02393 Programming in C++
Module 3: Data Types, Pointers

Sebastian Mödersheim

September 12, 2016

Lecture Plan

#	Date	Topic
1	29.8.	Introduction
2	5.9.	Basic C++
3	12.9.	Data Types, Pointers
4	19.9.	Libraries and Interfaces; Containers
5	26.9.	
6	3.10.	Classes and Objects I
7	10.10.	Classes and Objects II
		Efterårsferie
8	24.10.	Classes and Objects III
9	31.10.	Recursive Programming
10	7.11.	Lists
11	14.11.	Trees
12	21.11.	Novel C++ features
13	28.11.	Summary
	5.12.	Exam

Outline

1. **Recap**

   Basic data types

2. **Data types**

3. **Pointers**

Small Programming Exercises via CodeJudge

Some general observations:

- Use the test examples to check your input/output
- If it seems to refuse correct solutions...
  - ★ check thoroughly the input and output of the tests
  - ★ beware of imprecisions in the input/output format (e.g. blank spaces)
  - ★ write clean code, e.g. initialize variables before using them
  - ★ please use the Piazza platform to ask questions (or contact the TAs)
  - ★ in case of technical problems with CodeJudge you may directly contact Anders Roy anders@codejudge.net

Last week's exercises (to hand in today):

- Questions?
- Model solutions online (and discussed here)

Recap: Last Programming Session

- Bounded numerical types, e.g. unsigned int $= [0, .., \mathrm{UINT\_MAX}] \subset \mathbb{Z}$;
- Arithmetic imprecision, e.g. sometimes $a * b/c$ may not give the same result as $a/c * b$
- Bottom line: Be aware of the limits of the used data types!
- Stack limits, e.g. recursion may crash;
- C++ functions are programs, not mathematical functions;

Outline

1. Recap
2. **Data types**
3. Pointers

The hierarchy of data types

Atomic/Fundamental types

- booleans **bool**
- characters **char**
- integer numbers: [**unsigned**] [**long**] **int**
- floating point numbers: [**unsigned**] **float**, **double**, **long double**
- define your own: **enum**

See http://en.cppreference.com/w/cpp/language/types

The hierarchy of data types

Atomic/Fundamental types

- booleans **bool**
- characters **char**
- integer numbers: [**unsigned**] [**long**] **int**
- floating point numbers: [**unsigned**] **float**, **double**, **long double**
- define your own: **enum**

See http://en.cppreference.com/w/cpp/language/types

New types composed from the existing type

1. **struct** (aka record): a collection of data values
2. **array**: sequence of data values of the same type
3. **pointer**: stores a memory address

Mixed data types, casting

What's the type of

- $9/6$?
- $9.0/6$?
- $9/6.0$?
- $9/\mathit{int}(6.0)$, and $\mathit{float}(9/6)$?

Mixed data types, casting

What's the type of

- 9/6?
- 9.0/6?
- 9/6.0?
- 9/*int*(6.0), and *float*(9/6)?

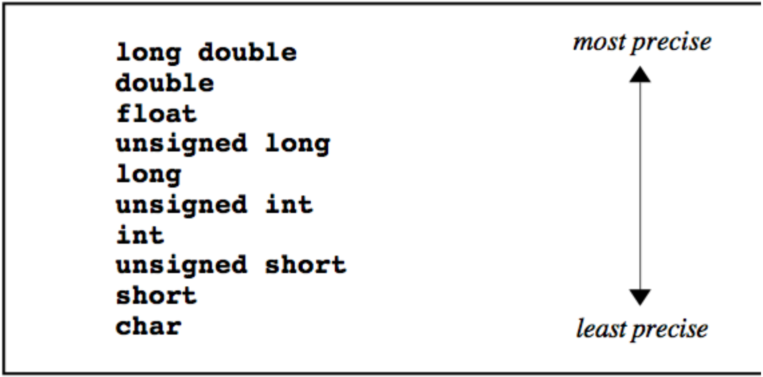
- for an operator whose operands are of different types, the compiler converts the operands to a common type (when possible)
- the type that is more precise will be chosen
- result is always that of the arguments after any conversions are applied

Mixed data types, casting

What's the type of

- $9/6$?
- $9.0/6$?
- $9/6.0$?
- $9/\mathit{int}(6.0)$, and $\mathit{float}(9/6)$?

