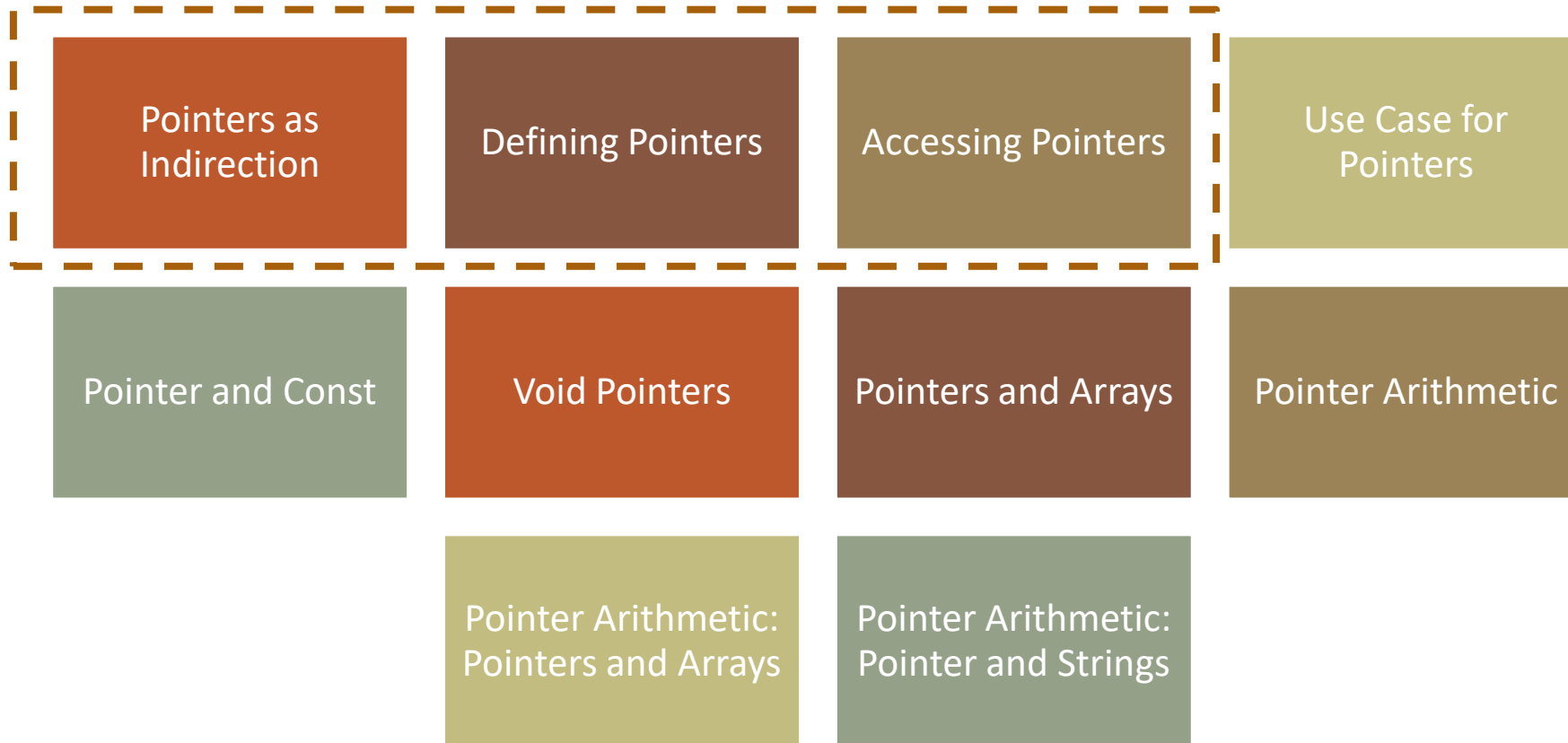


Pointers



Pointers as Indirection

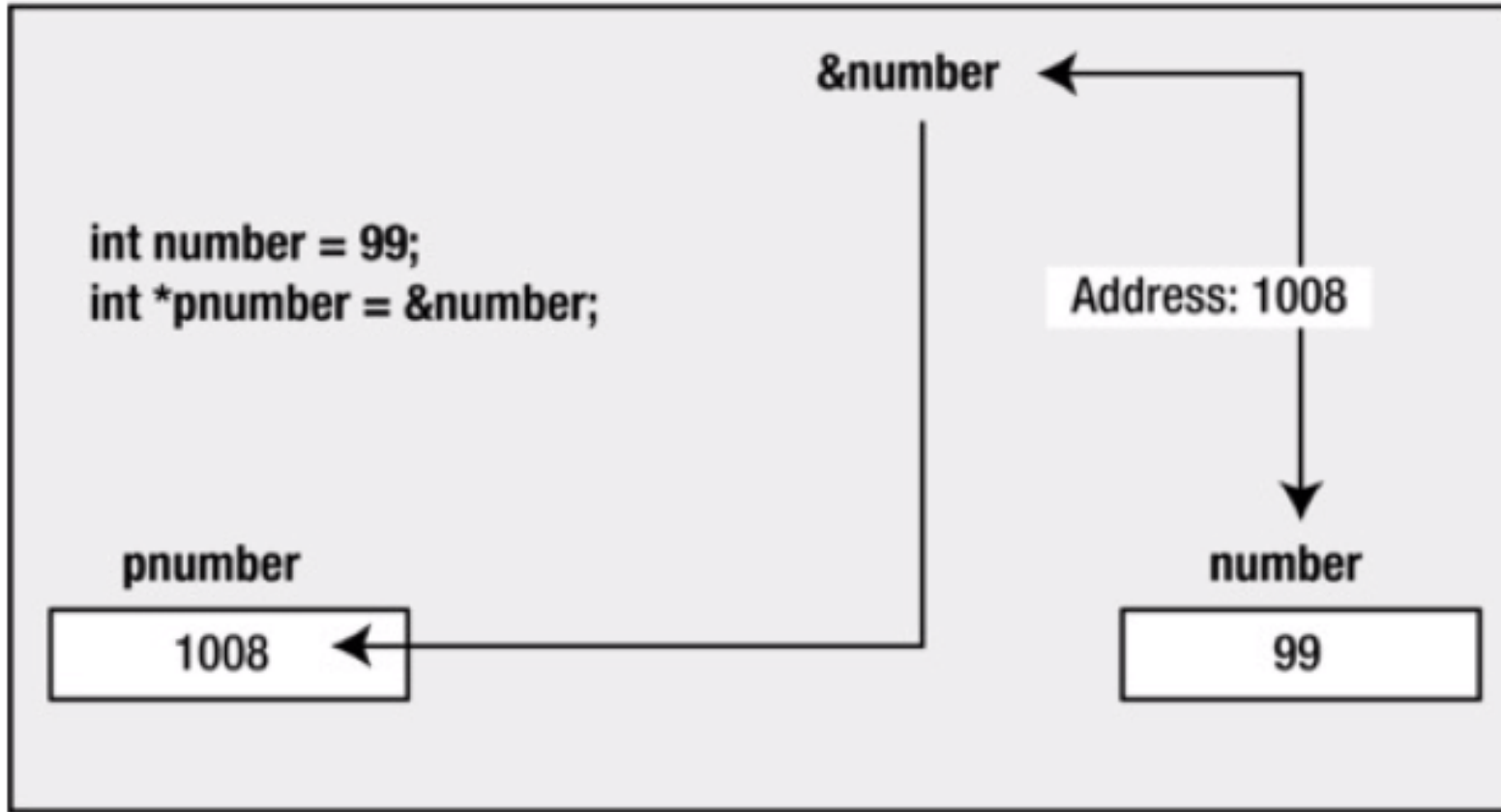
Indirection

- pointers are very similar to the concept of indirection that you employ in your everyday life
 - suppose you need to buy a new ink cartridge for your printer
 - all purchases are handled by the purchasing department
 - you call Joe in purchasing and ask him to order the new cartridge for you
 - Joe then calls the local supply store to order the cartridge
 - you are not ordering the cartridge directly from the supply store yourself (indirection)
- in programming languages, indirection is the ability to reference something using a name, reference, or container, instead of the value itself
- the most common form of indirection is the act of manipulating a value through its memory address
- a pointer provides an indirect means of accessing the value of a particular data item
 - a variable whose value is a memory address
 - its value is the address of another location in memory that can contain a value

