南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

CS205

Prof. Shiqi Yu (于仕琪)

yusq@sustech.edu.cn

http://faculty.sustech.edu.cn/yusq/

Friend Classes

`friend` Functions

- A friend function is defined out of the class.
- No `MyTime::` before its function name

```
class MyTime
{
  // ...
  public:
    friend MyTime operator+(int m, const MyTime & t);
};

MyTime operator+(int m, const MyTime & t)
{
    return t + m;
}
```

`friend` Classes

- A class is a friend of another class.
- The friend class can access all members even private members.
- A friend class can be public, protected and private.

```
class Sniper
{
private:
    int bullets;
public:
    Sniper(int bullets = 0);
    friend class Supplier;
};
```

```
class Supplier
{
    int storage;
public:
    Supplier(int storage = 1000);
    bool provide(Sniper & sniper)
    {
        // bullets is a private member
        if (sniper.bullets < 20)
        // ...
    }
};
```

friend.cpp

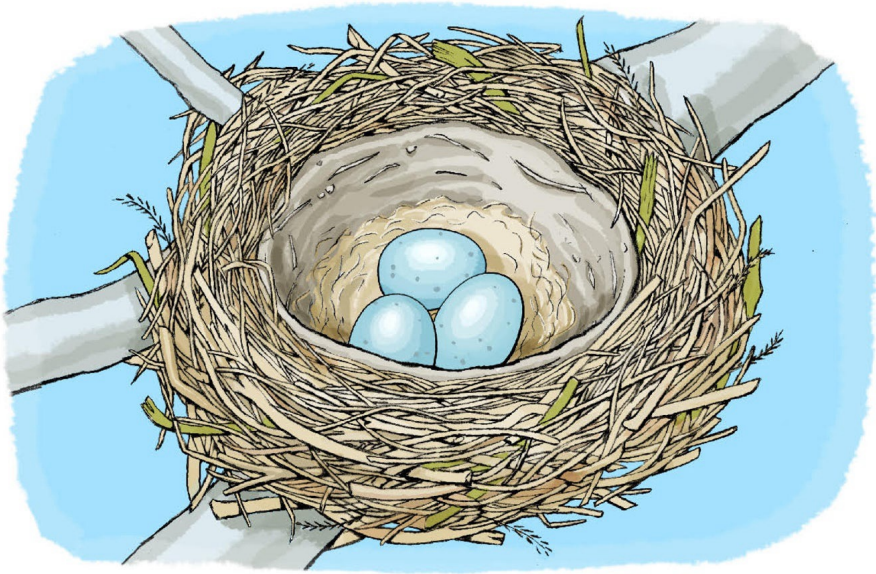
friend Member Functions

- A single member function of a class is a friend.
- Different from friend functions.
- But very similar to a normal friend function.
- But... declaration problem ...

```
class Sniper
{
private:
    int bullets;
public:
    Sniper(int bullets = 0): bullets(bullets){}
    friend bool Supplier::provide(Sniper &);
};
```

friend2.cpp

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY



Nested Types

https://www.scientificamerican.com/article/build-a-bird-nest/

Nested Enumerations (C++11)

- `enum DataType` is only used in class Mat, we can put it inside of Mat.

```
enum DataType
{
    TYPE8U,
    TYPE8S,
    TYPE32F,
    TYPE64F
};
class Mat
{
private:
    DataType type;
    void * data;
public:
    Mat(DataType type) : type(type), data(NULL){}
    DataType getType() const { return type; }
};
```

nested-enum.cpp

Nested Enumerations (C++11)

- It can be accessed outside of the class, but with the class name scope qualifier.

```
class Mat
{
public:
    enum DataType
    {
        TYPE8U,
        TYPE8S,
        TYPE32F,
        TYPE64F
    };
private:
    DataType type;
    void * data;
public:
    Mat(DataType type) : type(type), data(NULL){}
    DataType getType() const { return type; }
};
```

```
DataType::TYPE8U
```

```
Mat::DataType::TYPE8U
```

nested-enum2.cpp

Nested Classes

- Nested classes: The declaration of a class/struct or union may appear inside another class.

```
class Storage
{
public:
    class Fruit
    {
        string name;
        int weight;
    public:
        Fruit(string name="", int weight=0);
        string getInfo();
    };
private:
    Fruit fruit;
public:
    Storage(Fruit f);
};
```

nestedclass.cpp

Nested Types: Scope

Private:

- Only visible to the containing class

Protected:

- Visible to the containing class and its derived class.

Public:

- Visible to the containing class, to its derived classes, and to the outside world.

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

RTTI and Type Cast Operators

Runtime Type Identification (RTTI)

- We can convert a pointer explicitly to another, even it isn't appropriate.
- How to convert safely?

```
class Person;
class Student: public Person;

Person person("Yu");
Student student("Sam", "20210212");
Person* pp = &student;
Person& rp = student;
Student * ps = (Student*)&person; //danger!
```

rtti.cpp

RTTI and Type Cast Operators

- Runtime type identification (RTTI)
  - C++ feature
  - The type of an object to be determined during runtime.

