8.2 Pointer basics

Pointer variables

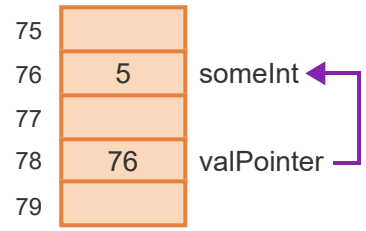
A ***pointer*** is a variable that holds a memory address, rather than holding data like most variables. A pointer has a data type, and the data type determines what type of address is stored in the pointer. Ex: An integer pointer contains a memory address of an integer, and a double pointer contains an address of a double. A pointer is declared by including * before the pointer's name. Ex: `int* maxItemPointer` declares an integer pointer named maxItemPointer.

Typically, a pointer is initialized with another variable's address. The ***reference operator*** (&) obtains a variable's address. Ex: `&someVar` returns the memory address of variable someVar. When a pointer is initialized with another variable's address, the pointer "points to" the variable.

PARTICIPATION ACTIVITY 8.2.1: Assigning a pointer with an address.

1 2 3 4 2x speed

```
int main() {
   int someInt;
   int* valPointer;

   someInt = 5;
   cout << "someInt address is " << &someInt << endl;

   valPointer = &someInt;
   cout << "valPointer is " << valPointer << endl;

   return 0;
}
```



someInt address is 76
valPointer is 76

valPointer is assigned with the memory address of someInt, so valPointer points to someInt.

Feedback?

Printing memory addresses

The examples in this material show memory addresses using decimal numbers for simplicity. Outputting a memory address is likely to display a hexadecimal value like 006FF978 or 0x7ffc3ae4f0e4. Hexadecimal numbers are base 16, so the values use the digits 0-9 and letters A-F.

PARTICIPATION ACTIVITY 8.2.2: Declaring and initializing a pointer.

1) Declare a double pointer called sensorPointer.

Check Show answer

Answer

double* sensorPointer;

The asterisk * must be placed variable name to make the variable a pointer.

2) Output the address of the double variable sensorVal.

```
cout << ;
```

Check Show answer

Answer

&sensorVal

The reference operator & obtains a variable's address.

3) Assign sensorPointer with the variable sensorVal's address. In other words, make sensorPointer point to sensorVal.

Check Show answer

Answer

sensorPointer = &sensorVal;

The reference operator & obtains a variable's address. Pointers are often initialized with a variable's address.

Feedback?

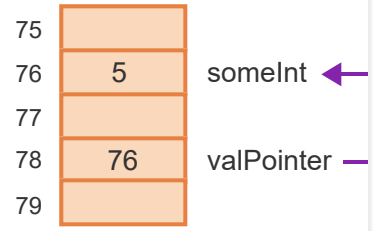
Dereferencing a pointer

The ***dereference operator*** (*) is prepended to a pointer variable's name to retrieve the data to which the pointer variable points. Ex: If valPointer points to a memory address containing the integer 123, then `cout << *valPointer;` dereferences valPointer and outputs 123.

PARTICIPATION ACTIVITY 8.2.3: Using the derefeence operator.

1 2 3 4 2x speed

```
int main() {
   int someInt;
   int* valPointer;

   someInt = 5;
   cout << "someInt address is " << &someInt << endl;

   valPointer = &someInt;
   cout << "valPointer is " << valPointer << endl;

   cout << "*valPointer is " << *valPointer << endl;

   *valPointer = 10;   // Changes someInt to 10

   cout << "someInt is " << someInt << endl;
   cout << "*valPointer is " << *valPointer << endl;

   return 0;
}
```



someInt address is 76
valPointer is 76

someInt is located in memory at address 76, and valPointer points to someInt.

Feedback?

PARTICIPATION ACTIVITY 8.2.4: Dereferencing a pointer.

Refer to the code below.

```
char userLetter = 'B';
char* letterPointer;
```

1) What line of code makes letterPointer point to userLetter?

- `letterPointer = userLetter;`
- `*letterPointer = &userLetter;`
- `letterPointer = &userLetter;`

Correct

letterPointer is assigned with the address of userLetter by using the reference operator &.

2) What line of code assigns the variable outputLetter

Correct

with the value letterPointer points to?

- ○ `outputLetter = letterPointer;`
- ◉ `outputLetter = *letterPointer;`
- ○ `someChar = &letterPointer;`

The dereference operator * gets the value letterPointer points to. If letterPointer points to userLetter, then outputLetter is now 'B'.