Overview

- just as there are reasons why it makes sense to go through the purchasing department to order new cartridges (you don't have to know which particular store the cartridges are being ordered from)
 - there are good reasons why it makes sense to use pointers in C
- using pointers in your program is one of the most powerful tools available in the C language
- pointers are also one of the most confusing concepts of the C language
 - it is important you get this concept figured out in the beginning and maintain a clear idea of what is happening as you dig deeper
- the compiler must know the type of data stored in the variable to which it points
 - need to know how much memory is occupied or how to handle the contents of the memory to which it points
 - every pointer will be associated with a specific variable type
 - it can be used only to point to variables of that type
- pointers of type "pointer to int" can point only to variables of type int

Overview (cont'd)



(taken from Beginning C, Horton)

- the value of `&number` is the address where `number` is located
 - this value is used to initialize `pnumber` in the second statement

count



count directly references a variable that contains the value 7

countPtr



count



Pointer countPtr indirectly references a variable that contains the value 7

Fig. 7.1 | Directly and indirectly referencing a variable.

Why use pointers?

- accessing data by means of only variables is very limiting
 - with pointers, you can access any location (you can treat any position of memory as a variable for example) and perform arithmetic with pointers
 - pointers in C make it easier to use arrays and strings
 - pointers allow you to refer to the same space in memory from multiple locations
 - means that you can update memory in one location and the change can be seen from another location in your program
 - can also save space by being able to share components in your data structures
 - pointers allow functions to modify data passed to them as variables
 - pass by reference - passing arguments to function in way they can be changed by function
 - can also be used to optimize a program to run faster or use less memory than it would otherwise
-

Why use pointers?

- pointers allow us to get multiple values from the function
 - a function can return only one value but by passing arguments as pointers we can get more than one values from the pointer
- with pointers dynamic memory can be created according to the program use
 - we can save memory from static (compile time) declarations
- pointers allow us to design and develop complex data structures like a stack, queue, or linked list
- pointers provide direct memory access.

Defining Pointers

Declaring pointers

- pointers are not declared like normal variables

`pointer ptr; // not the way to declare a pointer/`

- it is not enough to say that a variable is a pointer
 - you also have to specify the kind of variable to which the pointer points
 - different variable types take up different amounts of storage
 - some pointer operations require knowledge of that storage size
- you declare a pointer to a variable of type `int` with:

`int *pnumber;`

- the type of the variable with the name `pnumber` is `int*`
 - can store the address of any variable of type `int`

`int * pi; // pi is a pointer to an integer variable`
`char * pc; // pc is a pointer to a character variable`
`float * pf, * pg; // pf, pg are pointers to float variables`

Declaring pointers (cont'd)

- the space between the * and the pointer name is optional
 - programmers use the space in a declaration and omit it when dereferencing a variable
 - the value of a pointer is an address, and it is represented internally as an unsigned integer on most systems
 - however, you shouldn't think of a pointer as an integer type
 - things you can do with integers that you can not do with pointers, and vice versa
 - you can multiply one integer by another, but you can not multiply one pointer by another
 - a pointer really is a new type, not an integer type
 - %p represents the format specifier for pointers
 - the previous declarations creates the variable but does not initialize it
 - dangerous when not initialized
 - you should always initialize a pointer when you declare it
-

NULL Pointers

- you can initialize a pointer so that it does not point to anything:

```
int *pnumber = NULL;
```

- NULL is a constant that is defined in the standard library
 - is the equivalent of zero for a pointer
- NULL is a value that is guaranteed not to point to any location in memory
 - means that it implicitly prevents the accidental overwriting of memory by using a pointer that does not point to anything specific
- add an `#include` directive for `stddef.h` to your source file

Common Programming Error

7.1

The asterisk (*) notation used to declare pointer variables does not distribute to all variable names in a declaration. Each pointer must be declared with the * prefixed to the name; e.g., if you wish to declare xPtr and yPtr as int pointers, use `int *xPtr, *yPtr;`.

