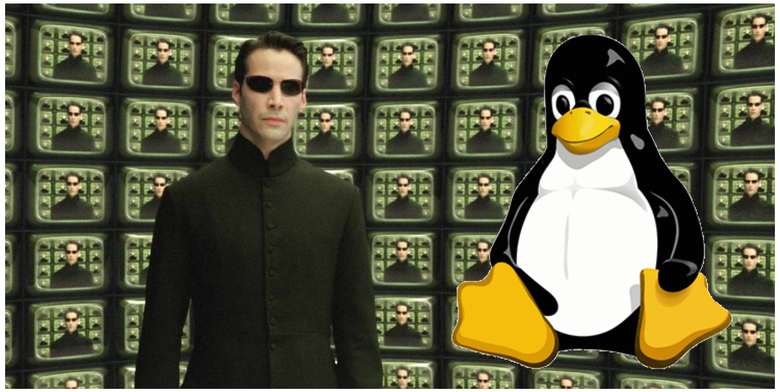
09.07.20 19:32

CSCI 125
Introduction to Computer Science and Programming II
Lecture 6: User Defined Datatypes



Jetic Gū

2020 Summer Semester (S2)

P0
Midterm

Bad

- I understand online studying can be bothering
- This is NOT an excuse to COPY others' work
- It's also a bad idea to ignore practice problems

Overview

- Focus: Basic C/C++ Syntax
- Architecture: Linux/Unix OS
- Core Ideas:
    1. Enumeration and Typedef
    2. namespace

P0
Review

Data Types

- Primary √
  - Integers, Characters, Boolean √
  - Floating point √
  - Void√
- Derived
  - Function√, Array√, Pointer, Reference√
- User Defined
  - Struct, Class, **Enumerate**, **Typedef**

Review

Enumeration and Typedef

Important? No, I never used them, not even once

Summary

Enumeration

- Data type: a set of values together with a set of operations on those values
- To define a new simple data type, called enumeration type, we need three things:
  - A name for the data type
  - A set of values for the data type
  - A set of operations on the values

Concept

Enumeration

- A new simple data type can be defined by specifying its name and the values, but not the operations
  - The values must be identifiers (usually all UPPERCASE letters)
- Syntax:

```
enum typeName {value1, value2, ...};
```

  - `value1`, `value2`, … are identifiers called enumerators
  - `value1` < `value2` < `value3` <...

Concept

Enumeration

- Enumeration type is an ordered set of values
- If a value has been used in one enumeration type it **can’t be used** by another in **same scope**
- The same rules apply to enumeration types declared outside of any blocks

Concept

Enumeration

- Valid examples:
  - Colours

```
enum colours {RED, GREEN, BLUE, YELLOW, BURGUNDY};
```

  - Status

```
enum status {MARRIED, SINGLE, DIVORCED};
```

Concept

Enumeration

- Invalid examples:

```
enum colours {ORANGE, RED, YELLOW, BURGUNDY};
enum fruits {ORANGE, APPLE, STRAWBERRY};
```

  - Values here needs to be unique identifiers in that scope

Concept

Declaring Variables

- Syntax, declaration just like any other type

```
dataType identifier1, identifier2, ...;
```

- e.g.

```
enum colours {RED, GREEN, BLUE, YELLOW, BURGUNDY} c;
// declaring the type and variable c

colours border=RED, background=GREEN;
```

Concept

Relational Operators

```
enum colours {RED, GREEN, BLUE, YELLOW, BURGUNDY};
```

- An enumeration type is an ordered set of values:

```
RED <= GREEN is true

GREEN < BLUE is true

BURGUNDY > BLUE is true
```

Concept

Input /Output of Enumeration Types

- Enumeration type cannot be input/output directly
- Modifying `cin` and `cout` is quite complicated (not covered in CSCI125)

Concept

Anonymous Data Types

- Anonymous type
  values are directly specified in the declaration, with no type name

```
enum {RED, GREEN, BLUE, YELLOW, BURGUNDY} c;
// c is a variable here
```

- Drawbacks:
  - Cannot pass/return an anonymous type to/from a function
  - Values used in one type can be used in another, but are treated differently:

```
enum {EN, DE, FR, CN} language1;
enum {EN, DE, FR, CN} language2;
language1 = language2; // doesn't work
```

Concept

typedef Statement

- Creates a synonym / alias to a data type
- Syntax

```
typedef existingType newTypeName;
```

- `typedef` does not create any new data types
  - (Almost) the same as preprocessor directive

Concept

typedef Statement

- Creates a synonym / alias to a data type
- `typedef` does not create any new data types
  - (Almost) the same as preprocessor directive (but is not!)
- This will NOT compile

```
#include <iostream>
#include <string>
using namespace std;
typedef int notInt;
notInt main() {
    int a = 12;
    cout << double(a) << endl;
    return 0;
}
```

Concept

Namespace

Namespaces

- ANSI/ISO standard C++ was officially approved in July 1998
- Most of the recent compilers are also compatible with ANSI/ISO standard C++
- For the most part, standard C++ and ANSI/ISO standard C++ are the same
  - However, ANSI/ISO Standard C++ has some features not available in Standard C++

Namespaces

- Global identifiers in a header file used in a program become global in the program
  - Syntax error occurs if an identifier in a program has the same name as a global identifier in the header file
- Same problem can occur with third-party libraries
  - Common solution: third-party vendors begin their global identifiers with _ (underscore)
    - Do not begin identifiers in your program with _

Concept

Declaration

- ANSI/ISO Standard C++ attempts to solve this problem with the namespace mechanism
- Syntax

```
namespace namespace_name {
    members
}
```

- Member
  - Variable declarations; Constant declarations
  - functions; function prototypes
  - another namespace(s)

Declaration

```
namespace csci125 {
    int classSize;
    int classID;
    void printGrade();
}
```

- Then to use it

```
std::cout << csci125::classID << std::endl;
```

Scope

- The scope of a `namespace` member is local to the namespace
- Ways a `namespace` member can be accessed outside the `namespace`:

```
csci125::classSize

using namespace csci125;
// then you can access any member without csci125::

using csci125::classSize;
// then you can just use classSize
```

Concept