Week 10 Lecture Notes: Pointers Objectives

Concepts covered in this lesson:

- Getting the Address of a Variable
- Pointer Variables
- The Relationship Between Arrays and Pointers
- Pointer Arithmetic
- Initializing Pointers
- Comparing Pointers
- Pointers as Function Parameters
- Dynamic Memory Allocation
- Returning Pointers from Functions

Getting the Address of a Variable

- Each variable in program is stored at a unique address
- Use address operator `&` to get address of a variable:

```
int num = -99;
cout << &num; // prints address
              // in hexadecimal
```

Pointer Variables

- Pointer variable : Often just called a pointer, it's a variable that holds an address

- Because a pointer variable holds the address of another piece of data, it "points" to the data

Something Like Pointers: Arrays

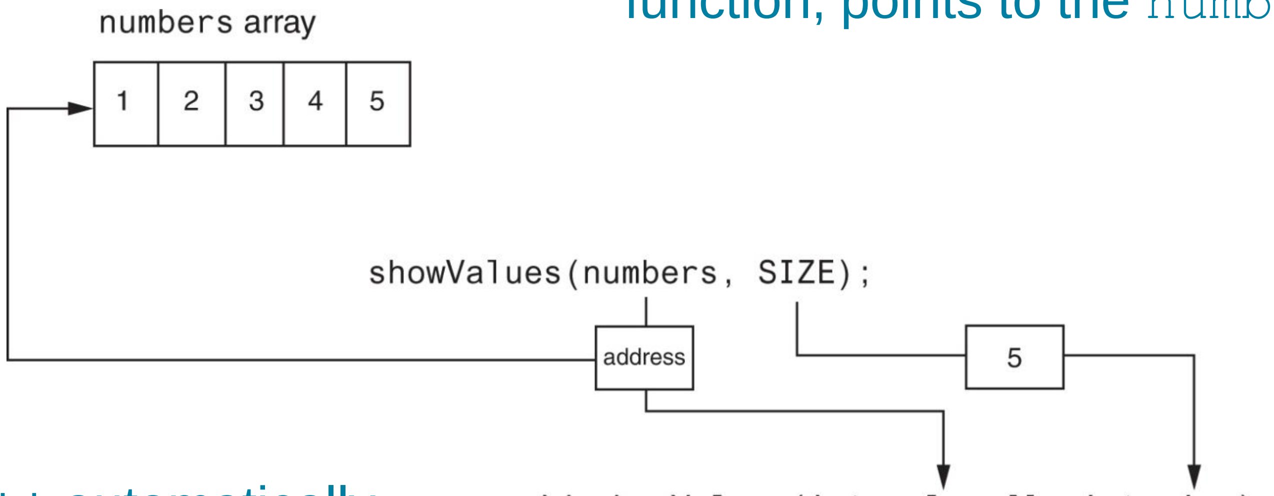
- We have already worked with something similar to pointers, when we learned to pass arrays as arguments to functions.

- For example, suppose we use this statement to pass the array `numbers` to the `showValues` function:

```
showValues(numbers, SIZE);
```

Something Like Pointers : Arrays

The `values` parameter, in the `showValues` function, points to the `numbers` array.



```
void showValues(int values[], int size)
{
   for (int count = 0; count < size; count++)
      cout << values[count] << endl;
}
```

C++ automatically stores the address of `numbers` in the `values` parameter.

Something Like Pointers: Reference Variables

- We have also worked with something like pointers when we learned to use reference variables. Suppose we have this function:

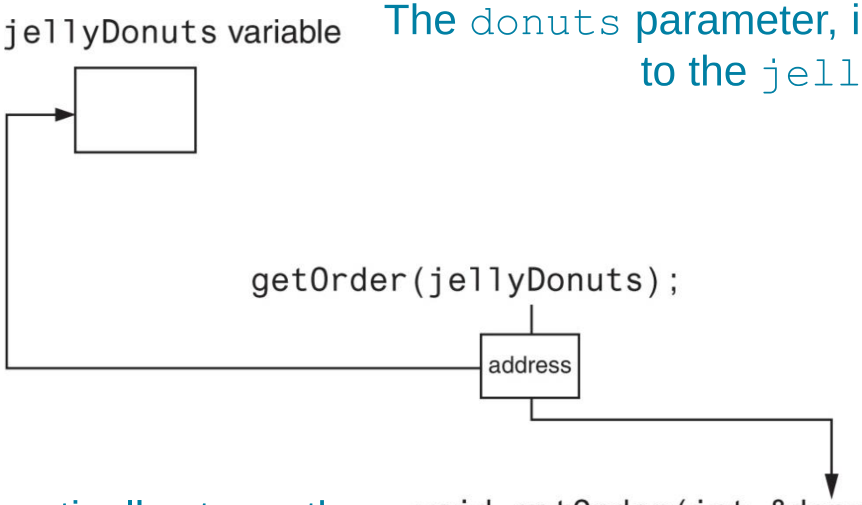
```
void getOrder(int &donuts)
{
   cout << "How many doughnuts do you want? ";
   cin >> donuts;
}
```

- And we call it with this code:

```
int jellyDonuts;
getOrder(jellyDonuts);
```

Something Like Pointers: Reference Variables

The donuts parameter, in the getOrder function, points to the jellyDonuts variable.



C++ automatically stores the address of jellyDonuts in the donuts parameter.

```
void getOrder(int &donuts)
{
    cout << "How many doughnuts do you want? ";
    cin >> donuts;
}
```

Pointer Variables

- Pointer variables are yet another way using a memory address to work with a piece of data.
- Pointers are more "low-level" than arrays and reference variables.
- This means you are responsible for finding the address you want to store in the pointer and correctly using it.

