Brief Introduction to C++

Computer Vision 2018

P. Zanuttigh
Some slides M. Bowdoin, F. Bari, Pearson Education

What is C++ ?

- C++ is an enhanced version of the C language
- C++ adds support for Object-Oriented Programming (OOP) without sacrificing any of C’s power, elegance, or flexibility
- C++ was invented in 1979 by Bjarne Stroustrup at Bell Laboratories in Murray Hill, New Jersey, USA.

Object Oriented Programming

- OOP is a powerful way to approach the task of programming
- OOP encourages developers to decompose a problem into its constituent parts
- Each component becomes a self-contained object that contains its own instructions and data that relate to that object
- So, complexity is reduced and the programmer can manage larger programs
- All OOP languages, including C++, share three common defining traits:
  1. ***Encapsulation***: Binds together code and data
  2. ***Polymorphism***: Allows one interface, multiple methods
  3. ***Inheritance***: Provides hierarchical classification and permits reuse of common code and data

A Simple C++ Program

```
//include headers; these are modules that include functions that you may use in your program;
#include <iostream.h>

int main() {

cout << “Hello world!”;
return 0;
}
```

After you write a C++ program you compile it; that is, you run a program called **compiler** that checks whether the program follows the C++ syntax

- if it finds errors, it lists them
- If there are no errors, it translates the C++ program into a program in machine language which you can execute

The Main Function

- The *main* function is a part of every C++ program
- The parentheses after main indicate that main is a program building block called a function
- C++ programs typically consist of one or more functions and classes
- Exactly *one* function in every program *must* be named main
- C++ programs begin executing at function main, even if main is *not* the first function defined in the program
- The keyword *int* to the left of main indicates that main "returns" an integer (whole number) value.
  - A keyword is a word in code that is reserved by C++ for a specific use
  - For now, simply include the keyword int to the left of main in each of your programs

C++ Keywords (partial list)

- bool
- catch
- *delete*
- false
- friend
- inline
- namespace
- *new*
- operator
- private
- protected
- public
- *template*
- this
- throw
- true
- try
- using
- virtual
- wchar_t

Functions

- A left brace, {, must *begin* the body of every function.
- A corresponding right brace, }, must *end* each function’s body
- A statement instructs the computer to perform an action
- Together, the quotation marks and the characters between them are called a string, a character string or a string literal
- We refer to characters between double quotation marks simply as strings
  - White-space characters in strings are not ignored by the compiler
- Most C++ statements end with a semicolon (;), also known as the statement terminator
  - Preprocessing directives (like #include) do not end with a semicolon

Variable Declaration

type variable-name;

Meaning: variable <variable-name> will be a variable of type <type>

Where type can be:

□ int	//integer
□ double	//real number
□ char	//character

Example:

```
int a, b, c;
double x;
int sum;
char my-character;
```

Output Statements

cout << variable-name;

Meaning: print the value of variable <variable-name> to the user

cout << “any message “;

Meaning: print the message within quotes to the user

cout << endl;