- `dynamic_cast` operator: conversion of polymorphic types.

- `typeid` operator: Identify the exact type of an object.

- `type_info` class. the type information returned by the `typeid` operator.

typeid

- typeid operator
  - determine whether two objects are the same type
  - Accept: the name of a class, an expression that evaluates to an object
- type_info class
  - The typeid operator returns a reference to a type_info object
  - Defined in the <typeinfo> header file
  - Comparing type using the overloaded == and != operators

typeid.cpp

dynamic_cast

- It can safely assign the address of an object to a pointer of a particular type.
- Invoke the correct version of a class method (remember virtual functions)

```
Person person("Yu");
Student student("Sam", "20210212");
Person* pp = NULL;
Student * ps = NULL;

ps = dynamic_cast<Student*>(&person); // NULL
pp = dynamic_cast<Person*>(&student);
```

More Type Cast Operators

Three more operators

- `const_cast`:
  - Type cast for const or volatile value

  `const_cast.cpp`

- `static_cast`:
  - It's valid only if `type_name` can be converted implicitly to the same type that expression has, or vice versa
  - Otherwise, the type cast is an error

```
Base * pB = static_cast<Base*>(derived); //valid
Derived * pD = static_cast<Derived*>(base); //valid
UnRelated * pU = static_cast<UnRelated*>(base); //invalid
```

More Type Cast Operators

- `reinterpret_cast`
  - Converts between types by reinterpreting the underlying bit pattern.

```
int i = 18;
float * p1 = reinterpret_cast<float *>(i); // static_cast will fail
int * p2 = reinterpret_cast<int*>(p1);
```

reinterpret_cast.cpp

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

The Style

A Typical Object-oriented Style

- Is it a good style?

```
class Matrix
{
  private:
    size_t rows;
    size_t cols;
    float * data;
  public:
    Matrix(size_t r, size_t c){...}
    size_t getRows() const {return rows;}
    size_t getCols() const {return cols;}
    float getElement(size_t r, size_t c)
    {
        if( r >= rows || c >= cols) { ... }
        return data[ r * cols + c];
    }
    void setElement(size_t r, size_t c, float v)
    {
         if( r >= rows || c >= cols) { ... }
         data[ r * cols + c] = v;
    }
```

```
...
    Matrix operator+(Matrix & other)
    {
        if(this->rows != other.rows ||
           this->cols != other.rows)
        { ... }
        Matrix result(this->rows, this->cols);

        for (size_t r = 0; r < this->rows; r++)
            for (size_t c = 0; c < this->cols; c++)
            {
                float v = this->getElement(r, c) +
                          other.getElement(r, c);
                result.setElement(r, c, v);
                //result(r, c) = (*this)(r, c) + other(r, c);
            }

         return result;
    }
};
```

Another Style: Computational efficiency first

- Another style: Is it a good style?

```
class Matrix
{
  pubic:
    size_t rows;
    size_t cols;
    float * data;
  public:
    Matrix(size_t r, size_t c){...}
    size_t getRows() const {return rows;}
    size_t getCols() const {return cols;}
    float getElement(size_t r, size_t c)
    {
        if( r >= rows || c >= cols) { ... }
        return data[ r * cols + c];
    }
    void setElement(size_t r, size_t c, float v)
    {
         if( r >= rows || c >= cols) { ... }
         data[ r * cols + c] = v;
    }
```

```
...
    Matrix operator+(Matrix & other)
    {
        if(this->rows != other.rows ||
           this->cols != other.rows)
        { ... }
        Matrix result(this->rows, this->cols);

        size_t length = rows * cols;
        for (size_t i = 0; i < length; i++)
        {
            result.data[i] = this->data[i] + other.data[i];
        }

         return result;
    }
};
```

Java vs C++: Different Philosophy

Developer: I may have some stupid colleagues.

Java: I may have some stupid developers.

Developer: Show my talent!

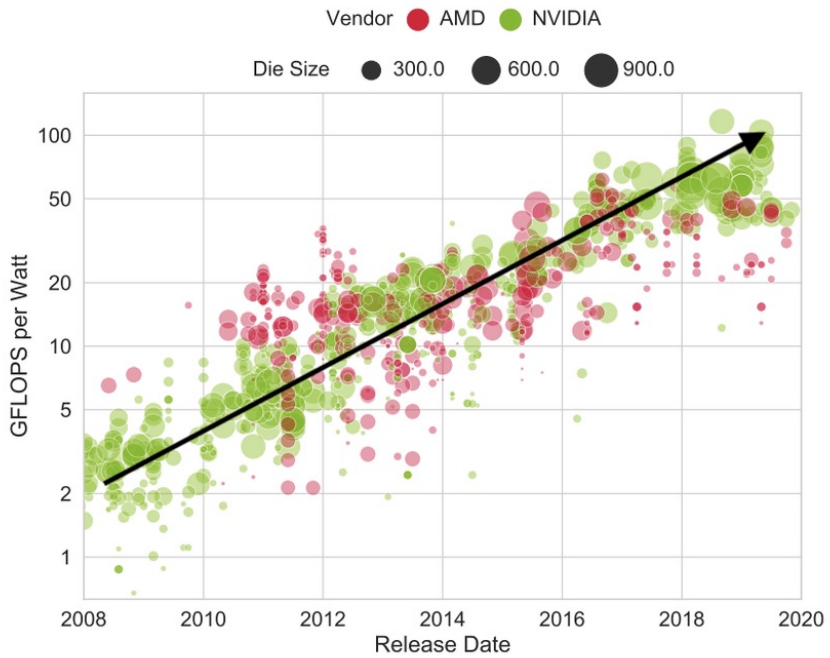
C++: Show your talent!