3) What does the code output?

```
letterPointer = &userLetter;
userLetter = 'A';
*letterPointer = 'C';
cout << userLetter;
```

- ○ A
- ○ B
- ◉ C

Correct

letterPointer points to userLetter, so changing *letterPointer to 'C' also changes userLetter to 'C'.

Feedback?

Null pointer

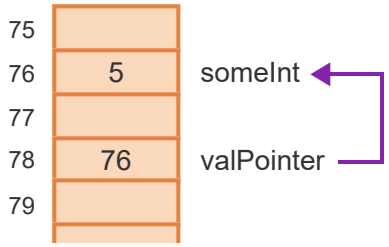
When a pointer is declared, the pointer variable holds an unknown address until the pointer is initialized. A programmer may wish to indicate that a pointer points to "nothing" by initializing a pointer to null. ***Null*** means "nothing". A pointer that is assigned with the keyword ***nullptr*** is said to be null. Ex: `int *maxValPointer = nullptr;` makes maxValPointer null.

In the animation below, the function PrintValue() only outputs the value pointed to by valuePointer if valuePointer is not null.

PARTICIPATION ACTIVITY 8.2.5: Checking to see if a pointer is null.

1 2 3 4 5 ◀ ☑ 2x speed

```
void PrintValue(int* valuePointer) {
   if (valuePointer == nullptr) {
      cout << "Pointer is null" << endl;
   }
   else {
      cout << *valuePointer << endl;
   }
}
```



```
int main() {
   int someInt = 5;
   int* valPointer = nullptr;

   PrintValue(valPointer);
   valPointer = &someInt;
   PrintValue(valPointer);

   return 0;
}
```

80

Pointer is null
5

The if statement is false because valuePointer is no longer null. valuePointer points to the value 5, so 5 is output.

Feedback?

Null pointer

The nullptr keyword was added to the C++ language in version C++11. Before C++11, common practice was to use the literal 0 to indicate a null pointer. In C++'s predecessor language C, the macro NULL is used to assign a null pointer.

PARTICIPATION ACTIVITY 8.2.6: Null pointer.

Refer to the animation above.

1) The code below outputs 3.

```
int numSides = 3;
int *valPointer =
&numSides;
PrintValue(valPointer);
```

- (●) True
- ( ) False

Correct

valPointer points to numSides, which is 3. PrintValue() prints the value pointed to by valPointer unless valPointer is null.

2) The code below outputs 5.

```
int numSides = 5;
int *valPointer =
&numSides;
valPointer = nullptr;
PrintValue(valPointer);
```

Correct

valPointer points to numSides initially, but then is assigned nullptr. PrintValue() prints "Pointer is null" when valPointer is null.

- True
- False

3) The code below outputs 7.

```
int numSides = 7;
int *valPointer =
nullptr;
cout << *valPointer;
```

- True
- False

Correct

The output statement dereferences valPointer, which is null. Dereferencing a null pointer causes the program to crash.

Feedback?

Common pointer errors

A number of common pointer errors result in syntax errors that are caught by the compiler or runtime errors that may result in the program crashing.

Common syntax errors:

- A common error is to use the dereference operator when initializing a pointer. Ex: `*valPointer = &maxValue;` is a syntax error because *valPointer is referring to the value pointed to, not the pointer itself.
- A common error when declaring multiple pointers on the same line is to forget the * before each pointer name. Ex: `int* valPointer1, valPointer2;` declares valPointer1 as a pointer, but valPointer2 is declared as an integer because no * exists before valPointer2. Good practice is to declare one pointer per line to avoid accidentally declaring a pointer incorrectly.

Common runtime errors:

- A common error is to use the dereference operator when a pointer has not been initialized. Ex: `cout << *valPointer;` may cause a program to crash if valPointer stores an unknown address or an address the program is not allowed to access.
- A common error is to dereference a null pointer. Ex: If valPointer is null, then `cout << *valPointer;` causes the program to crash. A pointer should always contain a valid address before the pointer is dereferenced.

PARTICIPATION ACTIVITY 8.2.7: Common pointer errors.

1 2 3 4 2x speed

syntax errors

```
int someInt = 5;
int* valPointer;

*valPointer = &someInt;

valPointer = &someInt;
```

✖ int value cannot be assigned int*

◀ remove *

```
int* valPointer1, valPointer2;
valPointer1 = nullptr;
valPointer2 = nullptr;

int* valPointer1;
int* valPointer2;
```

✖ int cannot be assigned nullptr

◀ declare on separate lines

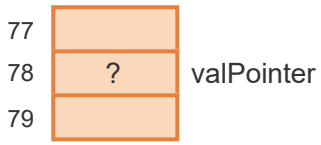
runtime errors

```
int* valPointer;
*valPointer = 4;
```

✖ dereferencing unknown address

```
int someInt = 2;
valPointer = &someInt;
*valPointer = 4;
```

◀ initialize pointer before dereferencing



valPointer is not initialized, so valPointer contains an unknown address. Dereferencing an unknown may cause a runtime error.

