Object Move, Copy & Destroy (1)

- The compiler generate implicit move, copy and destroy functions for you.
  - Unless you are allocating raw memory with new, the compiler generated functions are better optimized
- Generated functions are
  - Default constructor (unless non default is provided)
  - Copy constructor
  - Move constructor
  - Copy assignment
  - Move assignment   => not always generated !!
  - Destructor

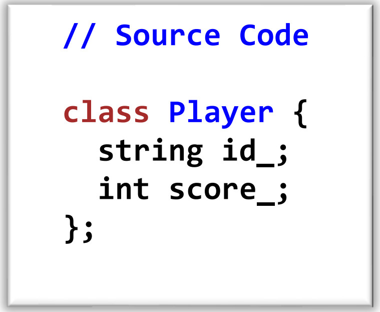
Object Move, Copy & Destroy (2)

- The compiler will create move, copy and destroy functions for you.
  - Unless you are allocating raw memory with new, the compiler generated functions are better optimized

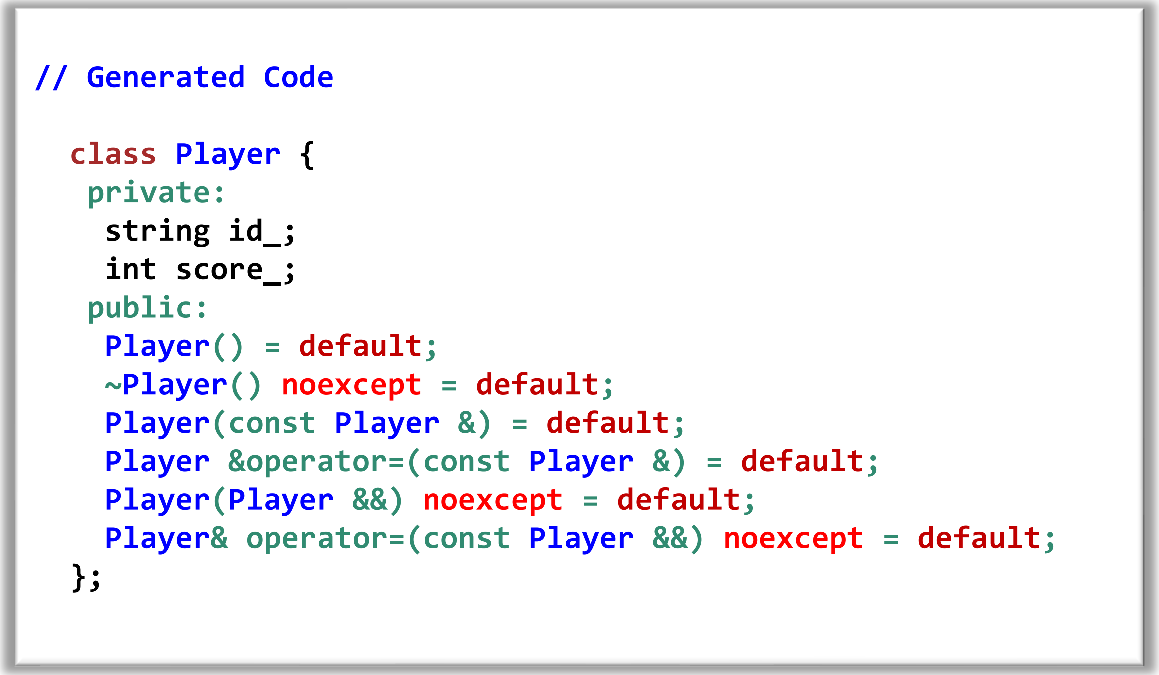
```
// Default Constructor
// => Player a;
Player();

// Copy Constructor
// => Player b{a};
// => Player b = a;
Player(const Player &player);

// Move Constructor
// => Player c{std::move(b)};
// => Player c = std::move(b);
Player(Player &&player) noexcept;
```

```
// Destructor
~Player() noexcept;

// Copy Assignment
// => Player d;
//    d = c;
Player &operator=(const Player &player);

// Move Assignment
// => Player e;
//    e = std::move(d);
Player &operator=(Player &&player) noexcept;
```

Object Move, Copy & Destroy (3)



```cpp
// Source Code

class Player {
  string id_;
  int score_;
};
```