Pointer Variables

- Definition:

```
int  *intptr;
```

- Read as:

  “`intptr` can hold the address of an int”

- Spacing in definition does not matter:

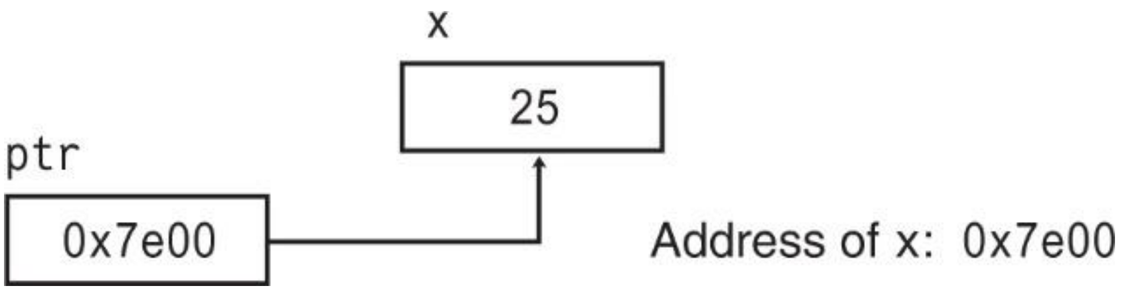
```
int * intptr;  // same as above
int*  intptr;  // same as above
```

Pointer Variables

- Assigning an address to a pointer variable:

```
int *intptr;
intptr = &num;
```

- Memory layout:

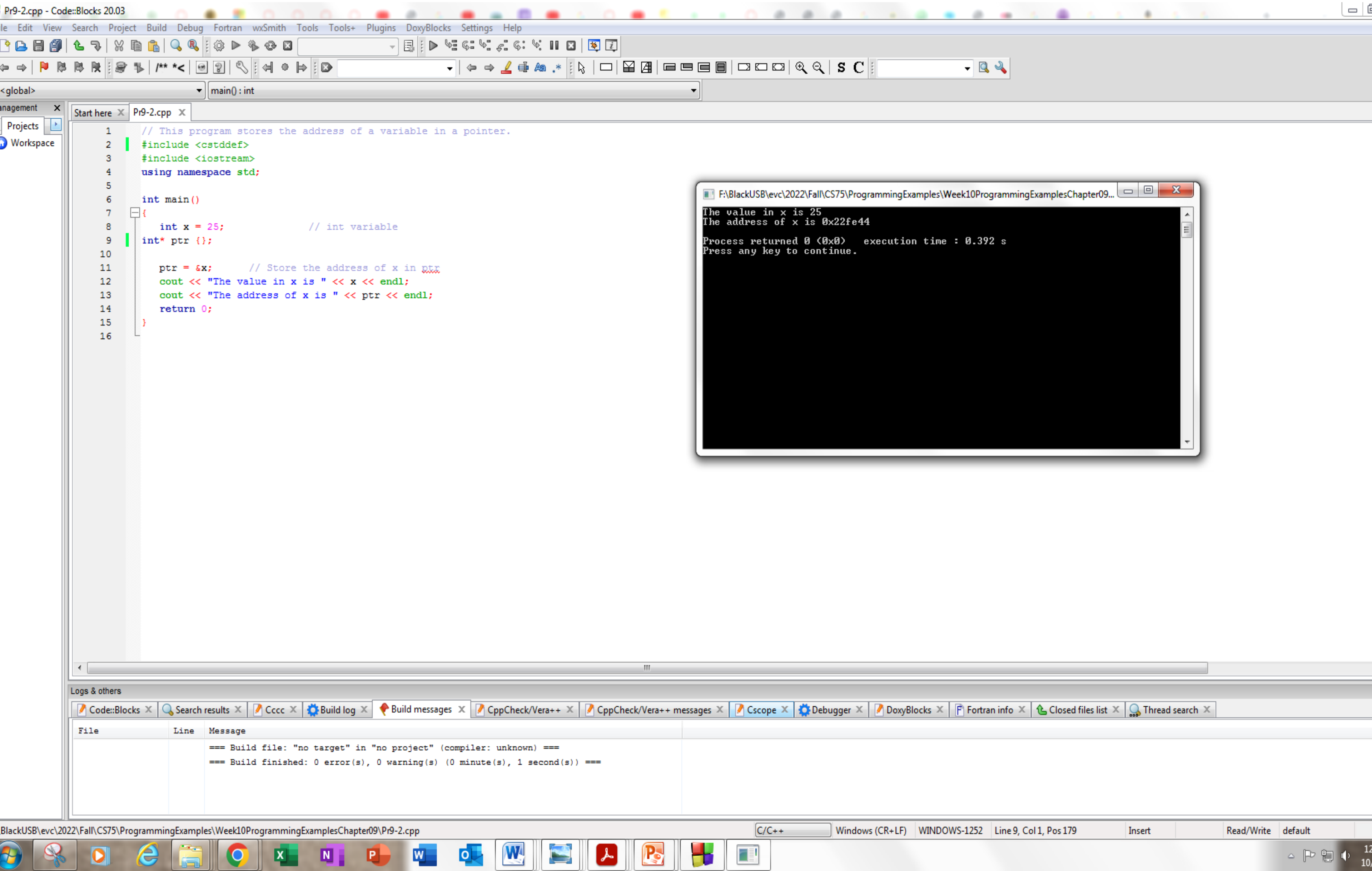


Pointer Variables

- Initialize pointer variables with the special value `nullptr`.
- In C++ 11, the `nullptr` key word was introduced to represent the address `0`.
- Here is an example of how you define a pointer variable and initialize it with the value `nullptr`:

```
int *ptr = nullptr;
```

A Pointer Variable in Program



```
// This program stores the address of a variable in a pointer.
#include <cstddef>
#include <iostream>
using namespace std;

int main()
{
    int x = 25;               // int variable
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;
    return 0;
}
```

F:\BlackUSB\evc\2022\Fall\CS75\ProgrammingExamples\Week10ProgrammingExamplesChapter09...

```
The value in x is 25
The address of x is 0x22fe44

Process returned 0 (0x0)   execution time : 0.392 s
Press any key to continue.
```

```
File        Line   Message
                   === Build file: "no target" in "no project" (compiler: unknown) ===
                   === Build finished: 0 error(s), 0 warning(s) (0 minute(s), 1 second(s)) ===
```