Table 1-5 Type conversion hierarchy for numeric types



```
long double
double
float
unsigned long
long
unsigned int
int
unsigned short
short
char
```

most precise

least precise

Enum, structs, and arrays in a maze

```
typedef enum {wood,stone} material;

typedef struct {
    int x,y;
    bool isWall;
    material type;
} field;

int main(){
    ...
    field playground[n][m];
    for (int i=0; i<n; i++){
        for (int j=0; j<m; j++){
            playground[i][j].x=i;
            playground[i][j].y=j;
            playground[i][j].isWall=(i==0||i==(n-1)||j==0||j==(m-1));
            if (playground[i][j].isWall)
                playground[i][j].type=stone;
            else
                playground[i][j].type=wood;
        }
    }
  ...
```

Remarks on these Data Types

- **enum** is just a bit of “syntactic sugar”

Remarks on these Data Types

- **enum** is just a bit of “syntactic sugar”
- **struct** is one of the core concepts: defining new types of data as record of existing types.
  - ★ Every entry in the record has a name and type.
  - ★ This is the basis for object-oriented programming later: (“just add functions”)

Remarks on these Data Types

- **enum** is just a bit of "syntactic sugar"
- **struct** is one of the core concepts: defining new types of data as record of existing types.
  - ★ Every entry in the record has a name and type.
  - ★ This is the basis for object-oriented programming later: ("just add functions")
- Arrays are also a universal concept. Note however in C++:
  - ★ Arrays range from [0] to [$n-1$] when the size is n
  - ★ The size of the array is not stored with the array! (It is your responsibility to keep track if it.)
  - ★ If you access outside the boundaries of the array, the compiler will not stop you; this may produce hard-to-find errors!
  - ★ The size of an array cannot be changed.
  - ★ Passing arrays as function arguments can be tricky (more later).

Remarks on these Data Types

- **enum** is just a bit of “syntactic sugar”
- **struct** is one of the core concepts: defining new types of data as record of existing types.
  - ★ Every entry in the record has a name and type.
  - ★ This is the basis for object-oriented programming later: (“just add functions”)
- Arrays are also a universal concept. Note however in C++:
  - ★ Arrays range from [0] to [$n-1$] when the size is n
  - ★ The size of the array is not stored with the array! (It is your responsibility to keep track if it.)
  - ★ If you access outside the boundaries of the array, the compiler will not stop you; this may produce hard-to-find errors!
  - ★ The size of an array cannot be changed.
  - ★ Passing arrays as function arguments can be tricky (more later).
- Next week: C++ offers a data-structure **vector** in the library that overcomes many of the problems with arrays.
  - ★ Usually a vector is preferable over an array!

Outline

1. Recap
2. Data types
3. Pointers

Pointers

- A pointer is a variable which contains a memory address
- Access to, and manipulation of, pointers by a program allows some interesting applications:
  - ★ Great way to screw up your code! Use with care!
  - ★ Classic way (pre 90's) to implement call-by-reference
    - ▶ We discuss an example; for most applications use modern C++ call-by-reference.
  - ★ Dynamic memory allocation:
    - ▶ the program asks the system for more memory with **new**.
    - ▶ the system answers with a pointer to the memory block
    - ▶ must be explicitly given back with **delete**—there is no garbage collection.
  - ★ Based on dynamic memory: recursive data structures (in 3rd part of course).

Pointers

Definition

A memory address (for example of a variable) is a pointer value, which can be stored in memory like “normal” data.

Pointers

Definition

A memory address (for example of a variable) is a pointer value, which can be stored in memory like "normal" data.

Declaring pointer variables:

```
int *p1, *p2, p3;
char *cptr;
```

Pointers

Definition

A memory address (for example of a variable) is a pointer value, which can be stored in memory like “normal” data.

Declaring pointer variables:

```
int *p1, *p2, p3;
char *cptr;
```

Pointer operations

- &: address-of. Takes a variable and returns the corresponding memory address
- ∗: value-pointed-to, returns the variable, or the pointee , the pointer points to.

Live Programming