Chapter 12

Pointers (Part 1)

12.1 Pointer Basics

12.1.1 What Are Pointers?

A *pointer* is a variable that contains a **memory address**.
Very often this address is the location of another variable.

The general form of a pointer variable declaration in C++ is:

type ***variable-name*;

- *type* — the pointer´s base type.
  It must be a valid C++ type.
- *variable-name* — the name of the pointer variable
- ***** — the “at address” operator
  Returns the **value** of the variable located at the address specified by its operand.

Examples of pointers:

```
int *int_pointer;     // pointer to integer

float *float_pointer; // pointer to float

char *str;  // pointer to a char or string

int **ptrptr; // pointer to a pointer
```

12.1.2 Pointer Operators

There are **two special operators** that are used with pointers:

- & : “address of …” operator

  A unary operator which returns the **memory address** of its operand.

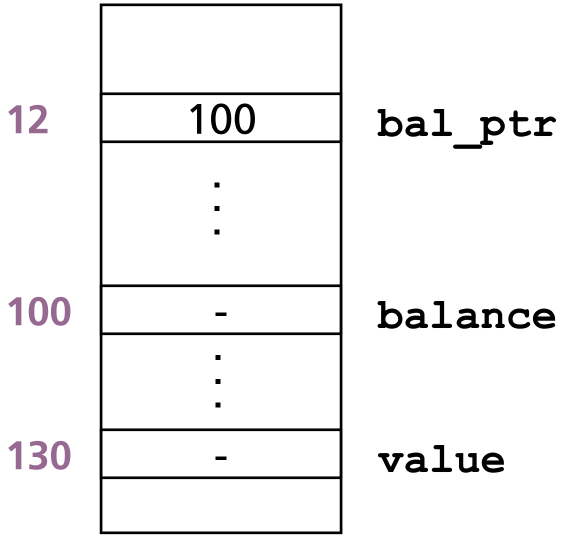
- * :“**value at address …**” operator

  A unary operator which returns the value of the variable located at the address specified by its operand.

