



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/>



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Functions



Why functions?

- A compound statement may be needed to execute many times
- You can copy them several times, but ...

```
Matrix matA;  
float maxa = FLT_MIN;  
for(int r = 0; r < matA.rows; r++)  
    for (int c = 0; c < matA.cols; c++)  
    {  
        float val = matA.pData[ r * matA.cols + c];  
        maxa = ( maxa > val ? maxa : val);  
    }
```

```
struct Matrix  
{  
    int rows;  
    int cols;  
    float * pData;  
};
```

nofunction.cpp



Why functions?

- We can put the compound statement into a function

```
float matrix_max(struct Matrix mat)
{
    float max = FLT_MIN;
    for(int r = 0; r < mat.rows; r++)
        for (int c = 0; c < mat.cols; c++)
        {
            float val = mat.pData[ r * mat.cols + c];
            max = ( max > val ? max : val);
        }
    return max;
}
```

```
float maxa = matrix_max(matA);
float maxb = matrix_max(matB);
float maxc = matrix_max(matC);
```



A Question

- If `Matrix::pData` is `NULL` or an invalid value, how to tell the calling function from the called one?
- The pointer should be checked first!

```
float matrix_max(struct Matrix mat)
{
    float max = FLT_MIN;
    for(int r = 0; r < mat.rows; r++)
        for (int c = 0; c < mat.cols; c++)
        {
            float val = mat.pData[ r * mat.cols + c];
            max = ( max > val ? max : val);
        }
    return max;
}
```



Where should a function be? Option 1

```
// draw.cpp
// The function must be defined before it was called

bool drawLine(int x1, int y1, int x2, int y2)
{
    // Source code here
    return true;
}

bool drawRectangle(int x1, int y1, int x2, int y2)
{
    // some calculation here
    drawLine(...);
    drawLine(...);
    drawLine(...);
    drawLine(...);

    return true;
}
```



Where should a function be? Option 2

```
// draw.cpp
// declared first, parameter names can be omitted
bool drawLine(int x1, int y1, int x2, int y2);

bool drawRectangle(int x1, int y1, int x2, int y2)
{
    // some calculation here
    drawLine(...);
    drawLine(...);
    drawLine(...);
    drawLine(...);

    return true;
}

// define it later
bool drawLine(int x1, int y1, int x2, int y2)
{
    // Source code here
    return true;
}
```



Where should a function be? Option 3

```
// draw.h
#ifndef __DRAW_H__
#define __DRAW_H__
bool drawLine(int x1, int y1, int x2, int y2);
bool drawRectangle(int x1, int y1, int x2, int y2);
#endif
```

```
// draw.cpp
#include <draw.h>

bool drawRectangle(int x1, int y1, int x2, int y2)
{
    // some calculation here
    drawLine(...);
    drawLine(...);
    drawLine(...);
    drawLine(...);

    return true;
}
```

```
// define it later
bool drawLine(int x1, int y1, int x2, int y2)
```

```
// main.cpp
#include <draw.h>

int main()
{
    // ...
    drawRectangle(10, 20,
    // ...
}
```




How are functions called?

- A call stack can store information about the active functions of a program
 - Store the address the program returns after the function call
 - Store the registers
 - Store the local variables
 - `//do some work of the called function`
 - Restore the registers
 - Restore the local variables
 - Store the function returned result
 - Jump to the return address

The cost to call a function!



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Function Parameters



Parameters

- The symbolic name for "data" that passes into a function.

Two ways to pass into a function:

- **Pass by value**
- **Pass by reference**



Pass by value: fundamental type

- The parameter is a copy of the original variable

```
int foo(int x)
{
    // x is a copy
    x += 10;
    return x;
}
```

Will `num1` be changed in `foo()`?

```
int main()
{
    int num1 = 20;
    int num2 = foo( num1 );

    return 0;
}
```





Pass by value: pointer

- What's the difference?

```
int foo(int * p)
{
    (*p) += 10;
    return *p;
}
```

It still is passing by value (the address!)
A copy of the address

```
int main()
{
    int num1 = 20;
    int * p = &num1;
    int num2 = foo( p );
    return 0;
}
```