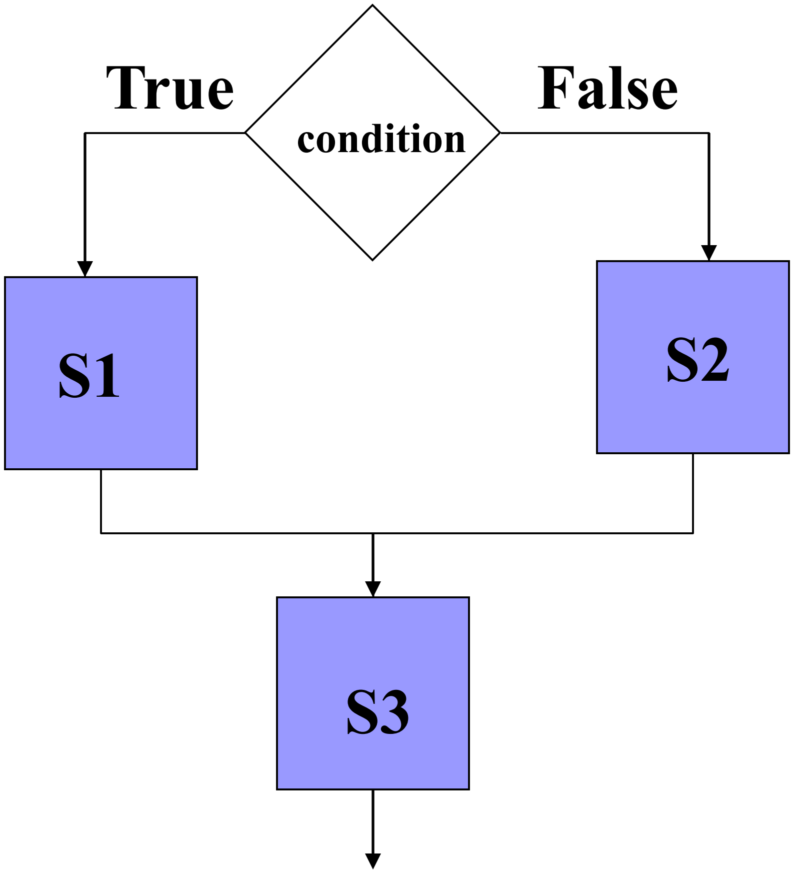
Meaning: print a new line

Example:

```
cout << a;
cout << b << c;
cout << “This is my character: “ << my-character << “ he he he”<< endl;
```

If Statements

```
if (condition) {
    S1;
}
else {
    S2;
}
S3;
```



Boolean conditions

..are built using

- Comparison operators

==	equal
!=	not equal
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal

- Boolean operators

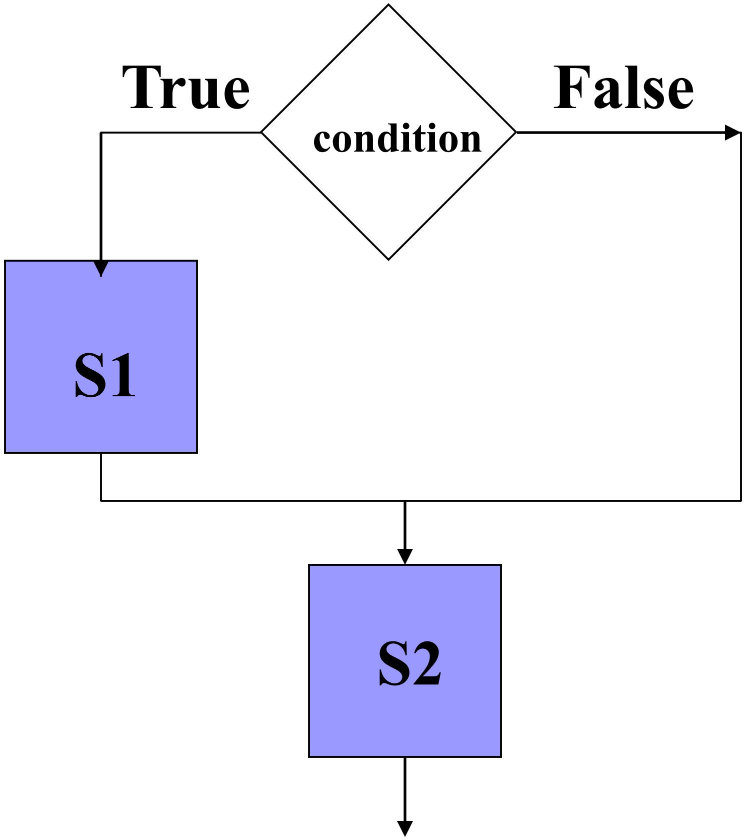
&&	and
||	or
!	not

If example

```
#include <iostream.h>

void main() {
int a,b,c;
cin >> a >> b >> c;

if (a <=b) {
        cout << “min is “ << a << endl;
        }
else {
        cout << “ min is “ << b << endl;
}
cout << “happy now?” << endl;
}
```

While Statements

```
while (condition) {
   S1;
}
S2;
```



While Example

```
//read 100 numbers from the user and output their sum
#include <iostream.h>

void main() {
int i, sum, x;
sum=0;
i=1;
while (i <= 100) {
    cin >> x;
    sum = sum + x;
    i = i+1;
}
cout << “sum is “ << sum << endl;
}
```

