Exploring Polymorphism in C++

Run-time vs. Compile-time

Presentation Material

San Diego C++ Meetup, Online, 2024-02-20

Andreas Fertig

Write unique code!

© 2024 Andreas Fertig
AndreasFertig.com
All rights reserved

All programs, procedures and electronic circuits contained in this book have been created to the best of our knowledge and belief and have been tested with care. Nevertheless, errors cannot be completely ruled out. For this reason, the program material contained in this book is not associated with any obligation or guarantee of any kind. The author therefore assumes no responsibility and will not accept any liability, consequential or otherwise, arising in any way from the use of this program material or parts thereof.

Version: v1.0

The work including all its parts is protected by copyright. Any use beyond the limits of copyright law requires the prior consent of the author. This applies in particular to duplication, processing, translation and storage and processing in electronic systems.

The reproduction of common names, trade names, product designations, etc. in this work does not justify the assumption that such names are to be regarded as free in the sense of trademark and brand protection legislation and can therefore be used by anyone, even without special identification.

Planning, typesetting and cover design: Andreas Fertig
Cover art and illustrations: Franziska Panter https://franziskapanter.com
Production and publishing: Andreas Fertig

Style and conventions

The following shows the execution of a program. I used the Linux way here and skipped supplying the desired output name, resulting in `a.out` as the program name.

```
$ ./a.out
Hello, C++!
```

- `<string>` stands for a header file with the name `string`
- `[[xyz]]` marks a C++ attribute with the name `xyz`.

© 2024 Andreas Fertig
https://AndreasFertig.com
post@AndreasFertig.com

Run-time polymorphism and its costs

```
struct Car {
  virtual ~Car() = default;
  void Drive(eDirect d) const
  {
    printf("driving a 4WD: %d\n", Is4WD());
  }

  virtual bool Is4WD() const = 0;
};

struct Tesla : public Car {
  bool Is4WD() const override { return false; }
};

struct Toyota : public Car {
  bool Is4WD() const override { return true; }
};

void DriveCar(Car& vehicle);
```


Curiously Recurring Template Pattern

- **C**uriously **R**ecurring **T**emplate **P**attern (CRTP) refers to the case when a class is derived from a class template that owns itself as a class template.
  - Basically, CRTP corresponds to static polymorphism.

```
template<class T>
class Base {
  // ...
};

class  Derived
: public Base< Derived > {
  // ...
};
```


© 2024 Andreas Fertig
https://AndreasFertig.com
post@AndreasFertig.com

Static polymorphism using CRTP

```cpp
template<typename T>
struct Car {
  void Drive(eDirect d) const
  {
    printf("driving a 4WD: %d\n",
           static_cast<const T*>(this)->Is4WD());
  }
};

struct Tesla : public Car<Tesla> {
  bool Is4WD() const { return false; }
};

struct Toyota : public Car<Toyota> {
  bool Is4WD() const { return true; }
};

template<typename T>
void DriveCar(T& vehicle);
```


Curiously Recurring Template Pattern

```cpp
class CharStorage {
  static constexpr int mSize{5};

public:
  char mData[mSize]{};

  char*       begin() { return mData; }
  const char* begin() const { return mData; }
  char*       end() { return mData + mSize; }
  const char* end() const { return mData + mSize; }

  char& front() { return *begin(); }
  char& back() { return *std::prev(end()); }
  auto  size() const
  {
    return std::distance(begin(), end());
  }
  char& operator[](size_t i)
  {
    return *std::next(begin(), i);
  }
};
```


© 2024 Andreas Fertig
https://AndreasFertig.com
post@AndreasFertig.com

Curiously Recurring Template Pattern

```cpp
template<typename T>
class Container {
  T&       impl() { return *static_cast<T*>(this); }
  const T& impl() const { return *static_cast<const T*>(this); }

public:
  auto& front() { return *impl().begin(); }
  auto& back() { return *std::prev(impl().end()); }
  auto  size() const { return std::distance(impl().begin(), impl().end()); }
  auto& operator[](size_t i) { return *std::next(impl().begin(), i); }
};

class CharStorage : public Container<CharStorage> {
  static constexpr int mSize{5};

public:
  char mData[mSize]{};

  char*       begin() { return mData; }
  const char* begin() const { return mData; }
  char*       end() { return mData + mSize; }
  const char* end() const { return mData + mSize; }
};
```


Policy based design

> In brief, policy-based class design fosters assembling a class with complex behavior out of many little classes (called policies), each of which takes care of only one behavioral or structural aspect. ”
>
> — Alexandrescu [1]

> A policy is a class or class template that defines an interface as a service to other classes. ”
>
> — Lischner [2]

> Traits define type interfaces, and policies define function interfaces, so they are closely related. ”
>
> — Lischner [2]

© 2024 Andreas Fertig
https://AndreasFertig.com
post@AndreasFertig.com

The deleter policy of `std::unique_ptr`

- A `std::unique_ptr` takes not one but two parameters.
- The second, often defaulted parameter is the `deleter`.

```
1 struct COMLikeObject {
2   ~COMLikeObject() = default;
3
4   void Release();
5   void Fun();
6 };
7
8 void Deleter(COMLikeObject* obj)
9 {
10   obj->Release();
11   delete obj;
12 }
13
14 void Use()
15 {
16   auto obj = std::unique_ptr<COMLikeObject, void (*)(COMLikeObject*)>(
17     new COMLikeObject{}, Deleter);
18
19   static_assert(sizeof(obj) == (sizeof(COMLikeObject*) * 2));
20 }
```

