Université de Paris

Programmation avancée 2 – Introduction to C++

Sylvain Lobry

26/09/2022

Université de Paris

Sondage: https://www.wooclap.com/PROGAC2

Université de Paris

Menu of the day:

- Dynamic memory, C++ style
- Exceptions
- Templates
- C++ STL (iterators + containers)
- Function’s parameters
- Namespaces

C++ 101

Dynamic memory

- Reminder: a program has access to a stack and a heap to deal with memory.
- No garbage collection in C++ (at least not in std)
- As most of the time, you can use C functions: malloc/free/…
- However you should use new/delete (and new[]/delete[] for arrays)
- Main reason: they call constructor/destructor.
- Careful: new/delete and malloc/free should not be swapped (i.e. do not free an element allocated with new)

memory.cpp

```cpp
#include <iostream>

int main ()
{
    int * test_single = new int (42);
    int * test_multiple = new int [42]; //Careful: not initialized
    delete test_single;
    delete [] test_multiple;
    return 0;
}
```

C++ 101

Dynamic memory

- If allocation is not possible, new will throw a bad_alloc exception
- Other method: new (nothrow) int [42];
- Will return a null pointer (which can be checked)

- No garbage collection in C++ (at least not in std)… but smart pointers (out of the scope of this class)

C++ 101

Exceptions

– Allows to react to potential problems

exception.cpp

```cpp
#include <iostream>

int main ()
{
    try
    {
        throw 42;
    }
    catch (int ex)
    {
        std::cout << "Caught exception " << ex << std::endl;
    }
    return 0;
}
```

```
Caught exception 42
```

C++ 101

Exceptions

- Allows to react to potential problems

fail_exception.cpp

```cpp
#include <iostream>

int main ()
{
    throw 42;
    return 0;
}
```

```
libc++abi.dylib: terminating with uncaught exception of type int
zsh: abort      ./fail_exception
```

C++ 101

Exceptions

- Allows to react to potential problems
- 3 keywords:
  - try: defines a block on which we want to detect particular exceptions
  - catch: defines that you want to catch an exception, and which one(s)
  - throw: to manually raise an exception

C++ 101

Exceptions

- Allows to react to potential problems
- Possible to catch any exception with catch (…)
- Standard exceptions in header exception:
  - bad_alloc
  - bad_cast
  - bad_exception
  - bad_typeid
  - bad_function_call
  - bad_weak_ptr
  - logic_error
  - runtime_error
- Possibility to define new ones (out of the scope of this class)

fail_exception.cpp

```cpp
#include <iostream>
#include <exception>

int main ()
{
    try
    {
        int count = 0;
        while (true)
        {
            std::cout << count++ << std::endl;
            int * oops = new int [100000000000000];
        }
    }
    catch (std::bad_alloc& e)
    {
        std::cout << "Cannot allocate." << std::endl;
    }
}
```

C++ 101

Template

- Paradigm: generic programming
- Template: allows to define function or class once for every type
- Powerful tool (design patterns)

template.cpp

```cpp
#include <iostream>

template <typename T>
T add(T a, T b)
{
    return a + b;
}

int main()
{
    float a = 0.7;
    float b = 4.2;

    std::cout << add<float>(a, b) << std::endl;
    std::cout << add<int>(a, b) << std::endl;

    return 0;
}
```

```
4.9
4
```

C++ 101

C++ STL

- C++ Standard Template Library (STL) provides common functions and data structures.
- 4 parts:
  - Algorithms
  - **Containers**
  - Functions
  - **Iterators**

C++ 101

Iterators

- Standard method to iterate through an element (e.g. string, vector, …)
- How do you do that usually ?

C++ 101

Iterators

- Standard method to iterate through an element (e.g. string, vector, …)
- How do you do that usually ?

- Warning: only work for sequential structures!
- Defined by all the STD containers:
  - container.begin()
  - container.end()
- Iterator can be incremented or decremented
- Access to the element by the dereferencing operator *

C++ 101

Iterators

– Also defined on strings

string.cpp

```cpp
#include <iostream>
#include <string>

int main ()
{
    std::string test = "hello";
    for (int i = 0; i < test.length(); ++i)
        std::cout << test[i] << " ";
    std::cout << std::endl;
    for (std::string::iterator it = test.begin(); it != test.end(); ++it)
        std::cout << *it << " ";
    return 0;
}
```

C++ 101

Containers

- Standard library provides widely-used containers
- Common interface (e.g. iterators)
- They are generic!

vector.cpp

```cpp
#include <iostream>
#include <vector>

int main ()
{
    std::vector<int> my_vec (3, 42);
    my_vec.insert(my_vec.begin(), 0);
    std::cout << "Vector of size " << my_vec.size() << std::endl;
    for (std::vector<int>::iterator it = my_vec.begin(); it != my_vec.end(); ++it)
        std::cout << *it << "; ";
}
```

```
Vector of size 4
0; 42; 42; 42; %
```

C++ 101

Containers

- Standard library provides widely-used containers
- Common interface (e.g. iterators)
- They are generic!
- List of containers in C++98:
  - vector
  - dequeue (double ended queue)
  - list
  - set
  - map

C++ 101

Function's parameters

- How can you pass a parameter to a function in C?
  - Value: cannot be modified, potential heavy copy
  - Address: heavy syntax, dangerous!