Example:

```
int balance;
int *balptr;
```

```
balptr = &balance;
```

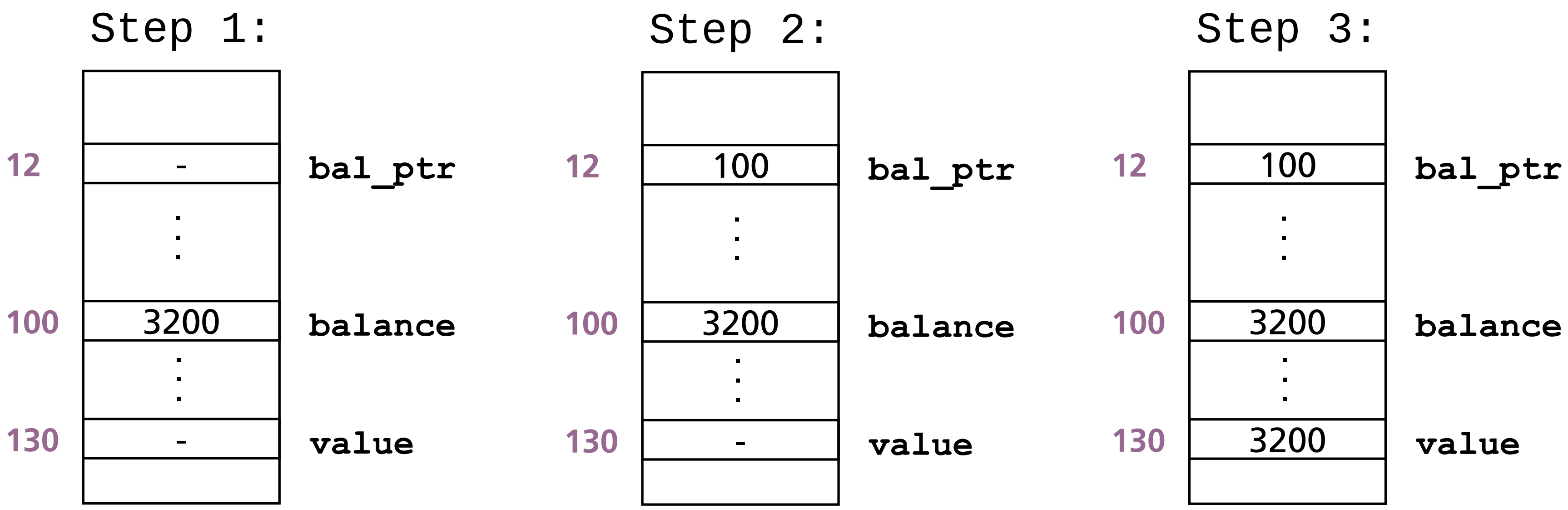


Example:

```
int balance, value;
int *balptr;

balance = 3200;       // Step 1
balptr = &balance;    // Step 2

value = *balptr;      // Step 3
```



Step 1:

12	-	bal_ptr
	⋮	
100	3200	balance
	⋮	
130	-	value

Step 2:

12	100	bal_ptr
	⋮	
100	3200	balance
	⋮	
130	-	value

Step 3:

12	100	bal_ptr
	⋮	
100	3200	balance
	⋮	
130	3200	value

Importance of the Base Type

How does C++ know how many bytes to copy into **`value`** from the address pointed to by **`balptr`**?

How does the compiler know the proper number of bytes for any assignment using a pointer?

Answer: The **base type** of the pointer determines the type of data that the ***compiler assumes*** the pointer is pointing to.

The following code fragment is incorrect:

```
int *int_ptr; double f;
int_ptr = &f; // ERROR
```

Technically correct, but not recommended (using a **type cast** operator):

```
int_ptr = (int *) &f;
```

Example for why to be careful about type casts with pointers:

```
void main()
{
  double x, y;
  int *ptr;

  x = 123.23;
  ptr = (int *) &x; // use cast to assign
                  // double* to int*

  y = *ptr;    // What will this do?
  cout << y;   // What will this print?
}
```

12.2 Working with Pointers

12.2.1 Assigning Values Through Pointers

- Pointers can be used on the left side of assignment statements.

  The following code fragment assigns a value to the location pointed to by the pointer.

```
int *ptr;
*ptr = 101;
```

“At the location pointed to by p, assign the value 101.”

- Increment and decrement operations work on pointers, too.

```
(*ptr)++;
```

“At the location pointed to by p, increment the value by 1.”

Example program for **assigning values through pointers**:

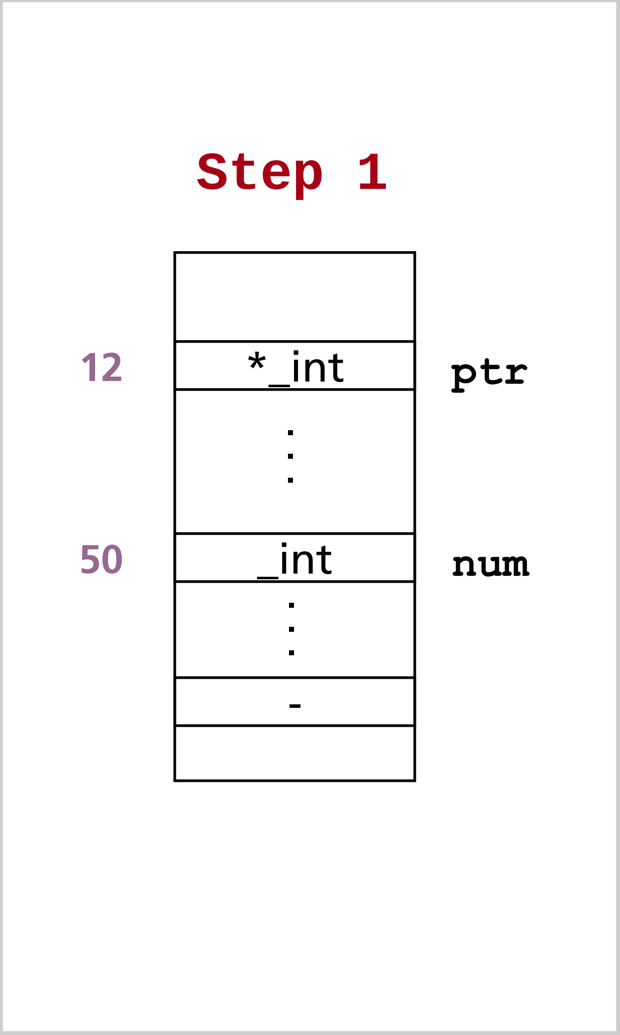
```
void main()
{
  int *ptr, num;       // 1

  ptr = &num;          // 2

  *ptr = 100;          // 3
  cout << num << ‘ ‘;

  (*ptr)++;            // 4
  cout << num << ‘ ‘;

  (*ptr)*2;            // 5
  cout << num << ‘\n’;
}
```



