C++ Data Types

- Built-in data types
- Literal constants
- Variables
- Pointers
- References
- The C++ string Type
- The const Qualifier

Numeric Data Types

- Type char represents individual characters and small integers (1 byte).
- Types short, int, and long represent integer values (half a machine word, 1 machine word, 1 or 2 machine words, resp.)
- Types float, double, and long double represent floating point values (1 machine word, 2 machine words, 3 or 4 machine words, resp.)

Literal Constants

- Type char, short, int, and long are also called integral types.
- Integral types can be signed or unsigned.
- Example: The value of an 8-bit unsigned char ranges from 0 to 255, while the range for an 8-bit signed char is from –128 to 127.

By default, the C++ compiler assumes that all literal integer constants are of type int and all literal floating point constants are of type double.

We can use prefixes to write literal integer constants in decimal, octal, or hexadecimal notation.

Examples:
Decimal (no prefix): 15
Octal (prefix 0 [zero]): 015 = 13 (decimal)
Hexadecimal (prefix 0x [zero-x]): 0x15 = 21 (decimal)

Literal Constants

- Literal constants are values that occur in a program.
- Example:
  int main()
  {
    int students = 21;
    double pi = 3.1416;
  }

  Here, 21 is a literal constant of type int, and 3.1416 is a literal constant of type double.

By default, the C++ compiler assumes that all literal integer constants are of type int and all literal floating point constants are of type double.

Examples:
2344U (unsigned)
1555L (long)
166UL (unsigned long)
3.1416F (float)
6.2831L (long double)
Literal Constants

Another built-in (or primitive) C++ data type is the type bool. Its only literals are true and false. Note that the type bool does not exist in C. In C, we represent the Boolean values true and false by the integers 1 and 0.

Literal Constants

We use single quotation marks to write literal character constants. Examples:
'x', '4', ':', ' ' (space)

Nonprintable characters and some special characters can be represented by escape sequences. Examples:
'\n' (newline), 'a' (bell), 't' (tab)
'\\' (backslash), '"' (double quote)

Literal Constants

Generalized escape sequences are indicated by a backslash followed by up to three octal digits. The value of the octal digits in such a sequence is interpreted as the corresponding literal constant in the ASCII character set.
Examples:
\7 (bell)
\14 (newline)
\65 ('5')

Literal Constants

A character literal can be preceded by an L, for example:
L'a'
This is called a wide-character literal and has type wchar_t. Such wide-character literals support language character sets like Chinese and Japanese, which cannot be represented within the 256 character ASCII set.

Literal Constants

A literal string constant is composed of zero or more characters enclosed in double quotation marks. Examples:
"" (null string)
"x"
"hello"
"Hi,\nHow are you?\n"

Literal Constants

A string literal can be written across multiple lines. You can use a backslash as the last character on a line to indicate that the string continues on the next line. Example:
"This is an \ excellent \ multi-line string literal."
Variables

Variables provide us with named memory storage that we can
- write to,
- read from, and
- manipulate throughout the course of our program.

Each variable has a specific type, which determines
- the size and layout of its associated memory,
- the range of values that can be stored, and
- the set of operations that can be applied to it.

Variables are also referred to as objects.

Variables

There are two values associated with a variable:

1. Its data value, which is stored at some memory address. It is also called the rvalue (read value) of the variable.
2. Its address value, indicating the location in memory where its data value is stored. This value is also referred to as the variable’s lvalue (location value).

While literal constants also have a rvalue, they do not possess an lvalue.

Variables

Names (identifiers) of variables can be made up of letters, digits, and underscores.
- Names cannot start with a digit.
- Notice that C++ compilers distinguish between lower- and uppercase letters.
- C++ keywords such as if, else, class, etc. cannot be used as variable names.

Naming conventions:
- Object-oriented style: Use lowercase letters for names of objects; capitalize the first letter of each embedded word in a multiword identifier.
  Examples: year, robotCameraModule
- “Microsoft style”: Also provide type information in object names.
  Examples: iYear, strFilename

General advice: Use mnemonic names, that is, names describing the purpose of the object.