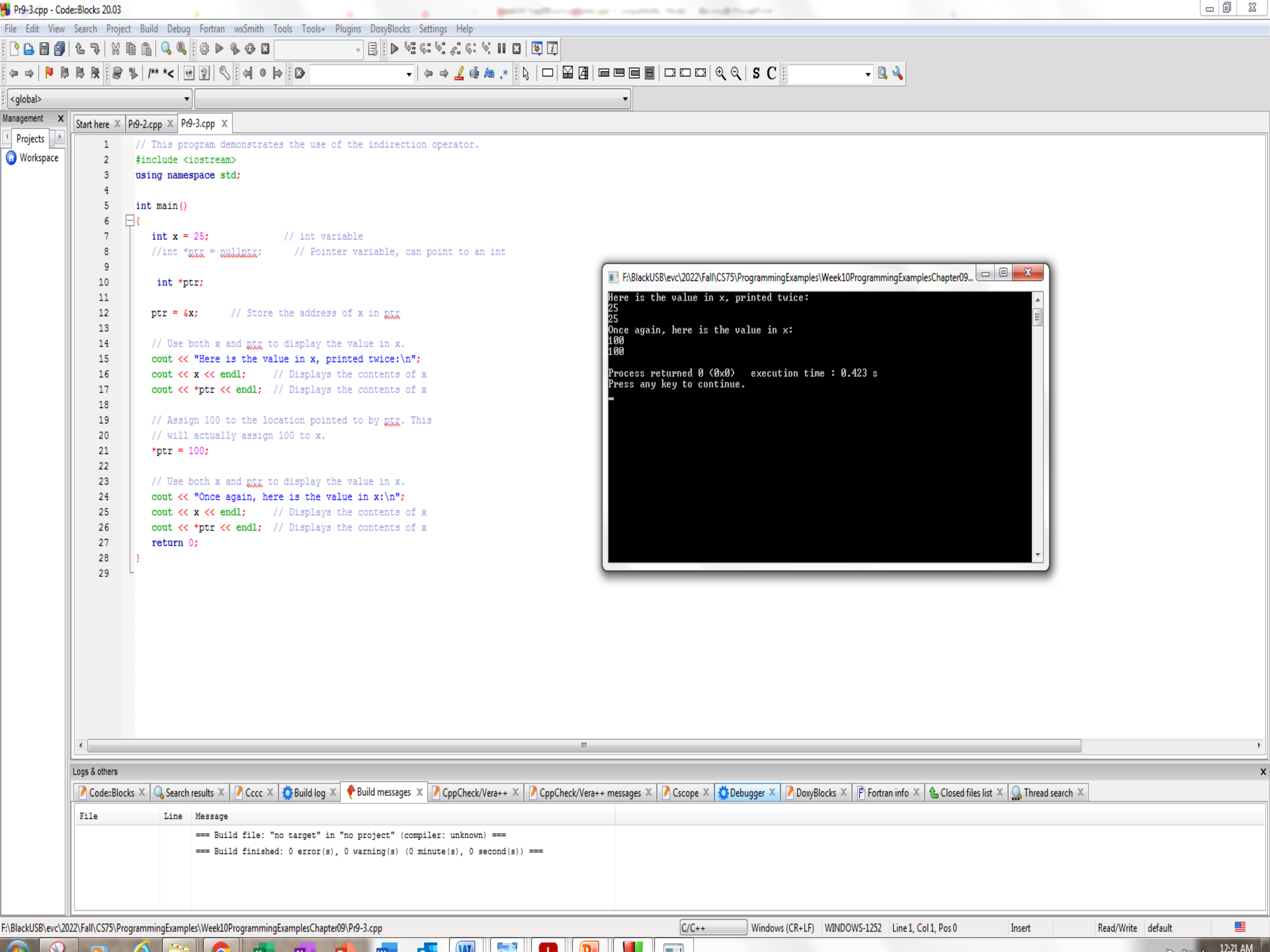
BlackUSB\evc\2022\Fall\CS75\ProgrammingExamples\Week10ProgrammingExamplesChapter09\Pr9-2.cpp

The Indirection Operator

- The indirection operator (`*`) dereferences a pointer.
- It allows you to access the item that the pointer points to.

```
int x = 25;
int *intptr = &x;
cout << *intptr << endl;
```

This prints 25.



```cpp
// This program demonstrates the use of the indirection operator.
#include <iostream>
using namespace std;

int main()
{
   int x = 25;              // int variable
   //int *ptr = nullptr;      // Pointer variable, can point to an int

    int *ptr;

   ptr = &x;      // Store the address of x in ptr

   // Use both x and ptr to display the value in x.
   cout << "Here is the value in x, printed twice:\n";
   cout << x << endl;     // Displays the contents of x
   cout << *ptr << endl;  // Displays the contents of x

   // Assign 100 to the location pointed to by ptr. This
   // will actually assign 100 to x.
   *ptr = 100;

   // Use both x and ptr to display the value in x.
   cout << "Once again, here is the value in x:\n";
   cout << x << endl;     // Displays the contents of x
   cout << *ptr << endl;  // Displays the contents of x
   return 0;
}
```

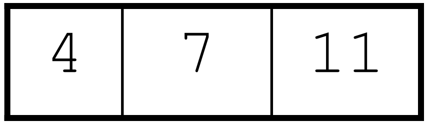
```
Here is the value in x, printed twice:
25
25
Once again, here is the value in x:
100
100

Process returned 0 (0x0)   execution time : 0.423 s
Press any key to continue.
```

```
File         Line   Message
                    === Build file: "no target" in "no project" (compiler: unknown) ===
                    === Build finished: 0 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

The Relationship Between Arrays and Pointers

- Array name is starting address of array