Example program for **assigning values through pointers**:

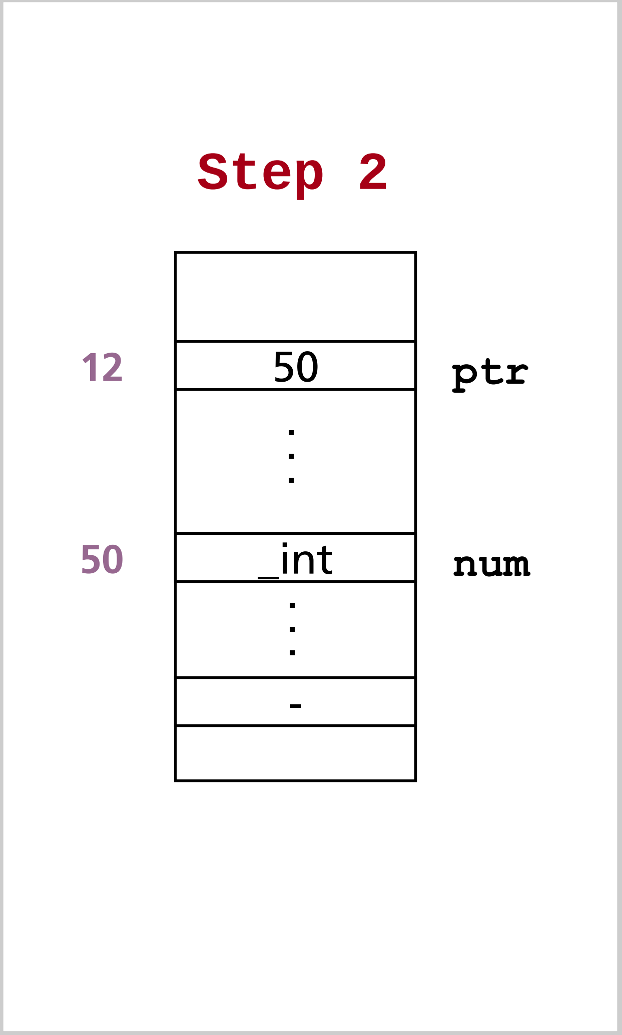
```
void main()
{
  int *ptr, num;        // 1

  ptr = &num;           // 2

  *ptr = 100;           // 3
  cout << num << ‘ ‘;

  (*ptr)++;             // 4
  cout << num << ‘ ‘;

  (*ptr)*2;             // 5
  cout << num << ‘\n’;
}
```



Example program for **assigning values through pointers**:

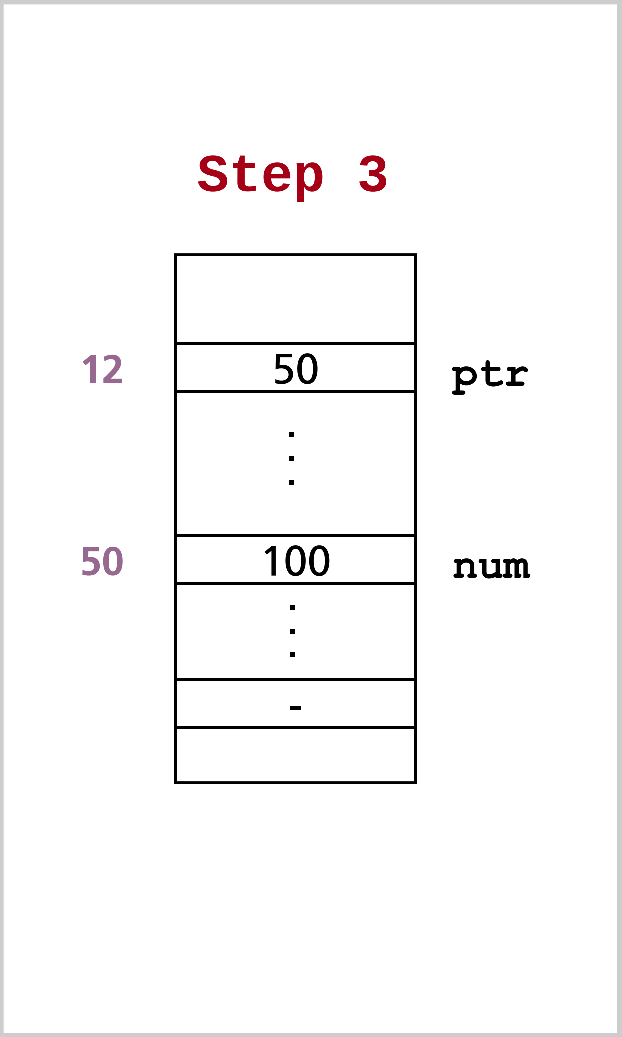
```
void main()
{
  int *ptr, num;       // 1

  ptr = &num;          // 2

  *ptr = 100;          // 3
  cout << num << ‘ ‘;

  (*ptr)++;            // 4
  cout << num << ‘ ‘;

  (*ptr)*2;            // 5
  cout << num << ‘\n’;
}
```



Step 3

Example program for **assigning values through pointers**:

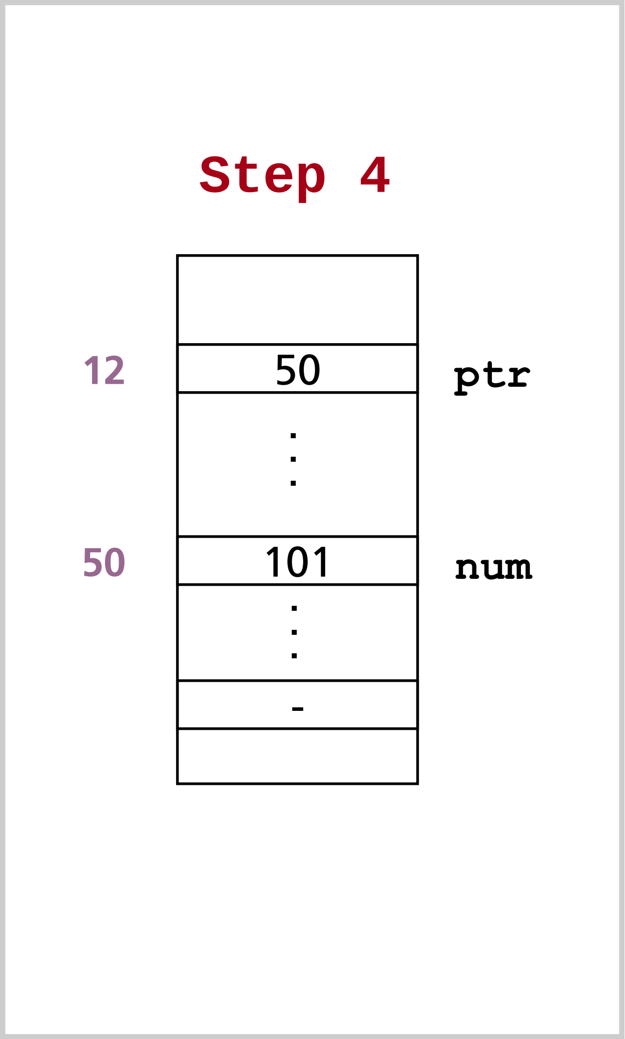
```
void main()
{
  int *ptr, num;     // 1

  ptr = &num;        // 2

  *ptr = 100;        // 3
  cout << num << ‘ ‘;

  (*ptr)++;          // 4
  cout << num << ‘ ‘;

  (*ptr)*2;          // 5
  cout << num << ‘\n’;
}
```