Additional Notes

- Indentation is for the convenience of the reader; compiler ignores all spaces and new line ;

- All statements ended by semicolon

- what follows after ***//*** on the same line is considered comment
- Comments containing more lines are enclosed in /* and */

- **Lower vs. upper case matters!!**
  - Void is different than void
  - Main is different that main

Preprocessor

- A preprocessing directive is a message to the C++ preprocessor
- Lines that begin with # are processed by the preprocessor before the program is compiled
- #include <iostream> notifies the preprocessor to include in the program the contents of the input/output stream header file <iostream>
  - This header is a file containing information used by the compiler when compiling any program that outputs data to the screen or inputs data from the keyboard using C++-style stream input/output.

Namespaces

- A namespace is a declarative region
- It localizes the names of identifiers to avoid name collisions
- The contents of new-style headers are placed in the **std** namespace
- A newly created class, function or global variable can put in an existing namespace, a new namespace, or it may not be associated with any namespace
  - In the last case the element will be placed in the global unnamed namespace.

Using Keyword

- using declarations that eliminate the need to repeat the std:: prefix as we did in earlier programs
- Once we insert these using declarations, we can write cout instead of std::cout, cin instead of std::cin, etc…, in the remainder of the program
- Many programmers prefer to use the declaration

  ```
  using namespace std;
  ```

  which enables a program to use all the names in any standard C++ header file (such as <iostream>) that a program might include

The New C++ Headers

- The new-style headers do not specify filenames.
- They simply specify standard identifiers that might be mapped to files by the compiler, but they need not be.
  - <iostream>
  - <vector>
  - <string>, not related with <string.h>
  - <cmath>, C++ version of <math.h>
  - <cstring>, C++ version of <string.h>
- Programmer defined header files should end in “.h”.
  - #include "myheader.h"