Good Programming Practice 7.1

Include the letters `ptr` in pointer variable names to make it clear that these variables are pointers and thus need to be handled appropriately.

Error-Prevention Tip 7.1

Initialize pointers to prevent unexpected results.

Assignment revisited — Pointers and Vars

`X = 17;`

lvalue = rvalue

lvalue: expression that evaluates to a location

rvalue: expression that evaluates to a value

Simple Pointers

Pointer is a value that points to a location in the memory

Pointer is associated with a type

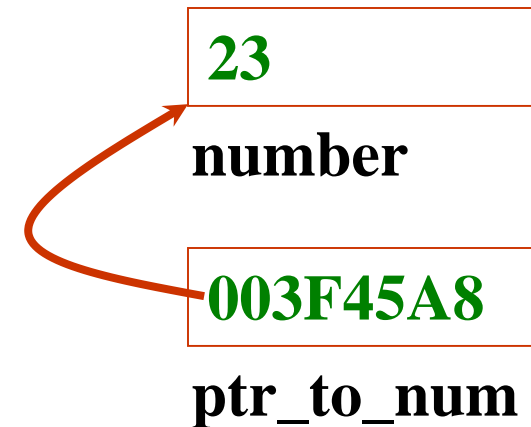
```
int number ;
```

```
int * ptr_to_num ;
```

```
number = 23;
```

```
ptr_to_num = & number;
```

```
printf("Value is %d \n", (*ptr_to_num) );
```



More Pointers

```
int number ;
```

```
int * p1, * p2;
```

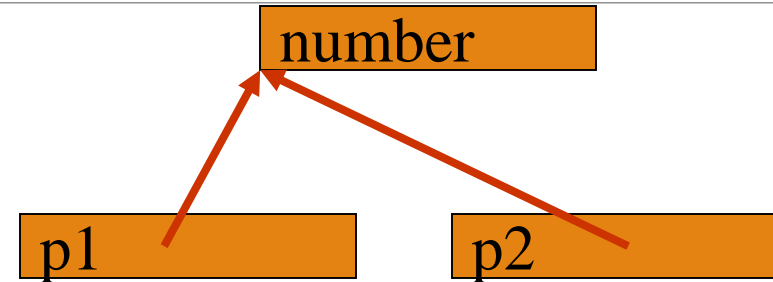
```
p1 = & number ;
```

```
number = 23;
```

```
p2 = & number ;
```

```
printf(" *p1 = %d  *p2 = %d ", *p1, *p2);
```

```
/* Output ?? */
```



Pointers and Arrays

```
char str[32];
```

```
char *ptr;
```

```
ptr = str ;
```

```
strcpy( str, "test" );
```

```
strcpy( ptr, "test" );    /* does the same as above */
```

Pointers and Arrays

```
int table [8];
```

```
int *ptr ;
```

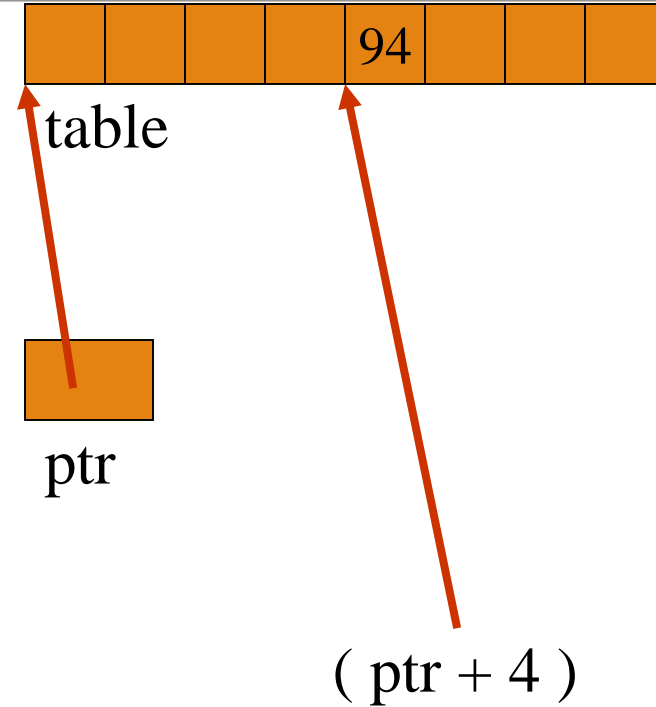
```
ptr = table ;
```

```
table [ 4 ] = 94;
```

```
*( ptr + 4 ) = 94;
```