```
int vals[] = {4, 7, 11};
```



4	7	11

starting address of `vals`: `0x4a00`

```
cout << vals;              // displays
                           // 0x4a00
cout << vals[0];        // displays 4
```

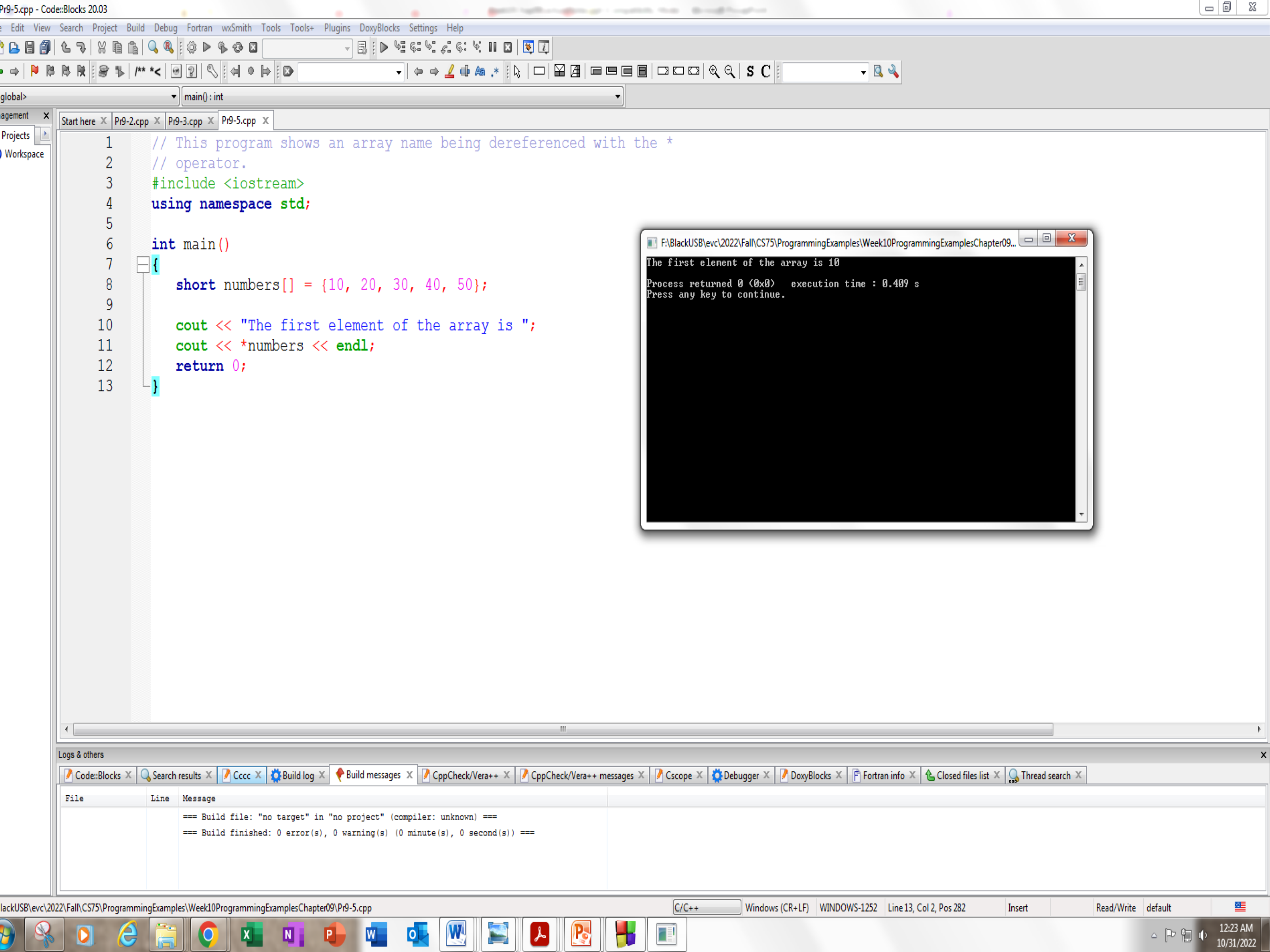
The Relationship Between Arrays and Pointers

- Array name can be used as a pointer constant:

```
int vals[] = {4, 7, 11};
cout << *vals;     // displays 4
```

- Pointer can be used as an array name:

```
int *valptr = vals;
cout << valptr[1]; // displays 7
```



```
// This program shows an array name being dereferenced with the *
// operator.
#include <iostream>
using namespace std;

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

   cout << "The first element of the array is ";
   cout << *numbers << endl;
   return 0;
}
```

F:\BlackUSB\evc\2022\Fall\CS75\ProgrammingExamples\Week10ProgrammingExamplesChapter09...

```
The first element of the array is 10

Process returned 0 (0x0)   execution time : 0.409 s
Press any key to continue.
```

```
File          Line   Message
                     === Build file: "no target" in "no project" (compiler: unknown) ===
                     === Build finished: 0 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

lackUSB\evc\2022\Fall\CS75\ProgrammingExamples\Week10ProgrammingExamplesChapter09\Pr9-5.cpp

Pointers in Expressions

Given:

```
int vals[]={4,7,11}, *valptr;
valptr = vals;
```

What is `valptr + 1`? It means (address in `valptr`) + (1 * size of an int)

```
cout << *(valptr+1); //displays 7
cout << *(valptr+2); //displays 11
```