C++ 101

Function's parameters

- How can you pass a parameter to a function in C?
  - Value: cannot be modified, potential heavy copy
  - Address: heavy syntax, dangerous!
- Solution in C++: reference

exRef.cpp

```cpp
#include <iostream>
#include <string>

void modify_str(std::string& str)
{
    str += ".";
}

int main()
{
    std::string a_str = "test";
    modify_str(a_str);
    std::cout << a_str << std::endl;
    return 0;
}
```

C++ 101

Function's parameters

- How can you pass a parameter to a function in C?
  - Value: cannot be modified, potential heavy copy
  - Address: heavy syntax, dangerous!
- Solution in C++: reference

exRef.cpp

```cpp
#include <iostream>
#include <string>

void modify_str(std::string& str)
{
    str += ".";
}

void modify_str(std::string* str)
{
    *str += "..";
}

int main()
{
    std::string a_str = "test";
    modify_str(&a_str);
    std::cout << a_str << std::endl;
    return 0;
}
```

C++ 101

Function's parameters

- How can you pass a parameter to a function in C?
  - Value: cannot be modified, potential heavy copy
  - Address: heavy syntax, dangerous!
- Solution in C++: reference

exRef.cpp

```cpp
#include <iostream>
#include <string>

void print_str(std::string& str)
{
    str = "I don't care.";
    std::cout << str << std::endl;
}

int main()
{
    std::string a_str = "test";
    print_str(a_str);
    return 0;
}
```

C++ 101

Function's parameters

- How can you pass a parameter to a function in C?
  - Value: cannot be modified, potential heavy copy
  - Address: heavy syntax, dangerous!
- Solution in C++: reference

exRef.cpp

```cpp
#include <iostream>
#include <string>

void print_str(const std::string& str)
{
    str = "I don't care.";
    std::cout << str << std::endl;
}

int main()
{
    std::string a_str = "test";
    print_str(a_str);
    return 0;
}
```

```
exRef.cpp:6:6: error: no viable overloaded '='
        str = "I don't care.";
        ~~~ ^ ~~~~~~~~~~~~~~~~
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchai
sr/bin/../include/c++/v1/string:890:45: note:
      candidate function not viable: 'this' argument has type
      'const std::string' (aka 'const basic_string<char, char_traits<char>,
      allocator<char> >'), but method is not marked const
    _LIBCPP_INLINE_VISIBILITY basic_string& operator=(const value_type* ...
                                            ^
```

C++ 101

Namespaces

- Did you ever get a redefinition error with C?
- Namespaces allow to group entities under a name

namespace.cpp

```cpp
#include <iostream>

namespace PA
{
    std::string a_var = "We are in PA!";

    void fancy_print(const std::string& a_string)
    {
        std::cout << "A very fancy print in PA!" << std::endl
                  << "Here is the string requested:" << std::endl
                  << a_string << std::endl
                  << a_var << std::endl;
    }
}

std::string a_var = "We are not in PA";

void fancy_print(const std::string& a_string)
{
    std::cout << "Here is the string requested:" << std::endl
              << a_string << std::endl
              << a_var << std::endl;
}

int main ()
{
    std::string test = "Test";
    fancy_print(test);
    std::cout << "=======" << std::endl;
    PA::fancy_print(test);
    return 0;
}
```

C++ 101

Namespaces

- Did you ever get a redefinition error with C?
- Namespaces allow to group entities under a name

```
Here is the string requested:
Test
We are not in PA
======
A very fancy print in PA!
Here is the string requested:
Test
We are in PA!
```

namespace.cpp

```cpp
#include <iostream>

namespace PA
{
    std::string a_var = "We are in PA!";

    void fancy_print(const std::string& a_string)
    {
        std::cout << "A very fancy print in PA!" << std::endl
                  << "Here is the string requested:" << std::endl
                  << a_string << std::endl
                  << a_var << std::endl;
    }
}

std::string a_var = "We are not in PA";

void fancy_print(const std::string& a_string)
{
    std::cout << "Here is the string requested:" << std::endl
              << a_string << std::endl
              << a_var << std::endl;
}

int main ()
{
    std::string test = "Test";
    fancy_print(test);
    std::cout << "======" << std::endl;
    PA::fancy_print(test);
    return 0;
}
```

Conclusion

- Very basics of C++
- However, « C with classes »
- Next Monday…
- During the labs: simple exercices in C++

Programmation avancée - Homework 1

- Language: C++
- Starts 27/09, Due 02/10
- Platform: Moodle
- You should only provide the cpp files (1/exercise). You can also provide a README.txt that explains how to compile and run your programs. The code must be packaged in a zip file named LASTNAME_HW1.zip
- For each exercise, you can only include the headers that are mentioned in the instructions. You do not have to use all of them. Including a header not mentioned in the instructions will be considered as cheating and treated accordingly.
- If any problem or something is not clear, contact Sylvain Lobry by email (@u-paris.fr).

Questions?