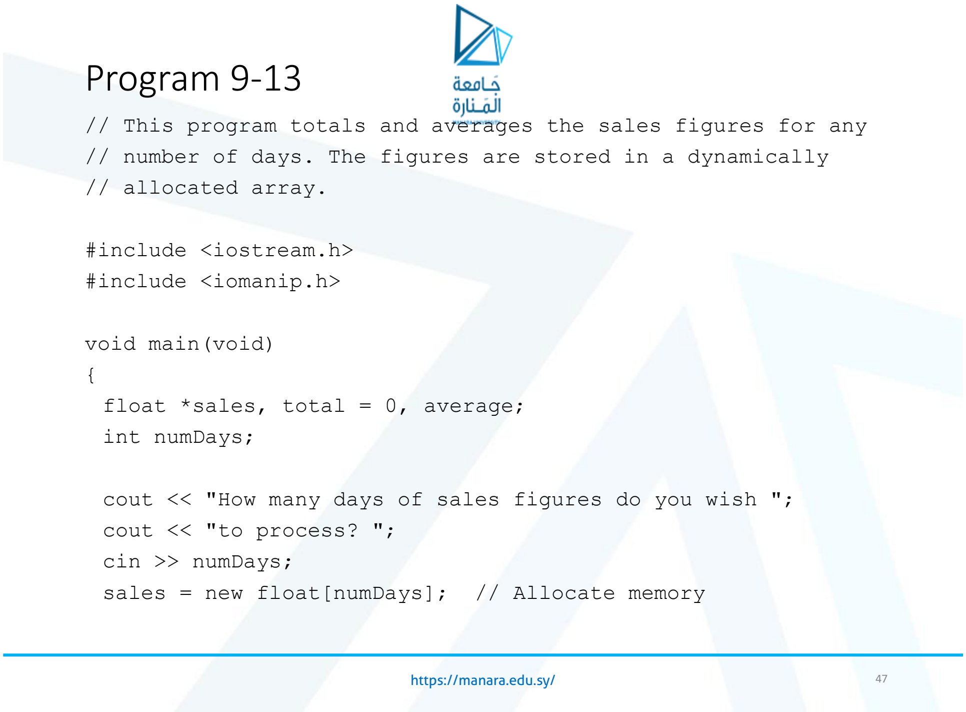
```
// Definition of totalSales. This function uses a pointer to
// accept the address of an array of four floats. The function
// gets the total of the elements in the array and returns that
// value. (Pointer notation is used in this function.)

float totalSales(float *array)
{
  float sum = 0.0;

  for (int count = 0; count < 4; count++)
  {
      sum += *array;
      array++;
  }
  return sum;
}
```

Program Output with Example Input

```
Enter the sales figure for quarter 1: 10263.98 [Enter]
Enter the sales figure for quarter 2: 12369.69 [Enter]
Enter the sales figure for quarter 3: 11542.13 [Enter]
Enter the sales figure for quarter 4: 14792.06 [Enter]
The total sales for the year are $48967.86
```

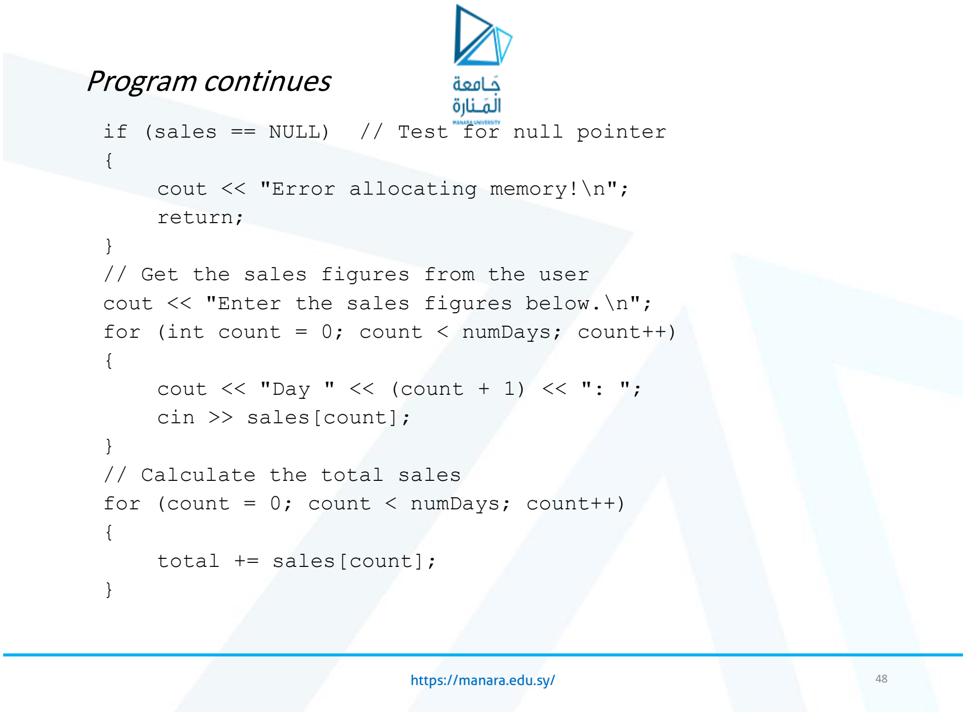
9.8 Focus on Software Engineering: Dynamic Memory Allocation

- Variables may be created and destroyed while a program is running.
- A pointer than contains the address 0 is called a null pointer.
- Use the new operator to dynamically allocate memory.
- Use delete to dynamically deallocate memory.



Program 9-13

```
// This program totals and averages the sales figures for any
// number of days. The figures are stored in a dynamically
// allocated array.

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

void main(void)
{
 float *sales, total = 0, average;
 int numDays;

 cout << "How many days of sales figures do you wish ";
 cout << "to process? ";
 cin >> numDays;
 sales = new float[numDays];  // Allocate memory
```



Program continues

```
 if (sales == NULL)  // Test for null pointer
 {
     cout << "Error allocating memory!\n";
     return;
 }
 // Get the sales figures from the user
 cout << "Enter the sales figures below.\n";
 for (int count = 0; count < numDays; count++)
 {
     cout << "Day " << (count + 1) << ": ";
     cin >> sales[count];
 }
 // Calculate the total sales
 for (count = 0; count < numDays; count++)
 {
     total += sales[count];
 }
```

Program continues

```
 // Calculate the average sales per day
 average = total / numDays;

 // Display the results
 cout.precision(2);
 cout.setf(ios::fixed | ios::showpoint);
 cout << "\n\nTotal sales: $" << total << endl;
 cout << "average sales: $" << average << endl;
 // Free dynamically allocated memory
 delete [] sales;
}
```

Program Output with Example Input

```
How many days of sales figures do you wish to process? 5 [Enter]
Enter the sales figures below.
Day 1: 898.63 [Enter]
Day 2: 652.32 [Enter]
Day 3: 741.85 [Enter]
Day 4: 852.96 [Enter]
Day 5: 921.37 [Enter]
total sales: $4067.13
average sales: $813.43
```

9.9 Focus on Software Engineering: Returning Pointers from Functions

- Functions can return pointers, but you must be sure the object the pointer references still exists.
- You should only return a pointer from a function if it is:
  - A pointer to an object that was passed into the function as an argument.
  - A pointer to a dynamically allocated object.