CS256

Chapter 9: Pointers

Hanoi University of Science and Technology

Slide contents are mainly based on the following documents:

- Tony Gaddis, Starting out with C++, Eighth Edition, Pearson, 2015
- Tony Gaddis, Power Points Slides:
  `https://www.umsl.edu/~antognolij/gaddis.html`

What we have learned in CS255

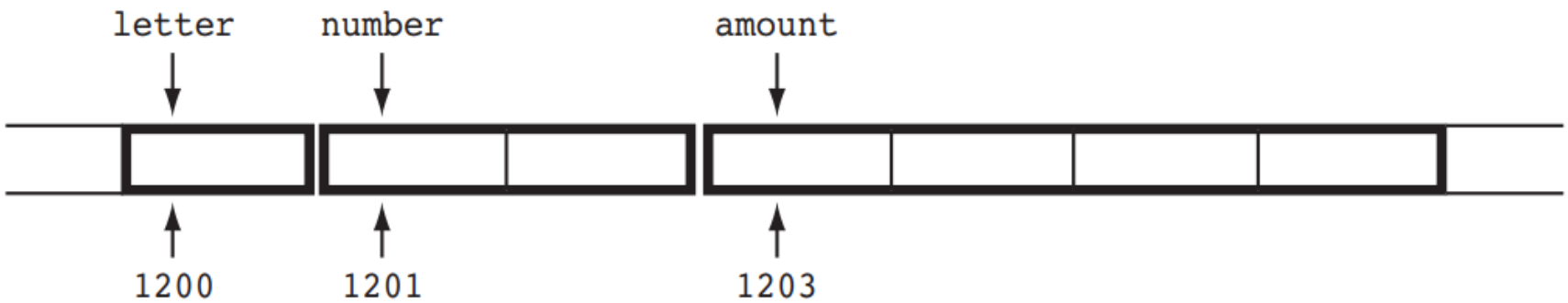
Basic C++:

- `if / else` statement
- `for, while, do ... while ..`
- functions
- arrays
- files
- ...

Getting the Address of a Variable

- Each byte of memory has a unique address.
- A variable's address: the **address of the first byte** allocated to that variable.

```
char letter;
short number;
float amount;
```



Getting the Address of a Variable

- Use the address operator & to get the address of a variable:

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

- **Note**: Do not confuse the address operator with the & symbol used when defining a reference variable

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

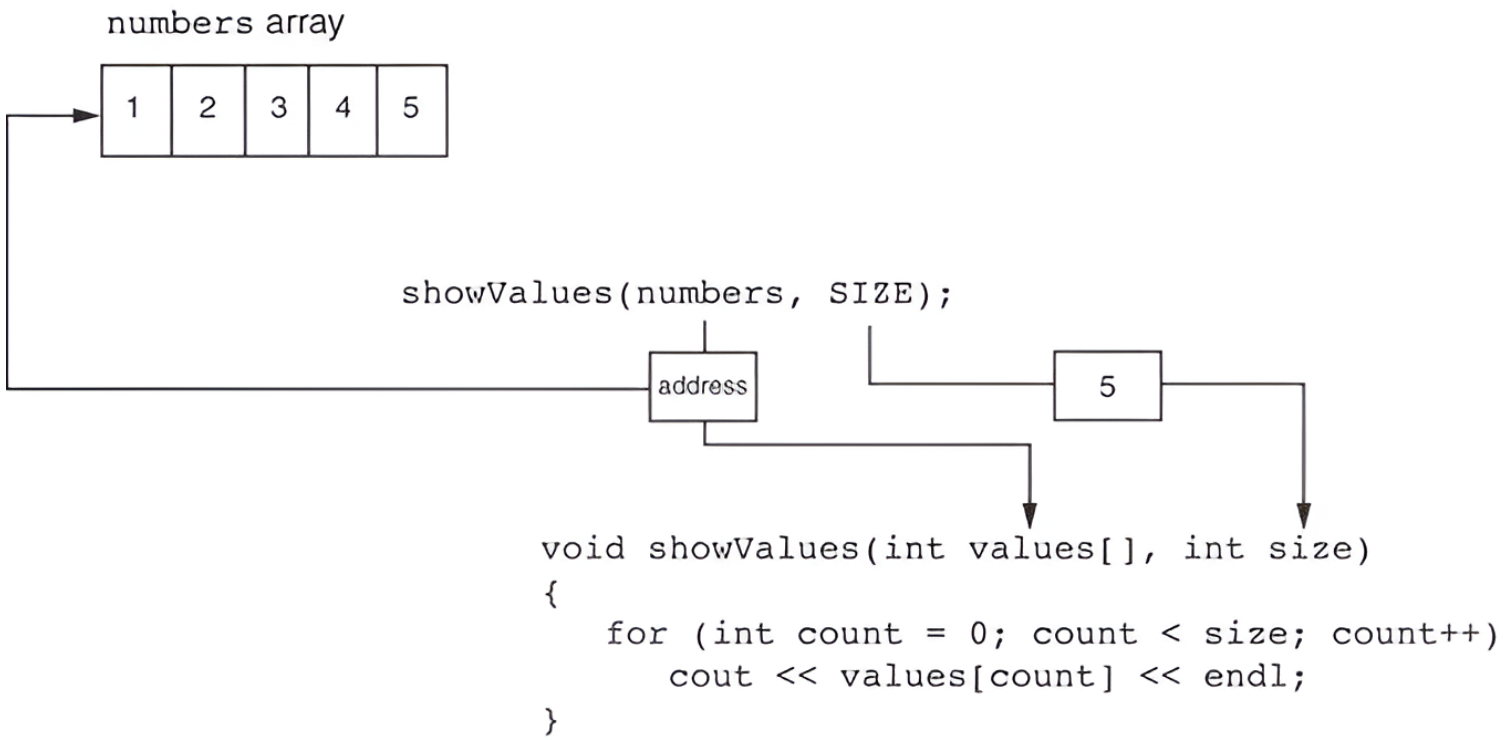
- For example, suppose we use this statement to pass the array `numbers` to the `showValues` function:

```
showValues(numbers, SIZE);

void showValues(int values[], int size) {
    for (int i = 0; i < size; i++)
        cout << values[i] << "\n";
}
```