Example program for **assigning values through pointers**:

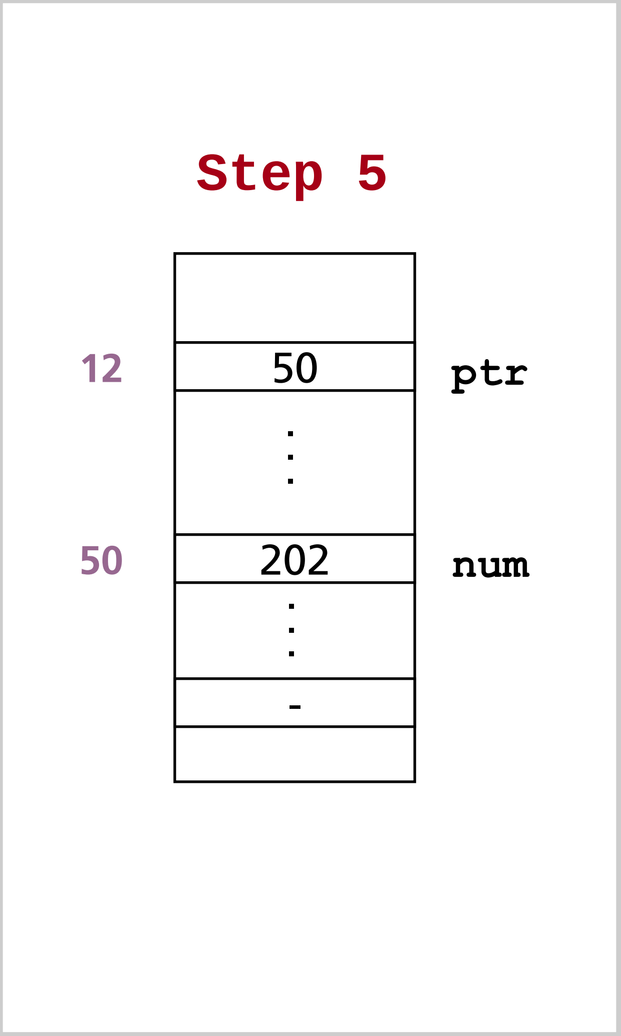
```
void main()
{
  int *ptr, num;          // 1

  ptr = &num;             // 2

  *ptr = 100;             // 3
  cout << num << ‘ ‘;

  (*ptr)++;               // 4
  cout << num << ‘ ‘;

  (*ptr)*2;               // 5
  cout << num << ‘\n’;
}
```



12.2.2 Pointer Expressions

Pointers can be used in most C++ expressions.

Keep in mind to use **parentheses** around pointer expressions.