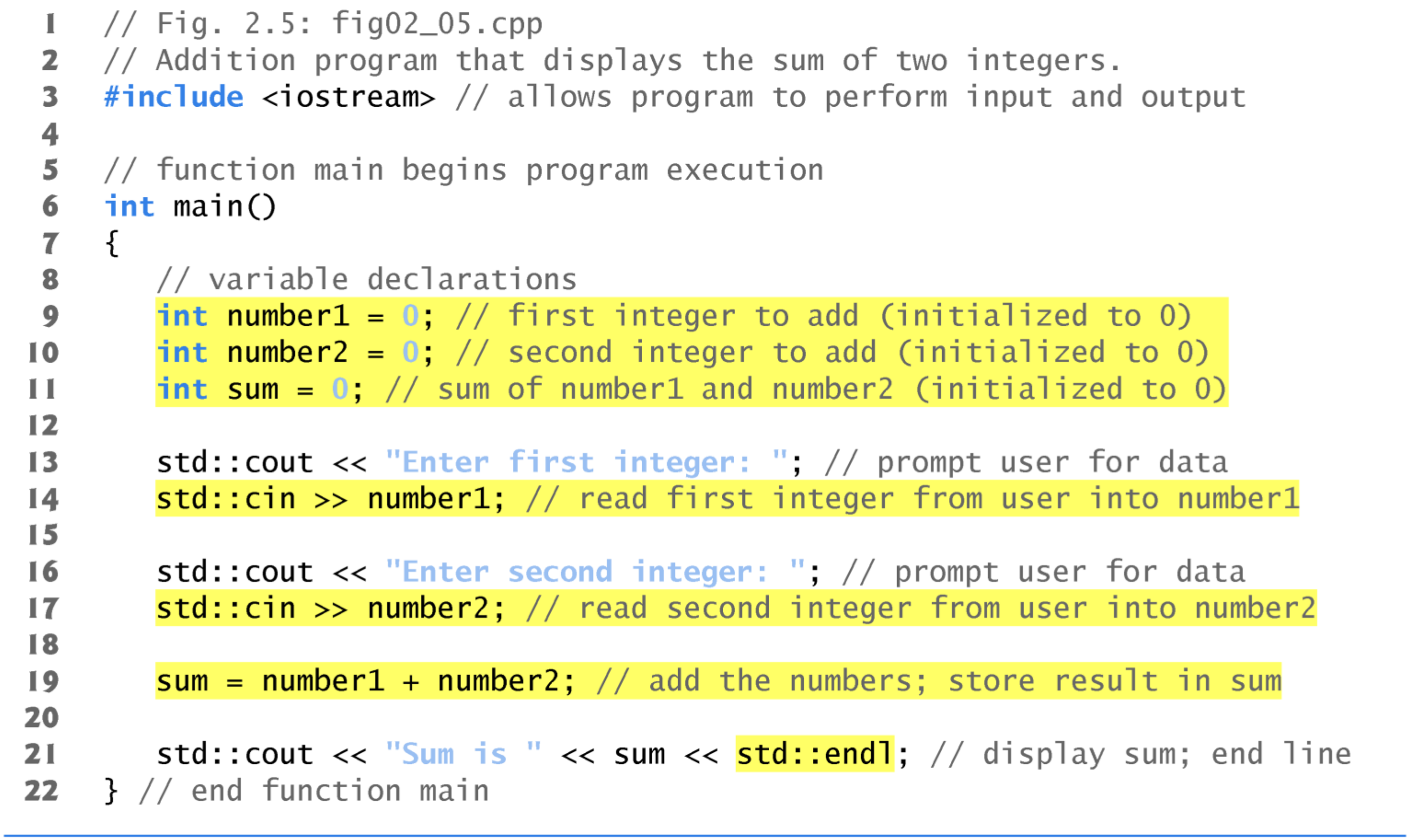
std::cout

- Typically, output and input in C++ are accomplished with streams of characters.
- When a cout statement executes, it sends a stream of characters to the standard output stream object—std::cout—which is normally “connected” to the screen.
- The std:: before cout is required when we use names that we’ve brought into the program by the preprocessing directive #include <iostream>
  - The notation std::cout specifies that we are using a name, in this case cout, that belongs to “namespace” std
  - The names cin (the standard input stream) and cerr (the standard error stream) also belong to namespace std

Scope Resolution Operator (::)

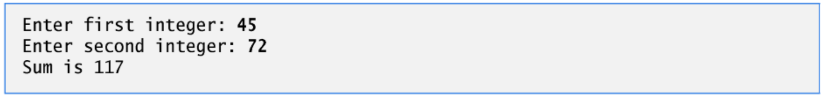
- Unary Scope Resolution Operator
  - Used to access a hidden global variable
- Binary Scope Resolution Operator
  - Used to associate a member function with its class (will be discussed shortly)
  - Used to access a hidden class member variable (will be discussed shortly)

Example: Adding Integers (1)



```cpp
// Fig. 2.5: fig02_05.cpp
// Addition program that displays the sum of two integers.
#include <iostream> // allows program to perform input and output

// function main begins program execution
int main()
{
   // variable declarations
   int number1 = 0; // first integer to add (initialized to 0)
   int number2 = 0; // second integer to add (initialized to 0)
   int sum = 0; // sum of number1 and number2 (initialized to 0)

   std::cout << "Enter first integer: "; // prompt user for data
   std::cin >> number1; // read first integer from user into number1

   std::cout << "Enter second integer: "; // prompt user for data
   std::cin >> number2; // read second integer from user into number2

   sum = number1 + number2; // add the numbers; store result in sum

   std::cout << "Sum is " << sum << std::endl; // display sum; end line
} // end function main
```