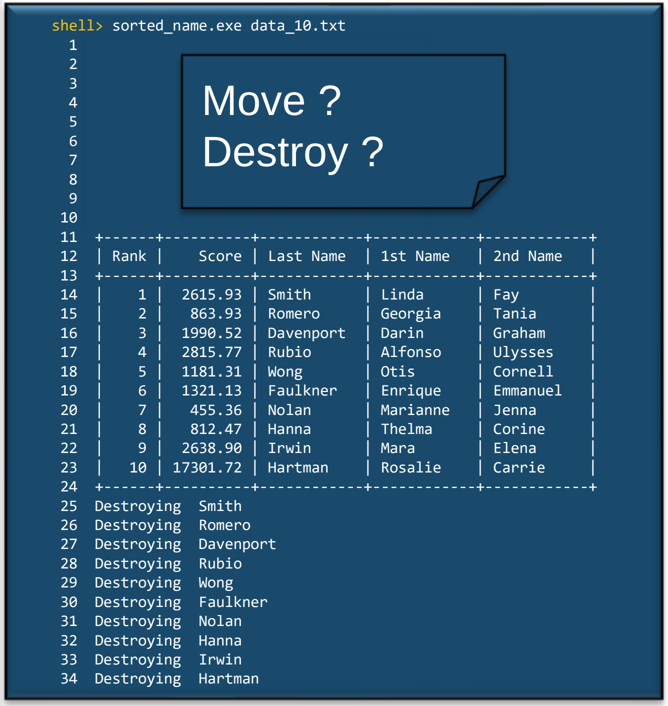
```cpp
// Generated Code

  class Player {
   private:
    string id_;
    int score_;
   public:
    Player() = default;
    ~Player() noexcept = default;
    Player(const Player &) = default;
    Player &operator=(const Player &) = default;
    Player(Player &&) noexcept = default;
    Player& operator=(const Player &&) noexcept = default;
  };
```

Assignment #3 (1)

```cpp
class Player {
  string last_name_;
  vector<string> names_;
  double score_;
  Player(const string &line) { ... }
  ~Player() {
    std::cout << "Destroying " << last_name_ << std::endl;
  }
  Player(Player &&from) noexcept {
    std::cout << "move constructor of " << from.last_name_ << std::endl;
    ...
  }
};

int main(int argc, char *argv[]) {
  string file_name(argv[1]);
  vector<Player> players;
  players.reserve(5);
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    players.emplace_back(player);
  }
  // sort removed
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```

Added Move Constructor

Reserved 5 Players in vector



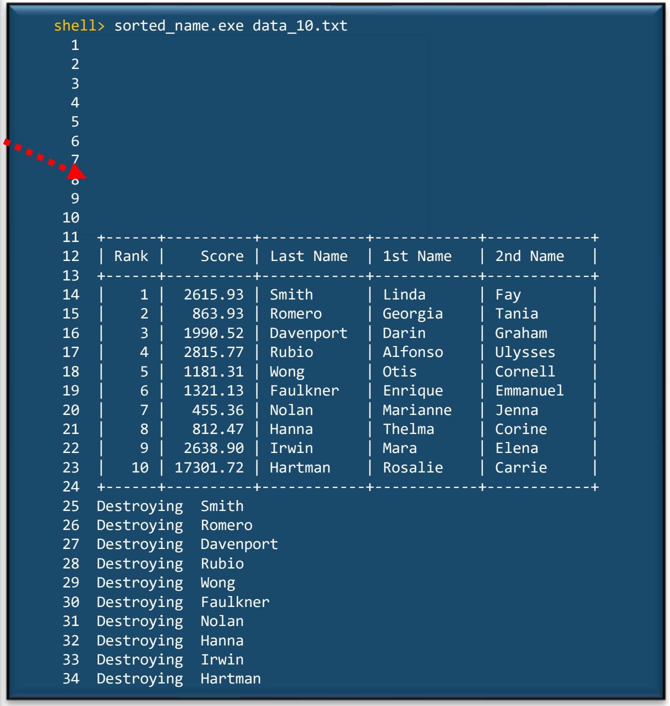
```
shell> sorted_name.exe data_10.txt










+------+-----------+-----------+-----------+-----------+
| Rank |     Score | Last Name | 1st Name  | 2nd Name  |
+------+-----------+-----------+-----------+-----------+
|    1 |   2615.93 | Smith     | Linda     | Fay       |
|    2 |    863.93 | Romero    | Georgia   | Tania     |
|    3 |   1990.52 | Davenport | Darin     | Graham    |
|    4 |   2815.77 | Rubio     | Alfonso   | Ulysses   |
|    5 |   1181.31 | Wong      | Otis      | Cornell   |
|    6 |   1321.13 | Faulkner  | Enrique   | Emmanuel  |
|    7 |    455.36 | Nolan     | Marianne  | Jenna     |
|    8 |    812.47 | Hanna     | Thelma    | Corine    |
|    9 |   2638.90 | Irwin     | Mara      | Elena     |
|   10 |  17301.72 | Hartman   | Rosalie   | Carrie    |
+------+-----------+-----------+-----------+-----------+
Destroying  Smith
Destroying  Romero
Destroying  Davenport
Destroying  Rubio
Destroying  Wong
Destroying  Faulkner
Destroying  Nolan
Destroying  Hanna
Destroying  Irwin
Destroying  Hartman
```

Move ?
Destroy ?

Assignment #3 (2)