Something Like Pointers: Arrays

- The `values` parameter points to the `numbers` array.
- C++ automatically stores the address of `numbers` in the `values` parameter.



Something Like Pointers: Reference Variables

- Suppose we have this function:

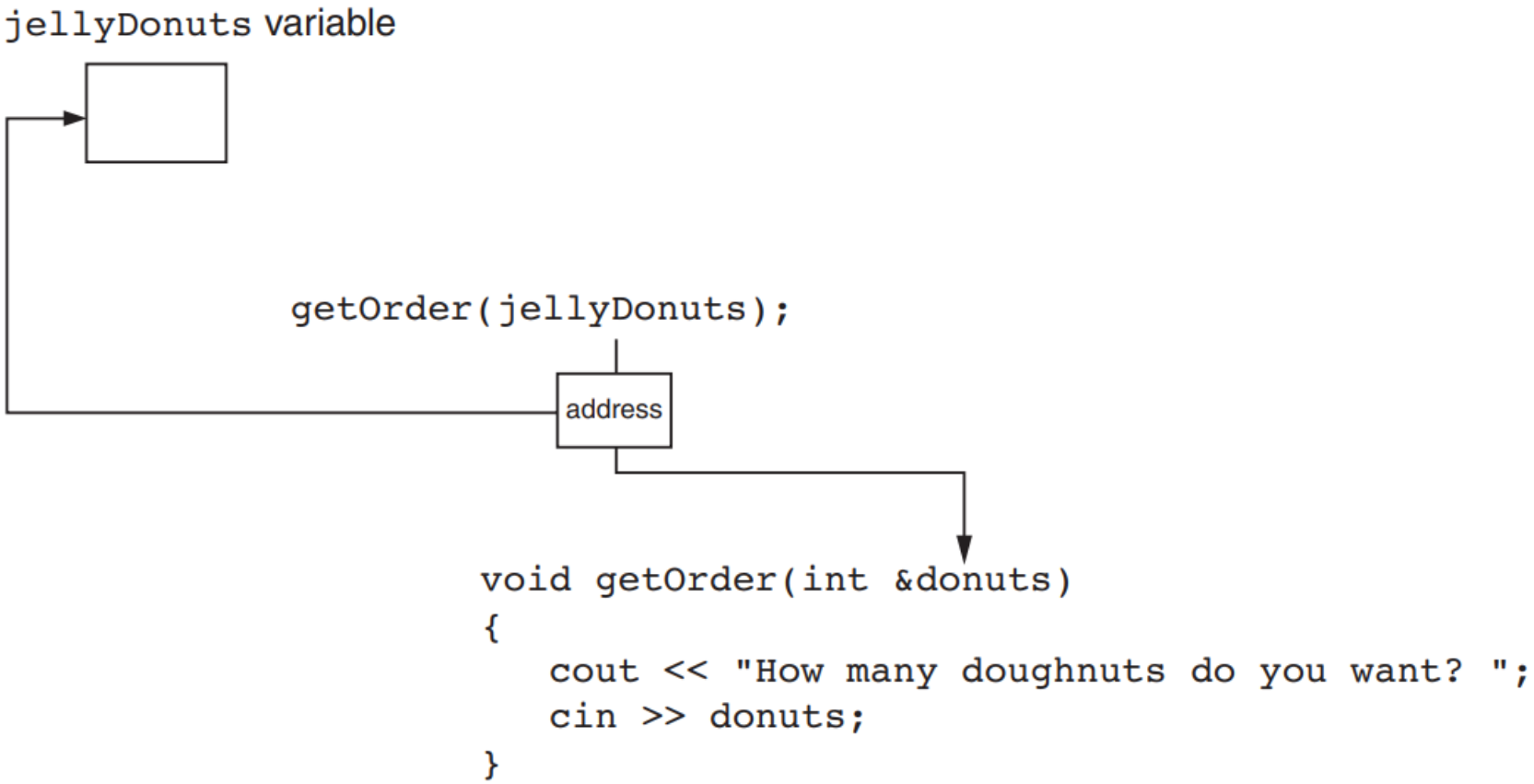
```
void getOrder(int &donuts) {
     cout << "How many doughnuts do you want? ";
     cin >> donuts;
}
```