Fig. 2.5 | Addition program that displays the sum of two integers.



```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Fig. 2.5 | Addition program that displays the sum of two integers.

Example: Adding Integers (2)

- Declarations introduce identifiers into programs.
- The identifiers number1, number2 and sum are the names of variables
- A variable is a location in the computer’s memory where a value can be stored for use by a program
- Variables number1, number2 and sum are data of type int, meaning that these variables will hold integer values, i.e., whole numbers such as 7, –11, 0 and 31914
- All variables *must* be declared with a *name* and a *data type* before they can be used in a program
- If more than one name is declared in a declaration (as shown here), the names are separated by commas (,); this is referred to as a comma-separated list

Example: Adding Integers (3)

- Data type double is for specifying real numbers, and data type char for specifying *character data*
- Real numbers are numbers with decimal points, such as 3.4, 0.0 and –11.19
- A char variable may hold only a single lowercase letter, a single uppercase letter, a single digit or a single special character (e.g., $ or *)
- Types such as int, double and char are called fundamental types
- Fundamental-type names are keywords and therefore *must* appear in all lowercase letters

Memory Concepts

- Variable names such as number1, number2 and sum actually correspond to locations in the computer's memory
- Every variable has a name, a type, a size and a value
- When a value is placed in a memory location, the value overwrites the previous value in that location; thus, placing a new value into a memory location is said to be destructive
- When a value is read out of a memory location, the process is nondestructive

Pointers (1)

- Some C++ tasks are performed more easily with pointers, and other C++ tasks, such as dynamic memory allocation, cannot be performed without them
- Every variable is a memory location and every memory location has its address defined which can be accessed using and (&) operator which denotes an address in memory

Consider the following which will print the address of the variables defined −

```
#include <iostream>
using namespace std;
int main () {
int var1 = 5;
cout << "Address of var variable: ";
cout << &var1 << endl;
cout << "Value of var variable: ";
cout << var1 << endl;
return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Address of var variable: 0xbfebd5c0
Value of var variable:5
```

Pointers (2)

- A **pointer** is a variable whose value is the address of another variable
- Like any variable or constant, you must declare a pointer before you can work with it
  - The general form of a pointer variable declaration is type *var-name;
  - Here, **type** is the pointer's base type; it must be a valid C++ type and **var-name** is the name of the pointer variable.
- The asterisk is being used to designate a variable as a pointer
  - Example: int *ip; // pointer to an integer
- The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address
- The only difference between pointers of different data types is the data type of the variable that the pointer points to

Pointers (3)

- There are few important operations, which we will do with the pointers very frequently
  a) We define a pointer variable
  b) Assign the address of a variable to a pointer.
  c) Finally access the value at the address available in the pointer variable.
- This is done by using unary operator * that returns the value of the variable located at the address specified by its operand

Pointers: Example

```
#include <iostream>
using namespace std;
int main () {
        int var = 20; // actual variable declaration.
        int *ip; // pointer variable
        ip = &var; // store address of var in pointer variable
        cout << "Value of var variable: ";
        cout << var << endl;
        // print the address stored in ip pointer variable
        cout << "Address stored in ip variable: ";
        cout << ip << endl;
        // access the value at the address in pointer
        cout << "Value of *ip variable: ";
        cout << *ip << endl;
        return 0;
}
```

When the above code is compiled and executed, it produces result something as follows −

```
Value of var variable: 20
Address stored in ip variable: 0xbfc601ac
Value of *ip variable: 20
```

from https://www.tutorialspoint.com/cplusplus/cpp_pointers.htm

References

- ❑ A reference variable is an alias, that is, another name for an already existing variable
- ❑ Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable

References vs Pointers

References are often confused with pointers but three major differences between references and pointers are:

1. You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage
2. Once a reference is initialized to an object, it cannot be changed to refer to another object. Pointers can be pointed to another object at any time
3. A reference must be initialized when it is created. Pointers can be initialized at any time

Creating References in C++

- Think of a variable name as a label attached to the variable's location in memory. You can then think of a reference as a second label attached to that memory location
- You can access the contents of the variable through either the original variable name or the reference
- References are usually used for function argument lists and function return values

```
int i = 17;
//We can declare reference variables for i as follows.
int& r = i;
// Read the & in this declaration as a reference. Thus, read the declaration as "r is an integer reference initialized to i"
```

Example: References

```
#include <iostream>
using namespace std;
int main () {
    // declare simple variables
    int i;
    double d;
    // declare reference variables
    int& r = i;
    double& s = d;
    i = 5;
    cout << "Value of i : " << i << endl;
    cout << "Value of i reference : " << r << endl;
    d = 11.7;
    cout << "Value of d : " << d << endl;
    cout << "Value of d reference : " << s << endl;
    return 0;
}
```

When the above code is compiled together and executed, it produces the following result