Pass by value: structure

- How about structure parameter?

```
struct Matrix
{
    int rows;
    int cols;
    float * pData;
};

float matrix_max(struct Matrix mat)
{
    float max = FLT_MIN;
    for(int r = 0; r < mat.rows; r++)
        for (int c = 0; c < mat.cols; c++)
        {
            float val = mat.pData[ r * mat.cols + c];
            max = ( max > val ? max : val);
        }
    return max;
}
```

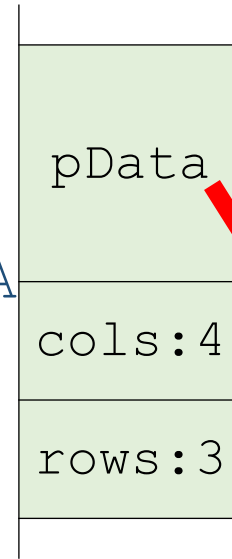


Pass by value: structure

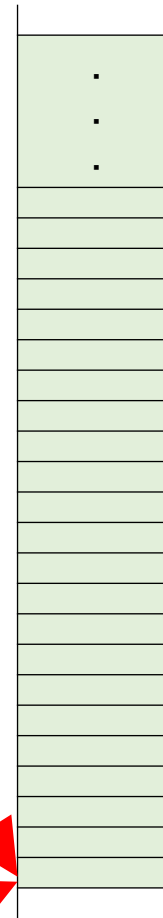
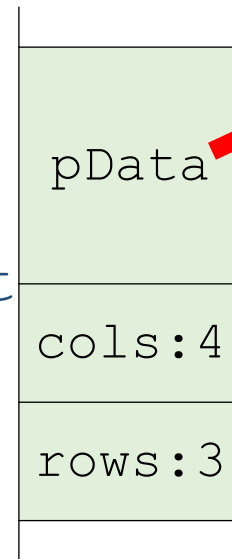
```
Matrix matA = {3,4};  
matrix_max(matA);
```

```
float matrix_max(struct Matrix mat)  
{  
    float max = FLT_MIN;  
    for(int r = 0; r < mat.rows; r++)  
        for (int c = 0; c < mat.cols; c++)  
        {  
            float val = mat.pData[ r * mat.cols + c];  
            max = ( max > val ? max : val);  
        }  
    return max;  
}
```

matA



mat





Pass by value: structure

- If the structure is a huge one, such as 1K bytes.
- A copy will cost 1KB memory, and time consuming to copy it.





南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

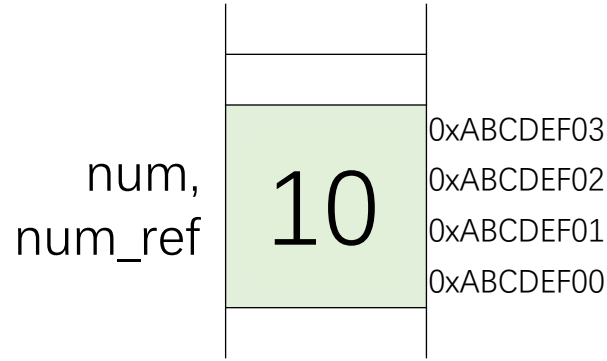
References



References in C++

- References are in C++, not in C.
- A reference is an alias to an already-existing variable/object.

```
int num = 0;  
int & num_ref = num;  
  
num_ref = 10;
```