How about

`ptr = & table[0]??` vs. `ptr=table;??`



Address of operator

- if you want to initialize your variable with the address of a variable you have already declared
 - use the address of operator, &

```
int number = 99;  
int *pnumber = &number;
```

- the initial value of pnumber is the address of the variable number
 - the declaration of number must precede the declaration of the pointer that stores its address
 - compiler must have already allocated space and thus an address for number to use it to initialize pnumber

Be careful

- there is nothing special about the declaration of a pointer
 - can declare regular variables and pointers in the same statement

```
double value, *pVal, fnum;
```

- only the second variable, pVal, is a pointer

```
int *p, q;
```

- the above declares a pointer, p of type int*, and a variable, q, that is of type int
 - a common mistake to think that both p and q are pointers
- also, it is a good idea to use names beginning with p as pointer names

Accessing Pointers

Accessing pointer values

- you use the indirection operator, *, to access the value of the variable pointed to by a pointer
 - also referred to as the dereference operator because you use it to “dereference” a pointer

```
int number = 15;  
int *pointer = &number;  
int result = 0;
```

- the pointer variable contains the address of the variable number
 - you can use this in an expression to calculate a new value for result

```
result = *pointer + 5;
```

- the expression *pointer will evaluate to the value stored at the address contained in the pointer
 - the value stored in number, 15, so result will be set to 15 + 5, which is 20
 - the indirection operator, *, is also the symbol for multiplication, and it is used to specify pointer types
 - depending on where the asterisk appears, the compiler will understand whether it should interpret it as an indirection operator, as a multiplication sign, or as part of a type specification
 - context determines what it means in any instance
-

Example

```
int main (void)
{
    int  count = 10, x;
    int  *int_pointer;

    int_pointer = &count;
    x = *int_pointer;

    printf ("count = %i, x = %i\n", count, x);

    return 0;
}
```

Displaying a pointers value

- to output the address of a variable, you use the output format specifier %p
 - outputs a pointer value as a memory address in hexadecimal form

```
int number = 0;           // A variable of type int initialized to 0
int *pnumber = NULL;      // A pointer that can point to type int

number = 10;
pnumber = &number;
printf("pnumber's value: %p\n", pnumber);    // Output the value (an address)
```

- pointers occupy 8 bytes and the addresses have 16 hexadecimal digits
 - if a machine has a 64-bit operating system and my compiler supports 64-bit addresses
 - some compilers only support 32-bit addressing, in which case addresses will be 32-bit addresses
-

Displaying an address (cont'd)

```
printf("number's address: %p\n", &number);           // Output the address  
printf("pnumber's address: %p\n", (void*)&pnumber);    // Output the address
```

- remember, a pointer itself has an address, just like any other variable
 - you use %p as the conversion specifier to display an address
- you use the & (address of) operator to reference the address that the pnumber variable occupies
- the cast to void* is to prevent a possible warning from the compiler
 - the %p specification expects the value to be some kind of pointer type, but the type of &pnumber is “pointer to pointer to int”

Displaying the number of bytes a pointer is using

- you use the sizeof operator to obtain the number of bytes a pointer occupies
- you may get a compiler warning when using sizeof this way
 - size_t is an implementation-defined integer type
 - to prevent the warning, you could cast the argument to type int like this:

```
printf("pnumber's size: %d bytes\n", (int)sizeof(pnumber)); // Output the size
```

Example

```
int main(void)
{
    int number = 0;           // A variable of type int initialized to 0
    int *pnumber = NULL;      // A pointer that can point to type int

    number = 10;
    printf("number's address: %p\n", &number);           // Output the address
    printf("number's value: %d\n\n", number);            // Output the value

    pnumber = &number;      // Store the address of number in pnumber

    printf("pnumber's address: %p\n", (void*)&pnumber);  // Output the address
    printf("pnumber's size: %zd bytes\n", sizeof(pnumber)); // Output the size
    printf("pnumber's value: %p\n", pnumber);           // Output the value (an address)
    printf("value pointed to: %d\n", *pnumber);         // Value at the address
    return 0;
}
```

main.c [Test] - Code::Blocks 16.01

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Debug <global> main(void) : int