Feedback?

PARTICIPATION ACTIVITY 8.2.8: Common pointer errors.

Indicate if each code segment has a syntax error, runtime error, or no error.

1)

```
char* newPointer;
*newPointer = 'A';
cout << *newPointer;
```

- ○ syntax error
- ◉ runtime error
- ○ no errors

Correct

newPointer contains an unknown address when declared and is not initialized. Dereferencing an uninitialized pointer may cause the program to crash.

2)

Correct

```
char* valPointer1,
*valPointer2;
valPointer1 = nullptr;
valPointer2 = nullptr;
```

valPointer1 and valPointer2 are both declared as char pointers because both variables have asterisks preceding the variable names. However, good practice is to declare pointers on separate lines.

- ○ syntax error
- ○ runtime error
- ◉ no errors

3)

```
char someChar = 'Z';
char* valPointer;
*valPointer = &someChar;
```

- ◉ syntax error
- ○ runtime error
- ○ no errors

Correct

A pointer cannot be assigned with an address when using the reference operator *. The correct way to assign valPointer: `valPointer = &someChar;`

4)

```
char* newPointer =
nullptr;
char someChar = 'A';
*newPointer = 'B';
```

- ○ syntax error
- ◉ runtime error
- ○ no errors

Correct

newPointer is initialized with nullptr when dereferenced and assigned 'B'. Dereferencing a null pointer causes the program to crash.

Feedback?

Two pointer declaration styles

Some programmers prefer to place the asterisk next to the variable name when declaring a pointer. Ex: ***`int *valPointer;`****. The style preference is useful when declaring multiple pointers on the same line:* ***`int *valPointer1, *valPointer2;`****. Good practice is to use the same pointer declaration style throughout the code: Either* ***`int* valPointer`*** *or* ***`int *valPointer`****.*

This material uses the style ***`int* valPointer`*** *and always declares one pointer per line to avoid accidentally declaring a pointer incorrectly.*

Advanced compilers can check for common errors

Some compilers have advanced code analysis capabilities to catch some runtime errors at compile time. Ex: The compiler may issue a warning if the compiler detects a null pointer is being dereferenced. An advanced compiler can never catch all runtime errors because a potential runtime error may depend on user input, which is unknown at compile time.

zyDE 8.2.1: Using pointers.

The following provides an example (not useful other than for learning) of assignin address of variable vehicleMpg to the pointer variable valPointer.

1. Run and observe that the two output statements produce the same output.
2. Modify the value assigned to *valPointer and run again.
3. Now uncomment the statement that assigns vehicleMpg. PREDICT whethe statements will print the same output. Then run and observe the output. Did correctly?

Load default template... Run

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     double vehicleMpg;
6     double* valPointer = nullptr;
7
8     valPointer = &vehicleMpg;
9
10    *valPointer = 29.6; // Assigns the number
11                        // POINTED TO by valPo
12
13    // vehicleMpg = 40.0;   // Uncomment this
14
15    cout << "Vehicle MPG = " << vehicleMpg <<
16    cout << "Vehicle MPG = " << *valPointer <<
17
18    return 0;
19 }
20
```

Feedback?

CHALLENGE

ACTIVITY | 8.2.1: Printing with pointers.

If the input is negative, make numItemsPointer be null. Otherwise, make numItemsPointer point to numItems and multiply the value to which numItemsPointer points by 10. Ex: If the user enters 99, the output should be:

```
Items: 990
```

```
6    int numItems;
7
8    cin >> numItems;
9
10   /* Your solution goes here  */
11   if (numItems<0){
12      numItemsPointer = nullptr;
13   }else{
14      numItemsPointer = &numItems;
15      *numItemsPointer =numItems*10;
16   }
17
18   if (numItemsPointer == nullptr) {
19      cout << "Items is negative" << endl;
20   }
21   else {
22      cout << "Items: " << *numItemsPointer << endl;
23   }
24
25   return 0;
26 }
```

Run ✓ All tests passed

✓ Testing with numItems = 99

Your output `Items: 990`

✓ Testing numItemsPointer is assigned numItem's address.

Your output `numItemsPointer is assigned numItem's address.`

✓ Testing with numItems = -1

Your output `Items is negative`

Feedback?