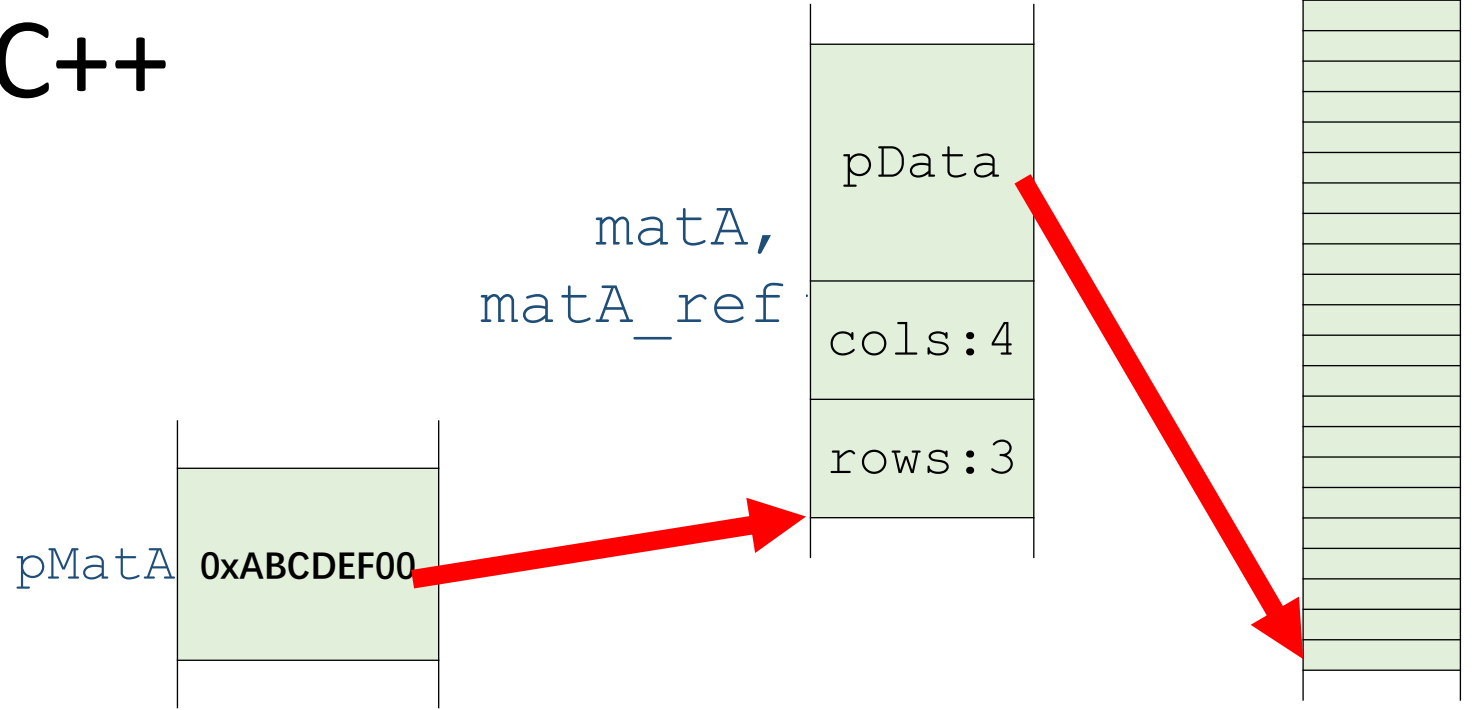




References in C++

- A reference to an object

```
struct Matrix
{
    int rows;
    int cols;
    float * pData;
};
```



```
Matrix matA = {3,4};
matA.pData = new float[matA.rows * matA.cols]{};
```

```
Matrix & matA_ref = matA;
```

```
Matrix * pMatA = &matA;
```



References in C++

- A reference must be initialized after its declaration.

```
int & num_ref; // error  
Matrix & mat_ref; // error
```

- Reference VS Pointer: References are much safer



Function parameters with a huge structure

- If the huge struct is passed as a function parameter

```
struct PersonInfo
{
    char firstname[256];
    char middlename[256];
    char lastname[256];
    char address[256];
    char nationalID[16];
    // and more
};

char * fullname(struct PersonInfo pi)
{
    // ...
}
```

- The data will be copied. Not a good choice!



The problem

- One solution is to use a pointer

```
struct PersonInfo
{
    char firstname[256];
    char middlename[256];
    char lastname[256];
    char address[256];
    char nationalID[16];
    // and more
};
```

```
char * fullname(struct PersonInfo * ppi)
```

```
{
    if (ppi == NULL)
    {
        cerr << "Invalid pointer" << endl;
        return NULL;
    }
```

```
// ...
}
```