Management test.c x main.c x

Projects S Workspace Test Sources

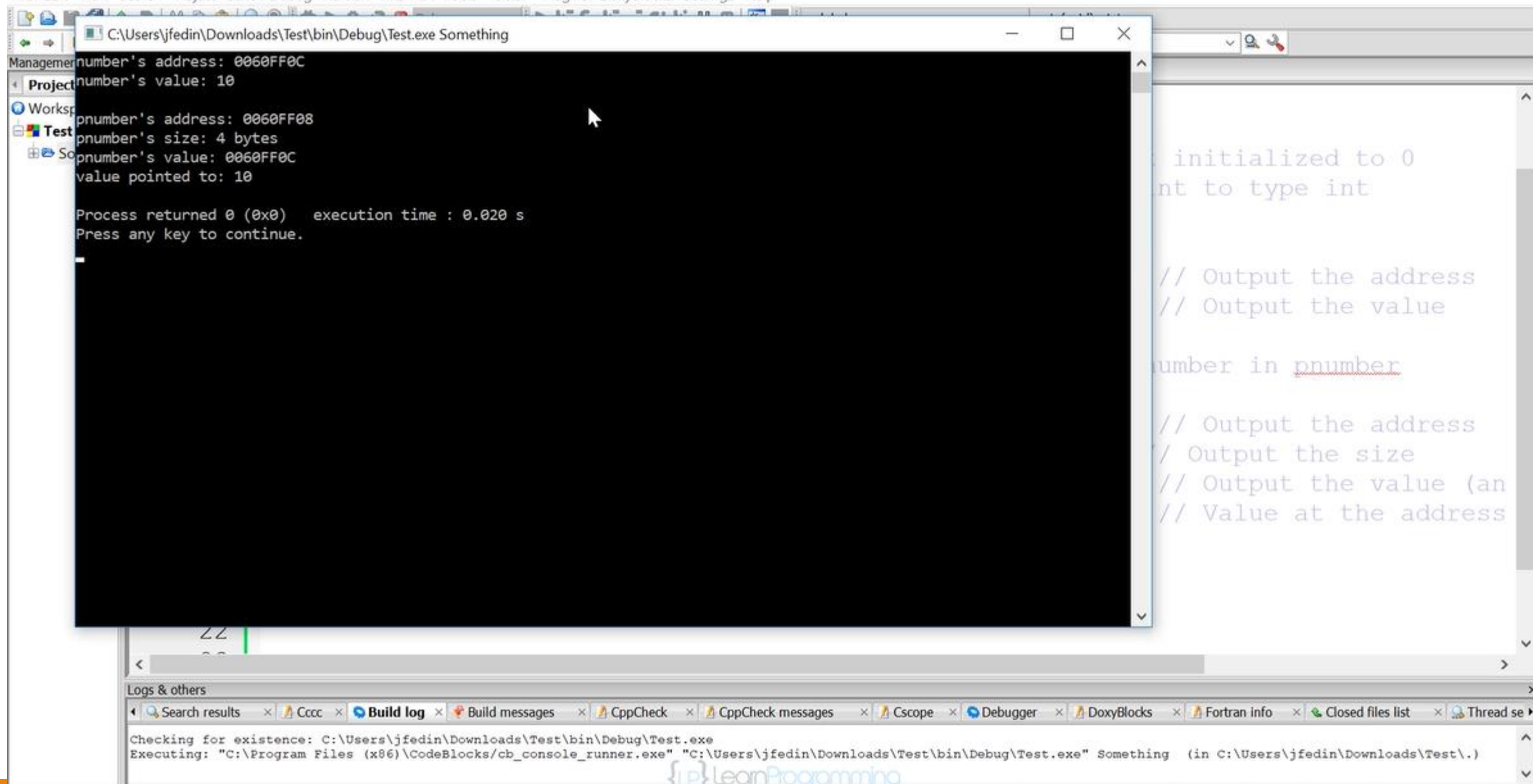
```
4 int main(void)
5 {
6     int number = 0;           // A variable of type int initialized to 0
7     int *pnumber = NULL;      // A pointer that can point to type int
8
9     number = 10;
10    printf("number's address: %p\n", &number);           // Output the address
11    printf("number's value: %d\n\n", number);             // Output the value
12
13    pnumber = &number;      // Store the address of number in pnumber
14
15    printf("pnumber's address: %p\n", (void*)&pnumber);   // Output the address
16    printf("pnumber's size: %d bytes\n", sizeof(pnumber)); // Output the size
17    printf("pnumber's value: %p\n", pnumber);             // Output the value (an
18    printf("value pointed to: %d\n", *pnumber);           // Value at the address
19    return 0;
20 }
21
22
```