- And we call it with this code:

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

Something Like Pointers: Arrays

- C++ automatically stores the address of `jellyDonuts` in the `donuts` parameter.
- The `donuts` parameter, in the `getOrder` function, points to the `jellyDonuts` variable.



Pointer variables

- Pointer variables are yet another way using a memory address to work with a piece of data.
- Definition:

```
int *intptr;
```

Meaning: `intptr` can hold the address of an int

- Spacing in definition does not matter:

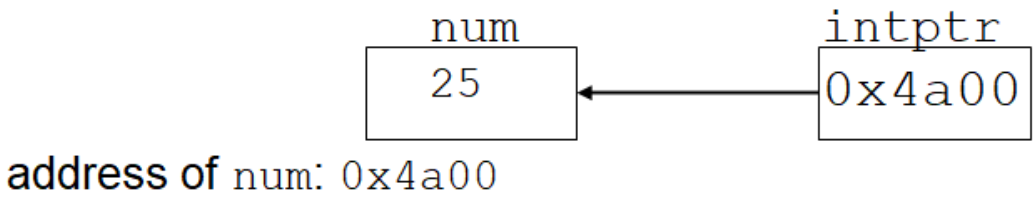
```
int * intptr;   // same as above
int*  intptr;   // same as above
```

Assigning an Address to a Pointer

- Assigning an address to a pointer variable:

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

- **Memory layout:**



Assigning an Address to a Pointer

Program 9-2

```
1  // This program stores the address of a variable in a pointer.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7       int x = 25;           // int variable
8       int *ptr = nullptr;   // Pointer variable, can point to an int
9
10      ptr = &x;             // Store the address of x in ptr
11      cout << "The value in x is " << x << endl;
12      cout << "The address of x is " << ptr << endl;
13      return 0;
14 }
```

Program Output

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

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.
```

The Indirection Operator

Program 9-3

```
1  // This program demonstrates the use of the indirection operator.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7       int x = 25;           // int variable
8       int *ptr = nullptr;   // Pointer variable, can point to an int
9
10      ptr = &x;             // Store the address of x in ptr
11
12      // Use both x and ptr to display the value in x.
13      cout << "Here is the value in x, printed twice:\n";
14      cout << x << endl;    // Displays the contents of x
15      cout << *ptr << endl; // Displays the contents of x
16
17      // Assign 100 to the location pointed to by ptr. This
18      // will actually assign 100 to x.
19      *ptr = 100;
```

(program continues)

The Indirection Operator

Program 9-3 *(continued)*

```
20
21      // Use both x and ptr to display the value in x.
22      cout << "Once again, here is the value in x:\n";
23      cout << x << endl;     // Displays the contents of x
24      cout << *ptr << endl;  // Displays the contents of x
25      return 0;
26  }
```

Program Output

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

The Relationship Between Arrays and Pointers

- The array name is the starting address of the array.