References as function parameters

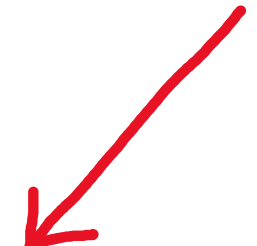
- No data copying in the reference version; Better efficiency
- The modification to a reference will affect the original object

```
struct Matrix
{
    int rows;
    int cols;
    float * pData;
};

float matrix_max(struct Matrix mat)
{
    float max = FLT_MIN;
    //find max value of mat
    for(int r = 0; r < mat.rows; r++)
        for (int c = 0; c < mat.cols; c++)
        {
            float val = mat.pData[ r * mat.cols + c];
            max = ( max > val ? max : val);
        }
    return max;
}
```

```
struct Matrix
{
    int rows;
    int cols;
    float * pData;
};

float matrix_max(struct Matrix & mat)
{
    float max = FLT_MIN;
    //find max value of mat
    for(int r = 0; r < mat.rows; r++)
        for (int c = 0; c < mat.cols; c++)
        {
            float val = mat.pData[ r * mat.cols + c];
            max = ( max > val ? max : val);
        }
    return max;
}
```


A red arrow is drawn on the right side of the slide, pointing from the top right towards the parameter '& mat' in the function signature 'float matrix_max(struct Matrix & mat)'. This highlights the use of a reference parameter.



References as function parameters

- To avoid the data is modified by mistakes,

```
float matrix_max(const struct Matrix & mat)
{
    float max = FLT_MIN;
    // ...
    return max;
}
```

A thick red arrow is drawn on the slide, pointing from the lower right towards the 'const' keyword in the function signature 'const struct Matrix & mat'. The arrow starts near the bottom right of the code block and points diagonally upwards and to the left, ending directly under the 'const' keyword.



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Return statement



Return statement

- Statement `return;` is only valid if the function return type is `void`.
- Just finish the execution of the function, no value returned.

```
void print_gender(bool isMale)
{
    if(isMale)
        cout << "Male" << endl;
    else
        cout << "Female" << endl;

    return;
}
```

```
void print_gender(bool isMale)
{
    if(isMale)
        cout << "Male" << endl;
    else
        cout << "Female" << endl;
}
```



Return statement

- The return type can be a fundamental type or a compound type.
- Pass by value:
 - Fundamental types: the value of a constant/variable is copied
 - Pointers: the address is copied
 - Structures: the whole structure is copied

```
float maxa = matrix_max(matA);
```

```
Matrix * pMat = create_matrix(4,5);
```

```
Matrix * create_matrix(int rows, int cols)
{
    Matrix * p = new Matrix{rows, cols};
    p->pData = new float[p->rows * p->cols]{1.f, 2.f, 3.f};
    // you should check if the memory is allocated successfully
    // and don't forget to release the memory
    return p;
}
```



If we have a lot to return

- Such as a matrix addition function ($A+B \rightarrow C$)
- A suggested prototype:
 - To use references to avoid data copying
 - To use const parameters to avoid the input data is modified
 - To use non-const reference parameters to receive the output

```
bool matrix_add(const Matrix & matA, const Matrix & matB, Matrix & matC)
{
    // check the dimensions of the three matrices
    // re-create matC if needed
    // do: matC = matA + matB
    // return true if everything is right
}
```



Similar mechanism in OpenCV

- Matrix add in OpenCV

<https://github.com/opencv/opencv/blob/master/modules/core/src/arithm.cpp>

```
→ ↻ 🔒 https://github.com/opencv/opencv/blob/master/modules/core/src/arithm.cpp 🔍 ☆ 📄  
912  
913 void cv::add( InputArray src1, InputArray src2, OutputArray dst,  
914               InputArray mask, int dtype )  
915 {  
916     CV_INSTRUMENT_REGION();  
917  
918     arithm_op(src1, src2, dst, mask, dtype, getAddTab(), false, 0, OCL_OP_ADD );  
919 }  
920
```