Must use ( ) as shown in the expressions

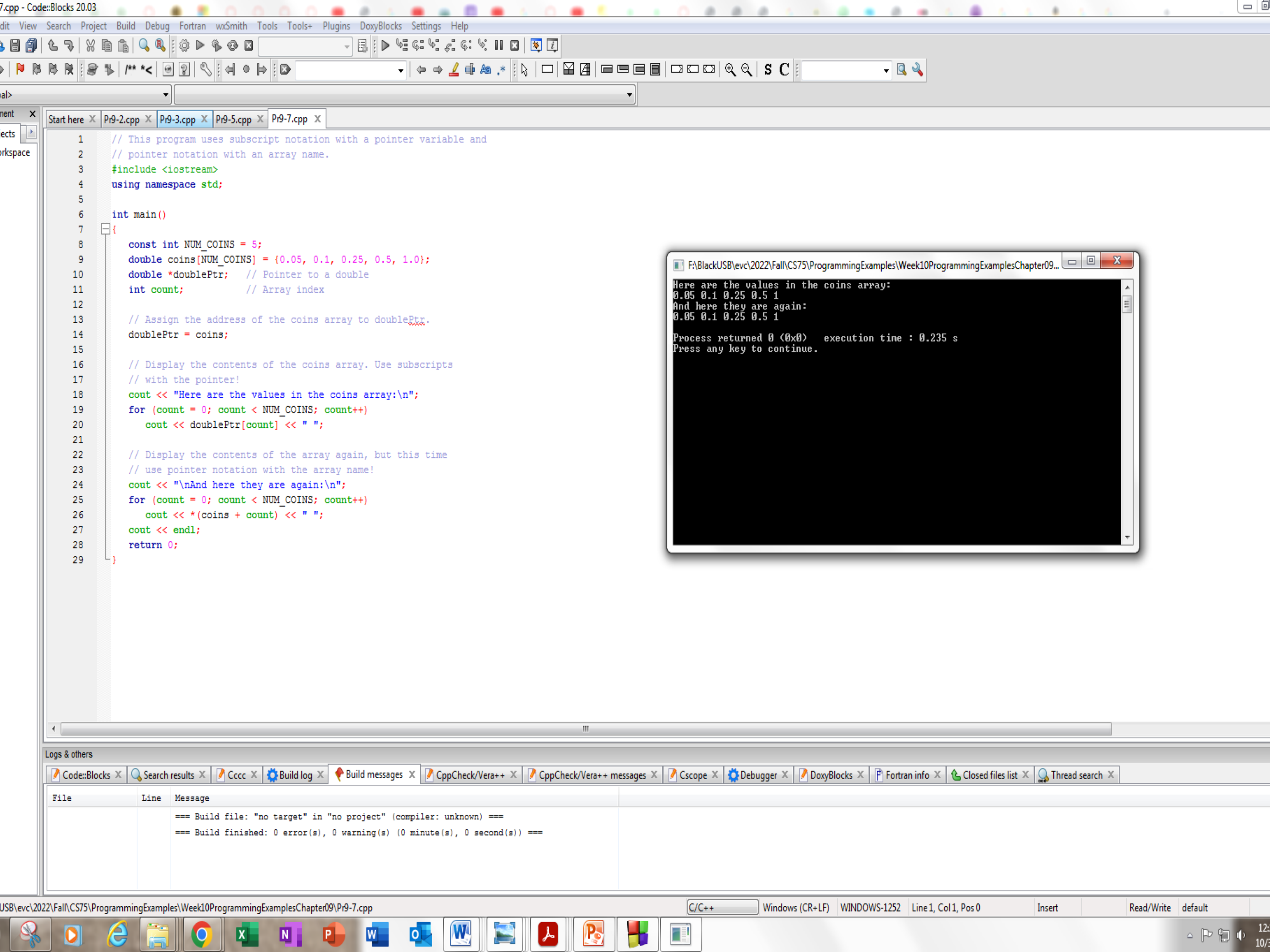
Array Access

- Array elements can be accessed in many ways:

Array access method	Example
array name and []	vals[2] = 17;
pointer to array and []	valptr[2] = 17;
array name and subscript arithmetic	*(vals + 2) = 17;
pointer to array and subscript arithmetic	*(valptr + 2) = 17;

Array Access

- Conversion: `vals[i]` is equivalent to `*(vals + i)`
- No bounds checking performed on array access, whether using array name or a pointer



```cpp
// This program uses subscript notation with a pointer variable and
// pointer notation with an array name.
#include <iostream>
using namespace std;

int main()
{
   const int NUM_COINS = 5;
   double coins[NUM_COINS] = {0.05, 0.1, 0.25, 0.5, 1.0};
   double *doublePtr;    // Pointer to a double
   int count;            // Array index

   // Assign the address of the coins array to doublePtr.
   doublePtr = coins;

   // Display the contents of the coins array. Use subscripts
   // with the pointer!
   cout << "Here are the values in the coins array:\n";
   for (count = 0; count < NUM_COINS; count++)
      cout << doublePtr[count] << " ";

   // Display the contents of the array again, but this time
   // use pointer notation with the array name!
   cout << "\nAnd here they are again:\n";
   for (count = 0; count < NUM_COINS; count++)
      cout << *(coins + count) << " ";
   cout << endl;
   return 0;
}
```

F:\BlackUSB\evc\2022\Fall\CS75\ProgrammingExamples\Week10ProgrammingExamplesChapter09...

```
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

Process returned 0 (0x0)   execution time : 0.235 s
Press any key to continue.
```

```
=== Build file: "no target" in "no project" (compiler: unknown) ===
=== Build finished: 0 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

USB\evc\2022\Fall\CS75\ProgrammingExamples\Week10ProgrammingExamplesChapter09\Pr9-7.cpp

Pointer Arithmetic

- Operations on pointer variables:

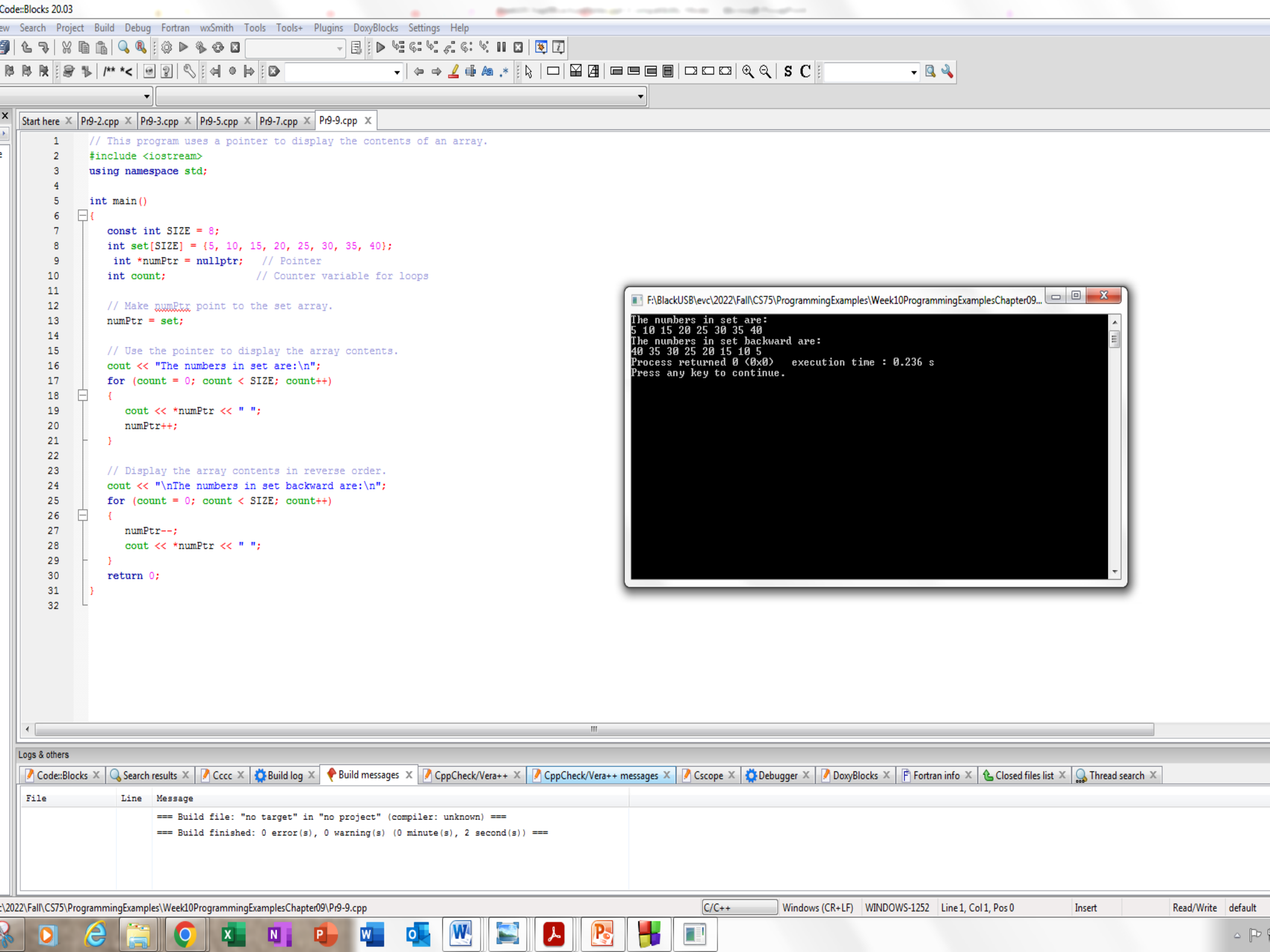
Operation	Example
	`int vals[]={4,7,11};`
	`int *valptr = vals;`

`++, --`	`valptr++; // points at 7`
	`valptr--; // now points at 4`

`+, -` (pointer and `int`)	`cout << *(valptr + 2); // 11`

`+=, -=` (pointer and `int`)	`valptr = vals; // points at 4`
	`valptr += 2;    // points at 11`

`-` (pointer from pointer)	`cout << valptr-val; // difference`
	`//(number of ints) between valptr`
	`// and val`


Code::Blocks 20.03
ew Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Start here
Pr9-2.cpp
Pr9-3.cpp
Pr9-5.cpp
Pr9-7.cpp
Pr9-9.cpp