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

The Trend

What will happen?

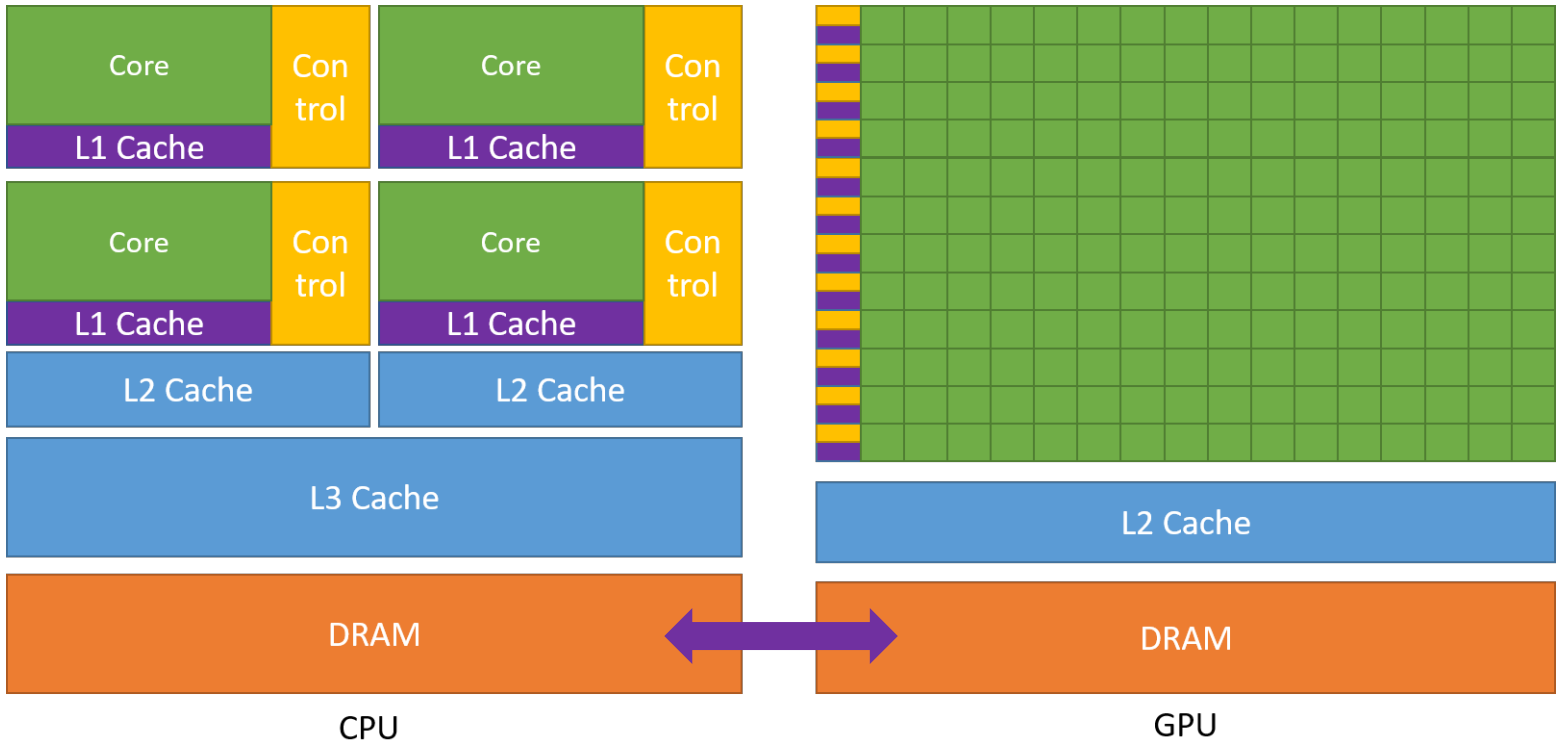
- I cannot predict, but …
- Moore's law
  - ➢ the number of transistors on a microchip doubled about 18 months.



Wu, Carole-Jean & Manne, Srilatha & Ranganathan, Parthasarathy & Bird, Sarah & Greenstein, Shane. (2021). Socio-Technological Challenges and Opportunities: Paths Forward.

The Architecture

CPU -> **CPU + *PU (GPU, NPU, TPU, etc.)**



https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

What's your position?

Hardware, OS, DB, Cloud, Deep Learning, C/C++/Java/Python/JS, …