```
int vals[] = {4, 7, 11};
cout << vals;      // displays address: 0x4a00
cout << vals[0];   // displays 4
```

- Array name can be used as a **pointer constant**:

```
cout << *vals;     // displays 4
```

- A pointer can be used as an array name:

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

The Relationship Between Arrays and Pointers

Program 9-5

```
1  // This program shows an array name being dereferenced with the *
2  // operator.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8       short numbers[] = {10, 20, 30, 40, 50};
9
10      cout << "The first element of the array is ";
11      cout << *numbers << endl;
12      return 0;
13 }
```

Program Output

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

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
```

- **Note:** Must use `()` as shown in the expressions.

Array Access

- Array elements can be accessed in many ways:
  - 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;`
- **Conversion:** `vals[i]` is equivalent to `*(vals + i)`.

Pointer Arithmetic

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`

Pointer Arithmetic

```
5  int main()
6  {
7       const int SIZE = 8;
8       int set[SIZE] = {5, 10, 15, 20, 25, 30, 35, 40};
9       int *numPtr = nullptr;  // Pointer
10      int count;              // Counter variable for loops
11
12      // Make numPtr point to the set array.
13      numPtr = set;
14
15      // Use the pointer to display the array contents.
16      cout << "The numbers in set are:\n";
17      for (count = 0; count < SIZE; count++)
18      {
19           cout << *numPtr << " ";
20           numPtr++;
21      }
22
23      // Display the array contents in reverse order.
24      cout << "\nThe numbers in set backward are:\n";
25      for (count = 0; count < SIZE; count++)
26      {
27           numPtr--;
28           cout << *numPtr << " ";
29      }
30      return 0;
31 }
```

Program Output

```
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
```

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 is **not** the same as comparing contents:

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

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

Pointers as Function Parameters

- A pointer can be a parameter. It works like a reference variable to allow changes to the argument from within the function.
- Requires an asterisk * on the parameter in the prototype and heading:

```
void getNum(int *ptr);
```

- Requires an asterisk * in the body to dereference the pointer:

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

- Pass the address as an argument:

```
getNum(&num);
```

Pointers as Function Parameters

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

// Function call:
int num1 = 2, num2 = -3;
swap(&num1, &num2);
```

Pointers as Function Parameters

Program 9-11

```
1  // This program uses two functions that accept addresses of
2  // variables as arguments.
3  #include <iostream>
4  using namespace std;
5
6  // Function prototypes
7  void getNumber(int *);
8  void doubleValue(int *);
9
10 int main()
11 {
12      int number;
13
14      // Call getNumber and pass the address of number.
15      getNumber(&number);
16
17      // Call doubleValue and pass the address of number.
18      doubleValue(&number);
19
20      // Display the value in number.
21      cout << "That value doubled is " << number << endl;
22      return 0;
23 }
24
```

Pointers as Function Parameters

```
25  //*********************************************************
26  // Definition of getNumber. The parameter, input, is a pointer. *
27  // This function asks the user for a number. The value entered  *
28  // is stored in the variable pointed to by input.               *
29  //*********************************************************
30
31  void getNumber(int *input)
32  {
33      cout << "Enter an integer number: ";
34      cin >> *input;
35  }
36
37  //*********************************************************
38  // Definition of doubleValue. The parameter, val, is a pointer. *
39  // This function multiplies the variable pointed to by val by   *
40  // two.                                                         *
41  //*********************************************************
42
43  void doubleValue(int *val)
44  {
45      *val *= 2;
46  }
```

Program Output with Example Input Shown in Bold

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

Exercises

- Programming Challenges: 1, 5, 10, 11, 12.
- 1. Write a program according to the requirements:
  - Allow the user to input a string from the keyboard.
  - Use a pointer to reverse the string and print the reversed string.
- 2. Find the largest number in an array using a pointer instead of array indices.
- 3. Write a function that takes an array and an integer n as arguments and returns the number of occurrences of n in the array.
- 4. Sort an integer array in ascending order by using pointers.