```
// This program uses a pointer to display the contents of an array.
#include <iostream>
using namespace std;

int main()
{
   const int SIZE = 8;
   int set[SIZE] = {5, 10, 15, 20, 25, 30, 35, 40};
    int *numPtr = nullptr;   // Pointer
   int count;               // Counter variable for loops

   // Make numPtr point to the set array.
   numPtr = set;

   // Use the pointer to display the array contents.
   cout << "The numbers in set are:\n";
   for (count = 0; count < SIZE; count++)
   {
      cout << *numPtr << " ";
      numPtr++;
   }

   // Display the array contents in reverse order.
   cout << "\nThe numbers in set backward are:\n";
   for (count = 0; count < SIZE; count++)
   {
      numPtr--;
      cout << *numPtr << " ";
   }
   return 0;
}

```

F:\BlackUSB\evc\2022\Fall\CS75\ProgrammingExamples\Week10ProgrammingExamplesChapter09...

```
The numbers in set are:
5 10 15 20 25 30 35 40
The numbers in set backward are:
40 35 30 25 20 15 10 5
Process returned 0 (0x0)   execution time : 0.236 s
Press any key to continue.
```

File	Line	Message
		=== Build file: "no target" in "no project" (compiler: unknown) ===
		=== Build finished: 0 error(s), 0 warning(s) (0 minute(s), 2 second(s)) ===

:\2022\Fall\CS75\ProgrammingExamples\Week10ProgrammingExamplesChapter09\Pr9-9.cpp
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 1, Col 1, Pos 0
Insert
Read/Write
default

Initializing Pointers

- Can initialize at definition time:

```
int num, *numptr = &num;
int val[3], *valptr = val;
```

- Cannot mix data types:

```
double cost;
int *ptr = &cost; // won’t work
```

- Can test for an invalid address for `ptr` with:

```
if (!ptr) ...
```

Comparing Pointers

- Relational operators (<, >=, etc.) can be used to compare addresses in pointers
- Comparing addresses <u>in</u> pointers is not the same as comparing contents <u>pointed at by</u> pointers:

```
if (ptr1 == ptr2)    // compares
                     // addresses
if (*ptr1 == *ptr2)  // compares
                     // contents
```

Pointers as Function Parameters

- A pointer can be a parameter
- Works like reference variable to allow change to argument from within function
- Requires:

1) asterisk * on parameter in prototype and heading

```
void getNum(int *ptr); // ptr is pointer to an int
```

2) asterisk * in body to dereference the pointer

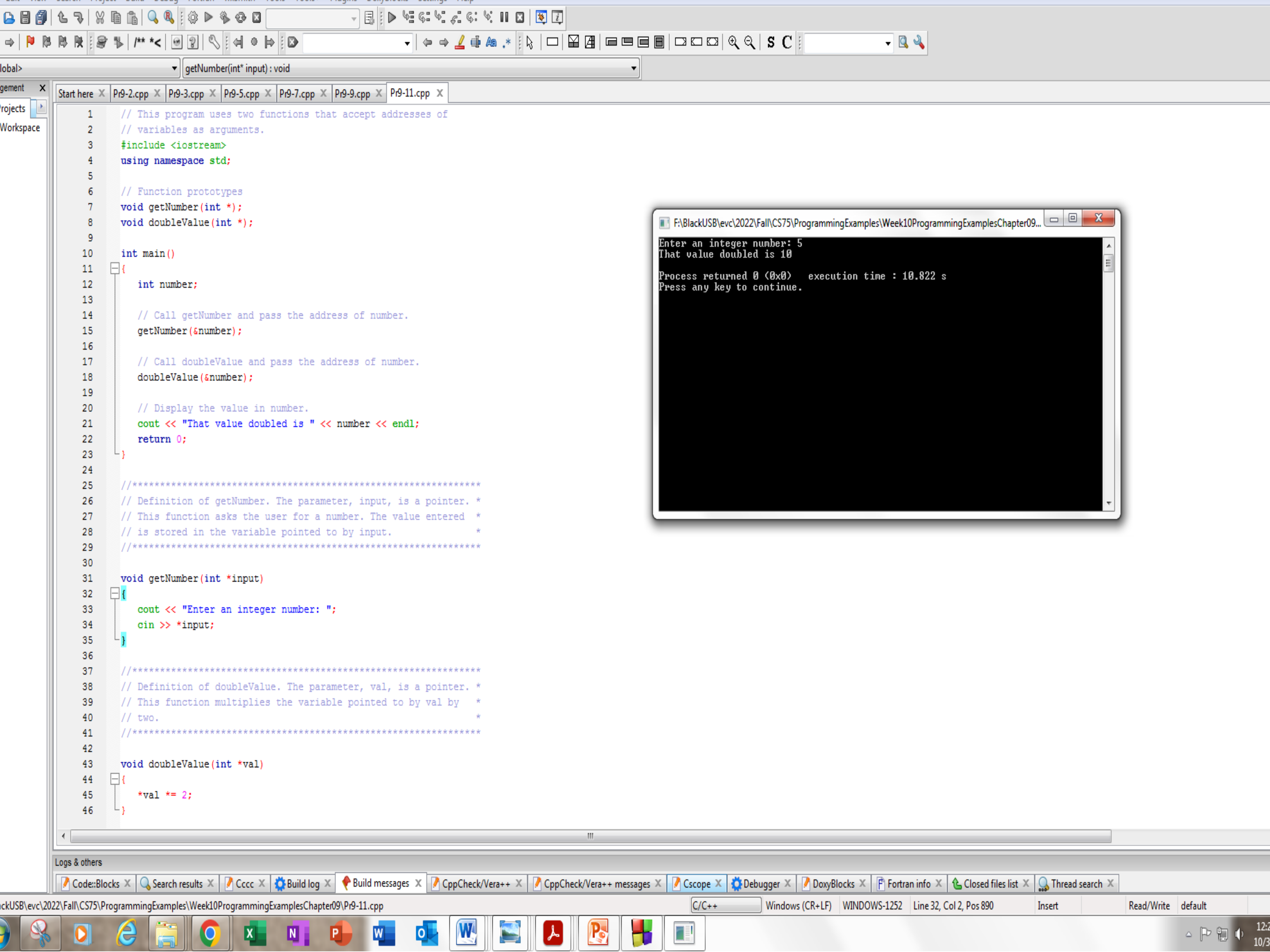
```
cin >> *ptr;
```

3) address as argument to the function