```
Value of i : 5
Value of i reference : 5
Value of d : 11.7
Value of d reference : 11.7
```

Structures (1)

- C/C++ arrays allow you to define variables that combine several data items of the same kind
- **structure** is another user defined data type which allows you to combine data items of different kinds
  - Structures are used to represent a record
- To define a structure, you must use the struct statement.
  - The struct statement defines a new data type, with more than one member, for your program

Structures (2)

- The format of the struct statement is :

```
struct [structure tag] {
member definition;
member definition;
...
member definition;
} [one or more structure variables];
```

- The **structure tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition
- At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional

```
struct product {
  int weight;
  double price;
} ;

product apple;
product banana, melon
```

or

```
struct product {
  int weight;
  double price;
} apple, banana, melon
```

Structures (3)

- suppose you want to keep track of your books in a library and you want to track the following attributes about each book:
    - Title
    - Author
    - Subject
    - Book ID

apple.weight
apple.price
banana.weight
banana.price
melon.weight
melon.price

- Here is the way you would declare the Book structure −
    - struct Books {
    - char title[50];
    - char author[50];
    - char subject[100];
    - int book_id;
    - } book;

Structures (4)

- To access any member of a structure, we use the **member access operator (.)**
  - It is coded as a period between the structure variable name and the structure member that we wish to access
- You can pass a structure as a function argument in very similar way as you pass any other variable or pointer
- You can define pointers to structures in very similar way as you define pointer to any othervariable as follows −
  - struct Books *struct_pointer;
- To find the address of a structure variable, place the & operator before the structure's name as follows
  - struct_pointer = &Book1;
- To access the members of a structure using a pointer to that structure, you must use the -> operator as follows
  - struct_pointer->title;

Classes: A First Look (1)

- General syntax

```
class class-name
{
        // private functions and variables
public:

        // public functions and variables
}object-list (optional);
```

Classes: A First Look (2)

- A class declaration is a logical abstraction that defines a new type
- It determines what an object of that type will look like
- An object declaration creates a physical entity of that type
- That is, an object occupies memory space, but a type definition does not
- Each object of a class has its own copy of every variable declared within the class, but they all share the same copy of member functions
  - How do member functions know on which object they have to work on?
    - The answer will be clear when "***this***" pointer is introduced

Introducing Function Overloading

- Provides the mechanism by which C++ achieves one type of polymorphism (called **compile-time polymorphism**).
- Two or more functions can share the same name as long as either
  - The type of their arguments differs, or
  - The number of their arguments differs, or
  - Both of the above
- The compiler will automatically select the correct version
- The return type alone is not a sufficient difference to allow function overloading

Constructors

- Every object we create will require some sort of initialization
- A class's constructor is automatically called by the compiler each time an object of that class is created
- A constructor function has the ***same name*** as the class and has ***no return type***
- There is no explicit way to call the constructor

Destructors

- The complement of a constructor is the destructor
- This function is automatically called by the compiler when an object is destroyed
- The name of a destructor is the ***name of its class***, preceded by a ~
- There is explicit way to call the destructor but highly discouraged

Constructors & Destructors

- For global objects, an object’s constructor is called once, when the program first begins execution
- For local objects, the constructor is called each time the declaration statement is executed
- Local objects are destroyed when they go out of scope
- Global objects are destroyed when the program ends

Constructors That Take Parameters

- It is possible to ***pass arguments*** to a constructor function
- Destructor functions ***cannot*** have parameters
- A constructor function with no parameter is called the ***default constructor*** and is supplied by the compiler automatically if no constructor defined by the programmer
- The compiler supplied default constructor ***does not initialize*** the member variables to any default value; so they contain garbage value after creation
- Constructors ***can be overloaded***, but destructors ***cannot be overloaded***
- A class can have multiple constructors

Object Pointers

- It is possible to access a member of an object via a pointer to that object
- When a pointer is used, the arrow operator (->) rather than the dot operator is employed
- Just like pointers to other types, an object pointer, when incremented, will point to the next object of its type