Types & Variables

Prof. Dr.-Ing. Angela Brennecke
Audio & Interactive Media Technologies
Film University Babelsberg KONRAD WOLF

Data Types

Data Types

- One of the fundamental strengths of C++ is the way the language deals with data and the associated data types

- On a design level, data types can be considered as representations of a concept with a certain kind of feature set

- On a hardware level, the data type determines the piece of memory that is allocated & reserved during compilation

Statically Typed

- C++ is a statically typed language

- Data types have to be defined (or deduced) at compile time
- Data types have to be associated with variables
- The type of the variable cannot change

Statically Typed

- Type checking is executed at compile time

- To identify type errors early in the development cycle
- To allocate the required memory at compile time
- To ensure faster program execution

Statically vs Dynamically Typed

- In contrast, dynamically typed languages do not associate data types with variables — data types are dynamically defined and checked at run time and variable type associations can change

- This is another reason for why interpreted languages are usually slower in execution than statically typed languages
  Examples: python, javascript

Supported Types

- C++ supports built-in data types (float, int, bool, …) and user-defined data types (enum, struct, and classes)

Supported Types

C++ has a set of fundamental types corresponding to the most common basic storage units of a computer and the most common ways of using them to hold data:

§4.2 A Boolean type (***bool***)
§4.3 Character types (such as *char*)
§4.4 Integer types (such as *int*)
§4.5 Floating-point types (such as ***double***)

In addition, a user can define

§4.8 Enumeration types for representing specific sets of values (***enum***)

There also is

§4.7 A type, ***void***, used to signify the absence of information

From these types, we can construct other types:

§5.1 Pointer types (such as ***int****)
§5.2 Array types (such as ***char***[ ])
§5.5 Reference types (such as ***double***&)
§5.7 Data structures and classes (Chapter 10)

… represent mathematical concepts like numbers, logical operations and language / text concepts.

… represent user-defined sets & the concept of nothingness, i.e., mathematical concept of zero.

… represent concepts of memory allocation, memory access & aliasing as well as fully user-defined concepts.

Source credit: Bjarne Stroustrup (1997): **The C++ Programming Language.** Upper Saddle River, NJ: Pearson Education, Inc.

Built-in Types

The table shows how many bytes in memory will be allocated when using a specific data type

Category	Type	Minimum Size	Note
boolean	bool	1 byte	
character	char	1 byte	May be signed or unsigned Always exactly 1 byte
	wchar_t	1 byte	
	char16_t	2 bytes	C++11 type
	char32_t	4 bytes	C++11 type
integer	short	2 bytes	
	int	2 bytes	
	long	4 bytes	
	long long	8 bytes	C99/C++11 type
floating point	float	4 bytes	
	double	8 bytes	
	long double	8 bytes	

Source credit: http://www.learncpp.com/cpp-tutorial/23-variable-sizes-and-the-sizeof-operator/

User-Defined Types

- The possibility to specify user-defined types allows
  - to create completely new data types
  - to specifically match the application and/or user needs
  - to translate complex concepts into software code
  - to improve the readability of complex systems

User-Defined Types

```
// Prof. Dr. Angela Brennecke | Creative Coding II | Filmuniversitaet Babelsberg | 2018
// Based on: Ulrich Breymann (2017): Der C++ Programmierer. Carl Hanser Verlag München.
#include <iostream>

enum Color
{
    red,
    yellow,
    green
};

struct Point
{
    int x;
    int y;
    bool isVisible;
    Color aColor;
};

```

- The enumeration type is a user-defined data type that allows to group a list of symbolic constant of type integer
- A variable of type Color functions like a constant integer variable

User-Defined Types

```
// Prof. Dr. Angela Brennecke | Creative Coding II | Filmuniversitaet Babelsberg | 2018
// Based on: Ulrich Breymann (2017): Der C++ Programmierer. Carl Hanser Verlag München.
#include <iostream>

enum Color
{
    red,
    yellow,
    green
};

struct Point
{
    int x;
    int y;
    bool isVisible;
    Color aColor;
};
```

- The structure type allows to combine different kinds of data types inside of one struct
- The keyword struct is required to define the data type