```
getNum(&num);      // pass address of num to getNum
```

Example

```
void swap(int *x, int *y)
{    int temp;
     temp = *x;
     *x = *y;
     *y = temp;
}

int num1 = 2, num2 = -3;
swap(&num1, &num2);
```



```cpp
// This program uses two functions that accept addresses of
// variables as arguments.
#include <iostream>
using namespace std;

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

int main()
{
    int number;

    // Call getNumber and pass the address of number.
    getNumber(&number);

    // Call doubleValue and pass the address of number.
    doubleValue(&number);

    // Display the value in number.
    cout << "That value doubled is " << number << endl;
    return 0;
}

//*****************************************************************
// 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;
}
```

F:\BlackUSB\evc\2022\Fall\CS75\ProgrammingExamples\Week10ProgrammingExamplesChapter09...

```
Enter an integer number: 5
That value doubled is 10

Process returned 0 (0x0)   execution time : 10.822 s
Press any key to continue.
```

Pointers to Constants

- If we want to store the address of a constant in a pointer, then we need to store it in a pointer-to-const.

Pointers to Constants

- Example: Suppose we have the following definitions:

```
const int SIZE = 6;
const double payRates[SIZE] =
      { 18.55, 17.45, 12.85,
        14.97, 10.35, 18.89 };
```

- In this code, `payRates` is an array of constant doubles.

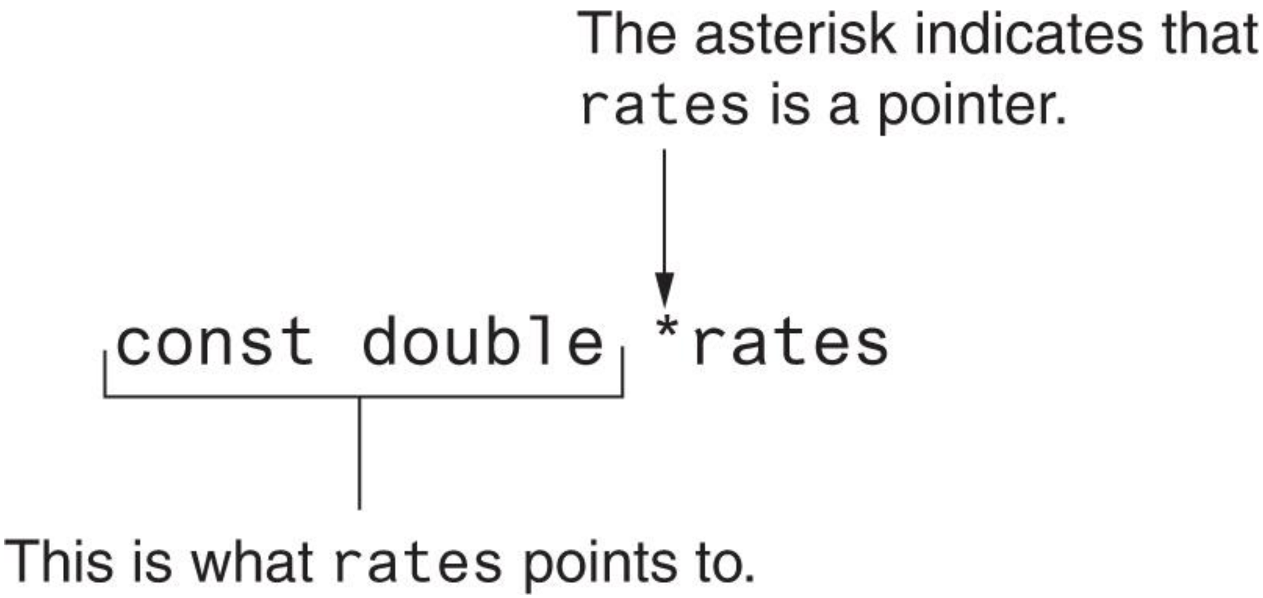
Pointers to Constants

- Suppose we wish to pass the `payRates` array to a function? Here's an example of how we can do it.

```
void displayPayRates(const double *rates, int
size)
{
   for (int count = 0; count < size; count++)
   {
      cout << "Pay rate for employee " <<
(count + 1)
           << " is $" << *(rates + count) <<
endl;
   }
}
```

The parameter, rates, is a pointer to `const double`.

Declaration of a Pointer to Constant



The asterisk indicates that rates is a pointer.