The deleter policy of `std::unique_ptr`

- A `std::unique_ptr` takes not one but two parameters.
- The second, often defaulted parameter is the `deleter`.

```
1 struct COMLikeObject {
2   ~COMLikeObject() = default;
3
4   void Release();
5   void Fun();
6 };
7
8 struct MyDeleter {
9   void operator()(COMLikeObject* obj)
10   {
11     obj->Release();
12     delete obj;
13   }
14 };
15
16 void Use()
17 {
18   auto obj = std::unique_ptr<COMLikeObject, MyDeleter>(new COMLikeObject{});
19
20   static_assert(sizeof(obj) == (sizeof(COMLikeObject*)));
21 }
```

© 2024 Andreas Fertig
https://AndreasFertig.com
post@AndreasFertig.com

The sort policy of `std::sort`

```cpp
std::array data{6, 7, 4, 2};

std::sort(data.begin(), data.end());

struct MyLess {
  bool operator()(int a, int b) const { return a < b; }
};

// different policy, same algorithm, same code/function
std::sort(data.begin(), data.end(), MyLess{});
```

Policy design decision

- When working with a policy, we have a design decision to make:
  - Derive from the policy
  - Store a data member of the policy type.

© 2024 Andreas Fertig
https://AndreasFertig.com
post@AndreasFertig.com

Policy design - Example

```
1  template<typename T, size_t SIZE>
2  struct Array {
3    T mData[SIZE];
4
5    T& operator[](size_t idx) { return mData[idx]; }
6  };
7
8  void Use()
9  {
10   Array<int, 2> ai{3, 5};
11
12   printf("%d\n", ai[2]);
13 }
```

Policy design - Example

```
1  template<typename T, size_t SIZE, bool checked = false>
2  struct Array {
3    T mData[SIZE];
4
5    T& operator[](size_t idx)
6    {
7      if constexpr (checked) { assert(idx < SIZE); }
8
9      return mData[idx];
10   }
11 };
12
13 void Use()
14 {
15   Array<int, 2> ai{3, 5};
16   printf("%d\n", ai[2]);
17
18   Array<int, 2, true> safe{3, 5};
19   printf("%d\n", safe[2]);
20 }
```

© 2024 Andreas Fertig
https://AndreasFertig.com
post@AndreasFertig.com

Policy design - Example

```
1  struct Unchecked {
2    void operator()(size_t idx, size_t max) {}
3  };
4
5  template<typename T, size_t SIZE, typename CheckingPolicy = Unchecked>
6  struct Array {
7    T mData[SIZE];
8
9    T& operator[](size_t idx)
10   {
11     CheckingPolicy{}(idx, SIZE);
12     return mData[idx];
13   }
14 };
15
16 struct Checked {
17   void operator()(size_t idx, size_t max) { assert(idx < max); }
18 };
19
20 void Use()
21 {
22   Array<int, 2> ai{3, 5};
23   printf("%d\n", ai[2]);
24
25   Array<int, 2, Checked> safe{3, 5};
26   printf("%d\n", safe[2]);
27 }
```


Policy-based design

- Each policy creates a new type and potential code!
- That code is fast! No vtable lookup and no space overhead.

© 2024 Andreas Fertig
https://AndreasFertig.com
post@AndreasFertig.com

}

I am Fertig.

https://AndreasFertig.com

Available online:

https://AndreasFertig.com

Images by Franziska Panter:

https://panther-concepts.de

Used Compilers & Typography

Used Compilers

- Compilers used to compile (most of) the examples.
  - GCC 13.2.0
  - Clang 17.0.0

Typography

- Main font:
  - Camingo Dos Pro by Jan Fromm (https://janfromm.de/)
- Code font:
  - CamingoCode by Jan Fromm licensed under Creative Commons CC BY-ND, Version 3.0 http://creativecommons.org/licenses/by-nd/3.0/

© 2024 Andreas Fertig
https://AndreasFertig.com
post@AndreasFertig.com

References

[1] Alexandrescu A., *Modern C++ Design: Generic Programming and Design Patterns Applied*, ser. C++ in-depth series. Addison-Wesley, 2001.

[2] Lischner R., *C++ In a Nutshell: A Desktop Quick Reference*, ser. In a Nutshell (O'Reilly). O'Reilly Media, 2003.

Images:
19: Franziska Panter

Upcoming Events

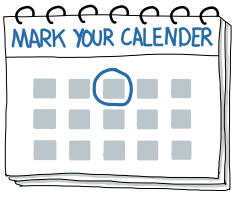
Talks

- *C++20's Coroutines for Beginners*, C++Online, March 01
- *C++20 Coroutinen - Ein Einstieg*, ADC, May 07

Training Classes

- *A day with coroutines*, C++Online, March 04
- *C++20 Coroutinen - Ein Einstieg*, ADC, May 06

For my upcoming talks you can check https://andreasfertig.com/talks/.
For my courses you can check https://andreasfertig.com/courses/.
Like to always be informed? Subscribe to my newsletter: https://andreasfertig.com/newsletter/.



© 2024 Andreas Fertig
https://AndreasFertig.com
post@AndreasFertig.com

About Andreas Fertig

Photo: Kristijan Matic www.kristijanmatic.de

Andreas Fertig, CEO of Unique Code GmbH, is an experienced trainer and consultant for C++ for standards 11 to 23.
Andreas is involved in the C++ standardization committee, developing the new standards. At international conferences, he presents how code can be written better. He publishes specialist articles, e.g., for iX magazine, and has published several textbooks on C++.
With C++ Insights (https://cppinsights.io), Andreas has created an internationally recognized tool that enables users to look behind the scenes of C++ and thus understand constructs even better.
Before training and consulting, he worked for Philips Medizin Systeme GmbH for ten years as a C++ software developer and architect focusing on embedded systems.
You can find Andreas online at andreasfertig.com.

© 2024 Andreas Fertig
https://AndreasFertig.com
post@AndreasFertig.com