Variables

Initialization of variables:
- If a variable is defined at global scope, it automatically receives an initial value of zero.
- Variables at local scope and dynamically allocated variables receive an undefined value.
- It is generally recommended that you assign an initial value to all variables.

Example: int loopCounter = 0;
- Class objects are automatically initialized through their default constructor.

Further examples of initialization:
- int year = 2002;
- string myName = "Peter";
- int year(2002);
- string myName("Peter");
- int loopCounter = int(); // sets loopCounter to 0
- double myWeight = double(); // sets myWeight to 0.0
- int value = 3*3;
- int value = GetValue();
Pointers

- A pointer holds the memory address of another object.
- Through the pointer we can indirectly manipulate the referenced object.

Pointers are useful for
- Creating linked data structures such as trees and lists,
- management of dynamically allocated objects, and
- as a function parameter type for passing large objects such as arrays.

Pointers

- Every pointer has an associated type.
- The type of a pointer tells the compiler how to interpret the memory content at the referenced location and how many bytes this interpretation includes.

Examples of pointer definitions:
- int *pointer;
- int *pointer1, *pointer2;
- string *myString;

Pointers

- The dereference operator (*) dereferences a pointer variable so that we can manipulate the memory content at the location specified by the pointer.
- The address-of operator (&) provides the memory address (a pointer) of a given object.

Example: Correct or incorrect?
- int var1 = 333, var2 = 444, *pvar1, *pvar2; correct.
- pvar1 = var1; incorrect.  *int ≠ int
- pvar2 = &var2; correct.  *int = *int
- *pvar1 = var2; correct.  int = int
- *pvar2 = *pvar1 + 100; correct.  int = int

Question: Considering only correct lines, what is the final value of var2?
Answer: var2 = 544

Pointers

Notice that in pointer definitions the '*' symbol indicates the pointer type and is not the dereference operator.

Example:
- int var;
- int *pvar1 = var;
- Incorrect! During initialization a pointer can only be assigned an address:
- int var;
- int *pvar1 = &var;
- Correct!

Pointers

You can use pointer arithmetic to iterate through an array:

- int ia[10];
- int *iter = &ia[0];
- int *iter_end = &ia[10];

while (iter != iter_end)
{
    do_something_with_value(*iter);
    ++iter;
}
References

- References (aliases) can be used as alternative names for objects.
- In most cases they are used as formal parameters to a function.
- A reference type is defined by following the type specifier with the address-of operator.

Example:
```cpp
int val1 = 333;
int &refVal1 = val1;
```

- A reference must be initialized.
- Once defined, a reference cannot be made to refer to another object.
- All operations on the reference are actually applied to the object to which the reference refers.

Example:
```cpp
int val1 = 333;
int &refVal1 = val1;
val1++;
refVal1 += 100;
cout << "Result: " << refVal1;
Result: 434
```

The C++ string Type

To use the C++ string type, you must include its associated header file:
```cpp
#include <string>
```

Different ways to initialize strings:
```cpp
string myString("Hello folks!");
string myOtherString(myString);
string myFinalString; // empty string
```

The length of a string is returned by its `size()` operation (without the terminating null character):
```cpp
cout << myString.size();
12
```

We can use the `empty()` operation to find out whether a string is empty:
```cpp
bool isStringEmpty = myString.empty();
```

Use the equality operator to check whether two strings are equal:
```cpp
if (myString == myOtherString)
cout << "Wow, the strings are equal."
```

Copy one string to another with the assignment operator:
```cpp
myFinalString = myOtherString;
```

Use the `plus` operator to concatenate strings:
```cpp
string s1 = "Wow! ", s2 = "Ouch! ";
const char *s3 = "Yuck! "
s2 += s1 + s3 + s2;
cout << s2;
Ouch! Wow! Yuck! Ouch!
```

The const Qualifier

The const type qualifier transforms an object into a constant.

Example:
```cpp
const double pi = 3.1416;
```

- Constants allow you to store parameters in well-defined places in your code.
- Constants have an associated type.
- Constants must be initialized.
- Constants cannot be modified after their definition.
- Constants replace the `#define` technique in C.