User-Defined Types

- Typical class interface of an openFrameworks application class

- The class prototype is usually defined in the header file *.h …

```
#pragma once
#include "ofMain.h"

// ofApp derives from ofBaseApp
class ofApp : public ofBaseApp
{
    // private section - everything here is private and only
    // accessible from within the class ofApp

public:
    // everything after keyword public can be accessed from
    // outside of the class, i.e., the public area represents
    // the interface of the class
    void setup();
    void update();
    void draw();

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y );
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);

private:
    // everything after keyword private can not be accessed from
    // outside of the class but only from this class alone
    // the private area represents the "hidden" implementation
    // details of the class
    std::string appFirstWords;
    bool startDrawing;
};
```

User-Defined Types

- … whereas the actual implementation is defined in the definition file *.cpp

```cpp
#include "ofApp.h"

//--------------------------------------------------------------
void ofApp::setup()
{
    ofBackground(0);
    ofSetBackgroundAuto(false);
}

//--------------------------------------------------------------
void ofApp::update()
{

}

//--------------------------------------------------------------
void ofApp::draw()
{

}

//--------------------------------------------------------------
void ofApp::keyPressed(int key)
{

}

//--------------------------------------------------------------
void ofApp::keyReleased(int key){

```

Variables

Variables

- Variables in C++ are **objects** that have a name, can be accessed by their name, and can be changed — unless they are constants
- On a hardware level, an object is a piece of memory that is allocated & reserved by the computer based on the **data type**
- The data type helps to interpret the allocated memory

Prof. Dr.-Ing. Angela Brennecke
Audio & Interactive Media Technologies
Film University Babelsberg KONRAD WOLF

Type Association

- Variables are associated with a certain data type in the code

```
float grade {5.0};          // uniform initialization
int years = 4 + 3;          // copy initialization (slower)
bool test;                  // declaration (better do initialization)

auto grade {5.0};
auto years = 4 + 3;
auto test;                  // error

myClass myObjectVar {};     // user defined type calling default constructor
```

- The newly introduced type auto allows for automatic type deduction — this only works for initializing variables upon creation (see https://www.learncpp.com/cpp-tutorial/4-8-the-auto-keyword/)

Initialization

- Initialization describes the process of immediately specifying the value of the variable once it has been defined

```
int numYears {10};       // uniform initialization of a variable since C++11
int numMonths {};        // uniform initialization to zero

int numDays = 55;        // copy initialization (slower)

int numMinutes(45);      // direct initialization (old version)
```

- Uninitialized variables hold some kind of random value
- Always initialize variables to avoid undefined behavior

Constants

- Variables are used to store a certain value of a certain data type

- Sometimes, these values shall change
- Sometimes, these values shall not change — rather the variable is used as a constant or even symbolic constant

- C++ provides to options to ensure that variables can only be initialized but not changed — const & constexpr

Prof. Dr.-Ing. Angela Brennecke
Audio & Interactive Media Technologies
Film University Babelsberg KONRAD WOLF

const Keyword

- The const keyword is most of used for function parameters to ensure that the function does not change the argument

```
void printInteger(const int myValue)
{
    std::cout << myValue;
}
```

Image credit: http://www.learncpp.com/cpp-tutorial/2-9-symbolic-constants-and-the-const-keyword/

- The const keyword is also used for variables that are being initialized during run-time — that are not know at compile-time

```
std::cout << "Enter your age: ";
int age;
std::cin >> age;

const int usersAge (age); // usersAge can not be changed
```

Image credit: http://www.learncpp.com/cpp-tutorial/2-9-symbolic-constants-and-the-const-keyword/

constexpr Keyword

```cpp
constexpr double gravity (9.8); // ok, the value of 9.8 can be resolved at compile-time
constexpr int sum = 4 + 5; // ok, the value of 4 + 5 can be resolved at compile-time

std::cout << "Enter your age: ";
int age;
std::cin >> age;
constexpr int myAge = age; // not okay, age can not be resolved at compile-time
```

Image credit: http://www.learncpp.com/cpp-tutorial/2-9-symbolic-constants-and-the-const-keyword/

- The constexpr keyword is used for variables that are known at compile-time & can be directly initialized with a value then
- constexpr variables can well be used for symbolic constants