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Inline functions



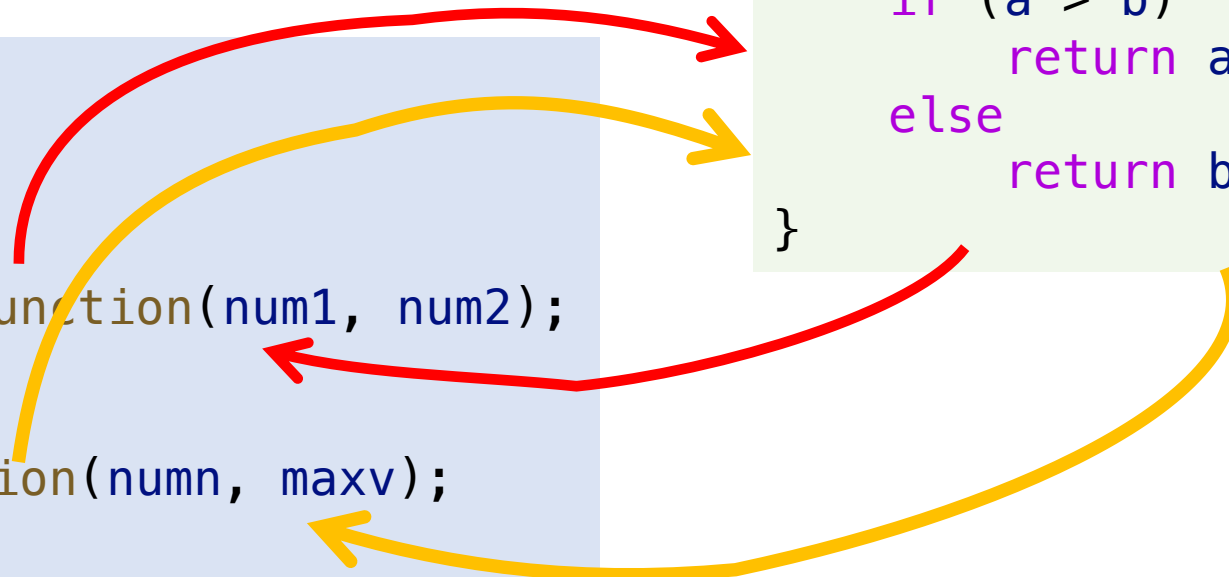
Inline functions

- Stack operations and jumps are needed for a function call.
- It is a heavy cost for some frequently called **tiny** functions.

```
int main()
{
    int num1 = 20;
    int num2 = 30;
    int maxv = max_function(num1, num2);

    maxv = max_function(numn, maxv);
}
```

```
float max_function(float a, float b)
{
    if (a > b)
        return a;
    else
        return b;
}
```





Inline functions

- The generated instructions by a compiler can be as follows to improve efficiency

```
int main()  
{  
    int num1 = 20;  
    int num2 = 30;  
    int maxv =  
        {if (num1 > num2)  
            return num1;  
        else  
            return num2;}  
    maxv =  
        {if (numn > maxv)  
            return numn;  
        else  
            return maxv;}  
}
```




Inline functions

- **inline suggests** the compiler to perform that kind of optimizations.
- The compiler may not follow your suggestion if the function is too complex or contains some constraints.
- Some functions without **inline** may be optimized as an inline one.

```
inline float max_function(float a, float b)
{
    if (a > b)
        return a;
    else
        return b;
}
```



Why not use a macros?

```
#define MAX_MACRO(a, b) (a)>(b) ? (a) : (b)
```

- The source code will be replaced by a preprocessor.
- Surely no cost of a function call,
- And `a`, `b` can be any types which can compare.

`inline.cpp`

Inline in OpenCV



inline



Pull request

Code

735

Commits

52

Issues

559

Discussions Beta

0

Packages

0

Wikis

9

Languages

C++ 562

C 124

OpenCL 20

CMake 7

Python 4

CSS 1

735 code results

3rdparty/caroten

```
94  template <> struct VecTraits<f32, 4> { typedef float32x4x4_t vec128; typedef
    float32x2x4_t vec64; typedef VecTraits< u32, 3> unsign; };
95
96  ////////////////////////////////// vld1q //////////////////////////////////
97
98  inline uint8x16_t vld1q(const u8 * ptr) { return vld1q_u8(ptr); }
99  inline int8x16_t vld1q(const s8 * ptr) { return vld1q_s8(ptr); }
100 inline uint16x8_t vld1q(const u16 * ptr) { return vld1q_u16(ptr); }
```

● C++ Showing the top three matches Last indexed on 26 Jun 2018

modules/imgproc/src/fixedpoint.inl.hpp

```
18  fixedpoint64(int64_t _val) : val(_val) {}
19  static CV_ALWAYS_INLINE uint64_t fixedround(const uint64_t& _val) { return
    ((1LL << fixedShift) >> 1)); }
...
24  typedef int64_t raw_t;
25  CV_ALWAYS_INLINE fixedpoint64() { val = 0; }
26  CV_ALWAYS_INLINE fixedpoint64(const fixedpoint64& v) { val = v.val; }
```