Pointer Arithmetic

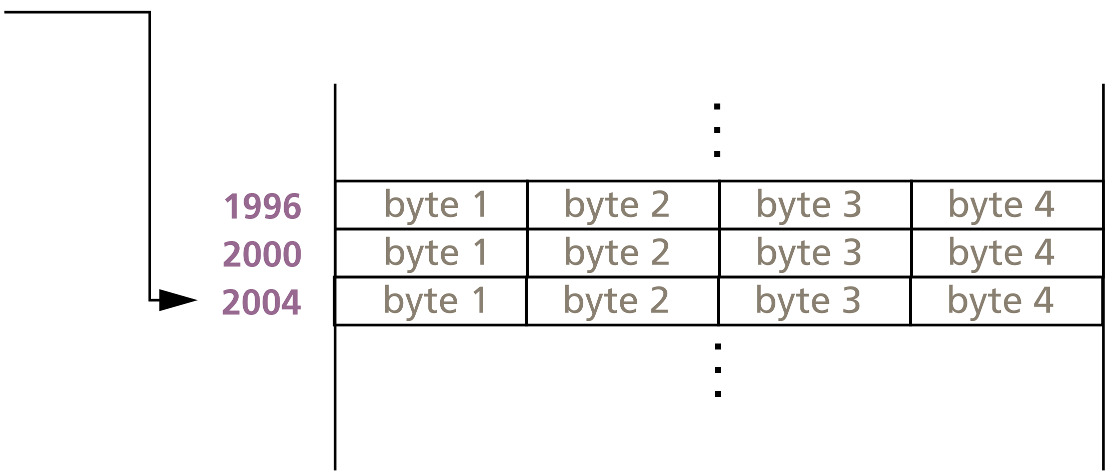
Only four arithmetic operators can be used on pointers:

- ++
- --
- +
- -

Example: (assuming 32-bit integers)

```
int *p1; // assume: p1 == 2000

p1++;
```



		⋮		
1996	byte 1	byte 2	byte 3	byte 4
2000	byte 1	byte 2	byte 3	byte 4
2004	byte 1	byte 2	byte 3	byte 4
		⋮		

- Integers can be added or subtracted from pointers:
- You can subtract pointers of the same type from one another.
  You can not add pointers! However, you can add **int** numbers to pointers:

```
void main()
{
  int i[10], *intPtr;
  double d[10], *doublePtr;
  int x;

  intPtr = i; // i_ptr points to first element of i
  doublePtr = d; // f_ptr points to first element of f

  for(x=0; x < 10; x++)
    cout << intPtr + x;
    cout << ‘ ‘;
    cout << doublePtr + x;
    cout << endl;
}
```

Output of the example program:

The addresses of the array elements:

4 bytes **int**   8 bytes **double**

```
0xeffffd9c 0xeffffd48
0xeffffda0 0xeffffd50
0xeffffda4 0xeffffd58
0xeffffda8 0xeffffd60
0xeffffdac 0xeffffd68
0xeffffdb0 0xeffffd70
0xeffffdb4 0xeffffd78
0xeffffdb8 0xeffffd80
0xeffffdbc 0xeffffd88
0xeffffdc0 0xeffffd90
```

If we want to see the values at these addresses, we have to use the "value at ..." operator (*):

```
void main()
{
   int i[3]={1,2,3}, *intPtr;
   double d[3]={1.1,2.2,3.3}, *doublePtr;
   int x;

   intPtr = i; // i_ptr points to first element of i
   doublePtr = d; // f_ptr points to first element of f

   for(x=0; x < 3; x++)
      cout << *(intPtr + x);
      cout << ’ ’;
      cout << *(doublePtr + x);
      cout << endl;
}
```

12.2.3 Pointer Comparisons

Pointers may be compared using relational operators, such as: !=, ==, <, and >.

```
void main()
{
  int num[10];
  int *start, *end;

  start = num;
  end = &num[9];

  while(start != end) {
    cout << “Enter a number: “;
    cin >> *start;
    start++;
} }
```

Pointer Comparisons (2): using pointer arithmetic

Pointers may be compared using relational operators, such as **!=**, **==**, **<**, and **>**.

```
void main()
{
  int num[10];
  int *start, *end;

  start = num;
  end = &num[9];

  while((end - start) > 0) {
    cout << “Enter a number: “;
    cin >> *start;
    start++;
} }
```

12.3 Pointers and Function Parameters

Back to Mine Sweeper:

```
void GetCoordinates(int &i, int &j);

...

void main()
{ ...
  int i, j; // local variables

  GetCoordinates(i, j);
  // Manipulates coordinates as a side effect
  ...
}
```

12.4 References

- G. Blank and R. Barnes, *The Universal Machine*, Boston, MA: WCB/ McGraw-Hill, 1998. Chapter 9.
- H. Schildt, C++ from the Ground Up, McGraw-Hill, Berkeley, CA, 1998. Chapter 6.