```cpp
class Player {
  string last_name_;
  vector<string> names_;
  double score_;
  Player(const string &line) { ... }
  ~Player() {
    std::cout << "Destroying " << last_name_ << std::endl;
  }
  Player(Player &&from) noexcept {
    std::cout << "move constructor of " << from.last_name_ << std::endl;
    ...
  }
};

int main(int argc, char *argv[]) {
  string file_name(argv[1]);
  vector<Player> players;
  players.reserve(5);
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    players.emplace_back(player);
  }
  // sort removed
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```

Can you explain the Destroy?

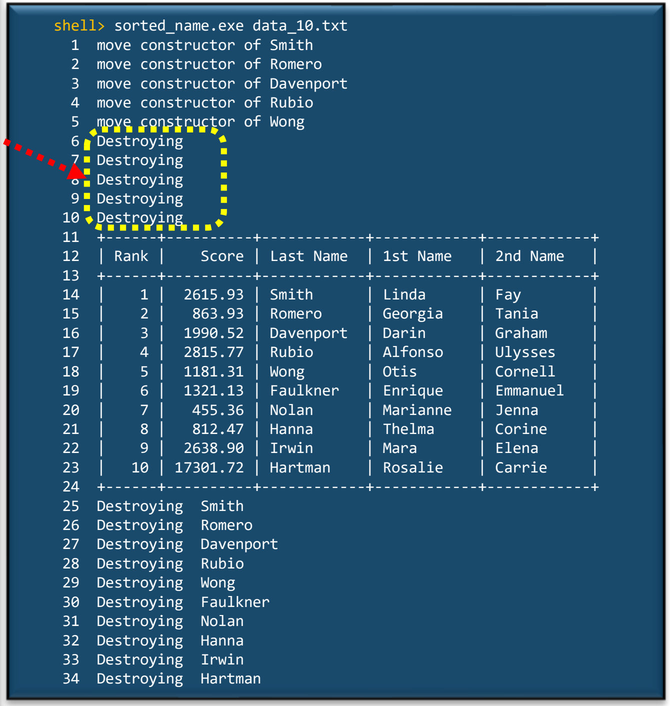


```
shell> sorted_name.exe data_10.txt










+------+-----------+-----------+-----------+-----------+
| Rank |     Score | Last Name | 1st Name  | 2nd Name  |
+------+-----------+-----------+-----------+-----------+
|    1 |   2615.93 | Smith     | Linda     | Fay       |
|    2 |    863.93 | Romero    | Georgia   | Tania     |
|    3 |   1990.52 | Davenport | Darin     | Graham    |
|    4 |   2815.77 | Rubio     | Alfonso   | Ulysses   |
|    5 |   1181.31 | Wong      | Otis      | Cornell   |
|    6 |   1321.13 | Faulkner  | Enrique   | Emmanuel  |
|    7 |    455.36 | Nolan     | Marianne  | Jenna     |
|    8 |    812.47 | Hanna     | Thelma    | Corine    |
|    9 |   2638.90 | Irwin     | Mara      | Elena     |
|   10 |  17301.72 | Hartman   | Rosalie   | Carrie    |
+------+-----------+-----------+-----------+-----------+
Destroying  Smith
Destroying  Romero
Destroying  Davenport
Destroying  Rubio
Destroying  Wong
Destroying  Faulkner
Destroying  Nolan
Destroying  Hanna
Destroying  Irwin
Destroying  Hartman
```

Assignment #3 (3)

```cpp
class Player {
  string last_name_;
  vector<string> names_;
  double score_;
  Player(const string &line) { ... }
  ~Player() {
    std::cout << "Destroying " << last_name_ << std::endl;
  }
  Player(Player &&from) noexcept {
    std::cout << "move constructor of " << from.last_name_ << std::endl;
    ...
  }
};

int main(int argc, char *argv[]) {
  string file_name(argv[1]);
  vector<Player> players;
  players.reserve(5);
  std::ifstream fin(file_name, std::ios::in);
  string line;
  while (std::getline(fin, line)) {
    players.emplace_back(player);
  }
  // sort removed
  int idx = 0;
  print_table_header();
  for (auto &player : players) {
    player.print_table_entry(++idx);
  }
  print_table_footer();
}
```