```
const double *rates
```

This is what rates points to.

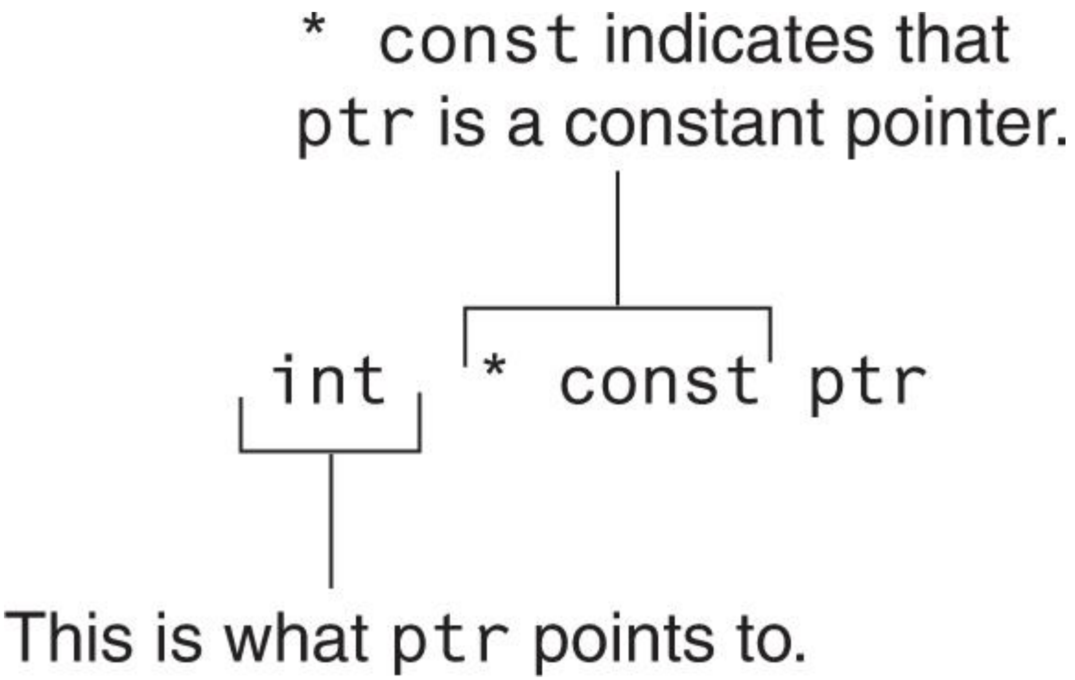
Constant Pointers

- A constant pointer is a pointer that is initialized with an address, and cannot point to anything else.

- Example

```
int value = 22;
int * const ptr = &value;
```

Constant Pointers



* const indicates that ptr is a constant pointer.

```
int * const ptr
```

This is what ptr points to.

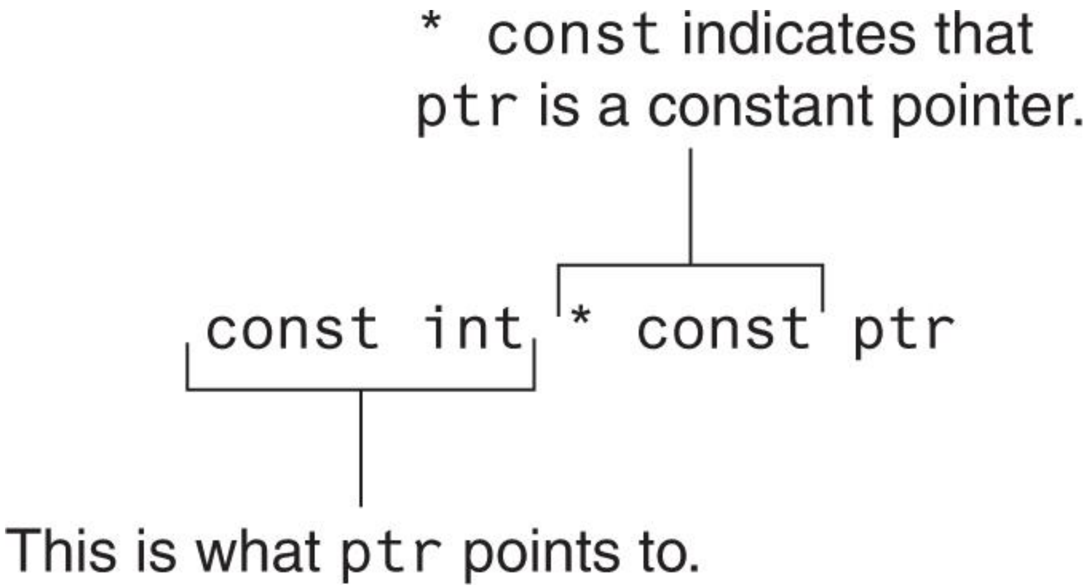
Constant Pointers to Constants

- A constant pointer to a constant is:
  - a pointer that points to a constant
  - a pointer that cannot point to anything except what it is pointing to

- Example:

```
int value = 22;
const int * const ptr = &value;
```

Constant Pointers to Constants



* const indicates that ptr is a constant pointer.

```
const int * const ptr
```

This is what ptr points to.

Dynamic Memory Allocation

- Can allocate storage for a variable while program is running
- Computer returns address of newly allocated variable
- Uses `new` operator to allocate memory:

```
double *dptr = nullptr;
dptr = new double;
```

- `new` returns address of memory location

Dynamic Memory Allocation

- Can also use `new` to allocate array:

```
const int SIZE = 25;
arrayPtr = new double[SIZE];
```

- Can then use `[]` or pointer arithmetic to access array:

```
  for(i = 0; i < SIZE; i++)
     *arrayptr[i] = i * i;
```

or

```
  for(i = 0; i < SIZE; i++)
     *(arrayptr + i) = i * i;
```

- Program will terminate if not enough memory available to allocate

Releasing Dynamic Memory

- Use `delete` to free dynamic memory:

```
delete fptr;
```

- Use `[]` to free dynamic array:

```
delete [] arrayptr;
```

- Only use `delete` with dynamic memory!

Returning Pointers from Functions

- Pointer can be the return type of a function:

```
int* newNum();
```

- The function must not return a pointer to a local variable in the function.
- A function should only return a pointer:
  - to data that was passed to the function as an argument, or
  - to dynamically allocated memory