Logs & others

Search results x Cccc x Build log x Build messages x CppCheck x CppCheck messages x Cscope x Debugger x DoxyBlocks x Fortran info x Closed files list x Thread se

Executing: "C:\Program Files (x86)\CodeBlocks\cb_console_runner.exe" "C:\Users\jfedin\Downloads\Test\bin\Debug\Test.exe" Something (in C:\Users\jfedin\Downloads\Test\.)
Process terminated with status 0 (0 minute(s), 11 second(s))

Learn Programming



The screenshot shows the Code::Blocks IDE interface. A console window titled "C:\Users\jfedin\Downloads\Test\bin\Debug\Test.exe Something" is open, displaying the output of a program. The output shows the address and value of a variable named 'number' at two different points in the program. The first output shows the address as 0060FF0C and the value as 10. The second output shows the address as 0060FF08, the size as 4 bytes, the value as 0060FF0C, and the value pointed to as 10. The console also shows the process returned 0 (0x0) and the execution time was 0.020 s. The source code editor in the background shows the following code:

```
int number = 0;

// Output the address
// Output the value

// Output the address
// Output the size
// Output the value (an
// Value at the address
```

The "Logs & others" panel at the bottom shows the build log, which includes the following text:

```
Checking for existence: C:\Users\jfedin\Downloads\Test\bin\Debug\Test.exe
Executing: "C:\Program Files (x86)\CodeBlocks\cb_console_runner.exe" "C:\Users\jfedin\Downloads\Test\bin\Debug\Test.exe" Something (in C:\Users\jfedin\Downloads\Test\.)
```

main.c [Test] - Code::Blocks 16.01

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Debug <global> main(void) : int

Management x test.c x main.c x

Projects S Workspace Test Sources

```
4 int main(void)
5 {
6     int number = 0;           // A variable of type int initialized to 0
7     int *pnumber = NULL;      // A pointer that can point to type int
8
9     number = 10;
10    printf("number's address: %p\n", &number);           // Output the address
11    printf("number's value: %d\n\n", number);             // Output the value
12
13    pnumber = &number;           // Store the address of number in pnumber
14
15    printf("pnumber's address: %p\n", (void*)&pnumber);   // Output the address
16    printf("pnumber's size: %d bytes\n", sizeof(pnumber)); // Output the size
17    printf("pnumber's value: %p\n", pnumber);             // Output the value (an
18    printf("value pointed to: %d\n", *pnumber);           // Value at the address
19    return
20
21 }
22
```

C:\Users\jfedin\Downloads\Test\bin\Debug\Test.exe Something

```
number's address: 0060FF0C
number's value: 10

pnumber's address: 0060FF08
pnumber's size: 4 bytes
pnumber's value: 0060FF0C
value pointed to: 10
```

Process returned 0 (0x0) execution time : 0.020 s
Press any key to continue.

Search results x Cccc x Bu

Checking for existence: C:\Users
Executing: "C:\Program Files (x86

{LP} LearnProgramming academy

C HOW TO PROGRAM, PEARSON EDUCATION, INC., TIM BUCHALKA'S - LP

Assignment

Requirements

- in this challenge, you are going to learn how to create, initialize, assign, and access a pointer
- write a program that creates an integer variable with a hard-coded value. Assign that variable's address to a pointer variable
- display as output the address of the pointer, the value of the pointer, and the value of what the pointer is pointing to.

References