Can you explain the Destroy?



```
shell> sorted_name.exe data_10.txt
move constructor of Smith
move constructor of Romero
move constructor of Davenport
move constructor of Rubio
move constructor of Wong
Destroying
Destroying
Destroying
Destroying
Destroying
+------+-----------+------------+------------+------------+
| Rank |     Score | Last Name  | 1st Name   | 2nd Name   |
+------+-----------+------------+------------+------------+
|    1 |   2615.93 | Smith      | Linda      | Fay        |
|    2 |    863.93 | Romero     | Georgia    | Tania      |
|    3 |   1990.52 | Davenport  | Darin      | Graham     |
|    4 |   2815.77 | Rubio      | Alfonso    | Ulysses    |
|    5 |   1181.31 | Wong       | Otis       | Cornell    |
|    6 |   1321.13 | Faulkner   | Enrique    | Emmanuel   |
|    7 |    455.36 | Nolan      | Marianne   | Jenna      |
|    8 |    812.47 | Hanna      | Thelma     | Corine     |
|    9 |   2638.90 | Irwin      | Mara       | Elena      |
|   10 |  17301.72 | Hartman    | Rosalie    | Carrie     |
+------+-----------+------------+------------+------------+
Destroying  Smith
Destroying  Romero
Destroying  Davenport
Destroying  Rubio
Destroying  Wong
Destroying  Faulkner
Destroying  Nolan
Destroying  Hanna
Destroying  Irwin
Destroying  Hartman
```

Explicit Keyword [1]

```
class Q {
  int num_;
  int den_;
 public:
  Q(int num, int den) : num_{num}, den_{den} {}

  void print(const string &sep) const {
    double value = static_cast<double>(num_) / den_;
    cout << "Q = " << num_ << sep << den_ << " = " << value << std::endl;
  }
};

vector<Q> qs;

void q_factory(Q q) {
  qs.push_back(q);
}

int main() {
  int n = 22, d = 7;
  q_factory({n, d});

  for(auto &q: qs) q.print(" / ");
}
```

Note the { , } and the argument (Q q)

Compiler is allowed to make this implicit conversion

Explicit Keyword [2]

```
class Q {
  int num_;
  int den_;
 public:
  explicit Q(int num, int den) : num_{num}, den_{den} {}

  void print(const string &sep) const {
    double value = static_cast<double>(num_) / den_;
    cout << "Q = " << num_ << sep << den_ << " = " << value << std::endl;
  }
};

vector<Q> qs;

void q_factory(Q q) {
  qs.push_back(q);
}

int main() {
  int n = 22, d = 7;
  q_factory({n, d});

  for(auto &q: qs) q.print(" / ");
}
```

Error: converting to ‘Q’ from initializer list would use explicit constructor ‘Q::Q(int, int)’

q_factory({n, d});

Returned Value: How it works? (1)

```
struct Rect {
  int w_;
  int h_;

  Rect(const int w, const int h) : w_{w}, h_{h} {}

  String stringify() const {
    std::stringstream ss;
    ss << "Width  = " << w_ << ", ";
    ss << "Height = " << h_;
    String s{ss};
    return s;
  }
};

{
  Rect rect{3,4}
  String s = rect.stringify();
  cout << "Rectangle info " << s << endl;
}
```

Tricky question: the scope close must delete all stack objects, then
what about s?

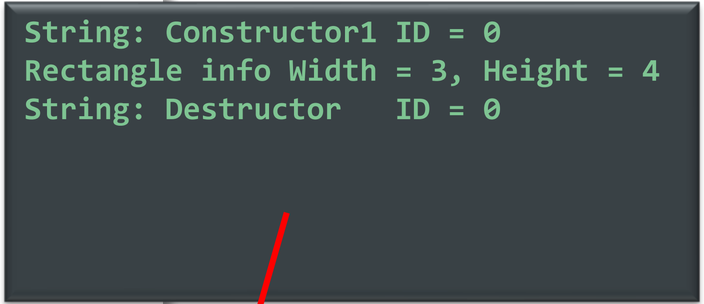
Returned Value: How it works? (2)

```
struct Rect {
  int w_;
  int h_;

  Rect(const int w, const int h) : w_{w}, h_{h} {}

  String stringify() const {
    std::stringstream ss;
    ss << "Width  = " << w_ << ", ";
    ss << "Height = " << h_;
    String s{ss};
    return s;
  }
};

{
  Rect rect{3,4}
  String s = rect.stringify();
  cout << "Rectangle info " << s << endl;
}
```

A new tmp object is created

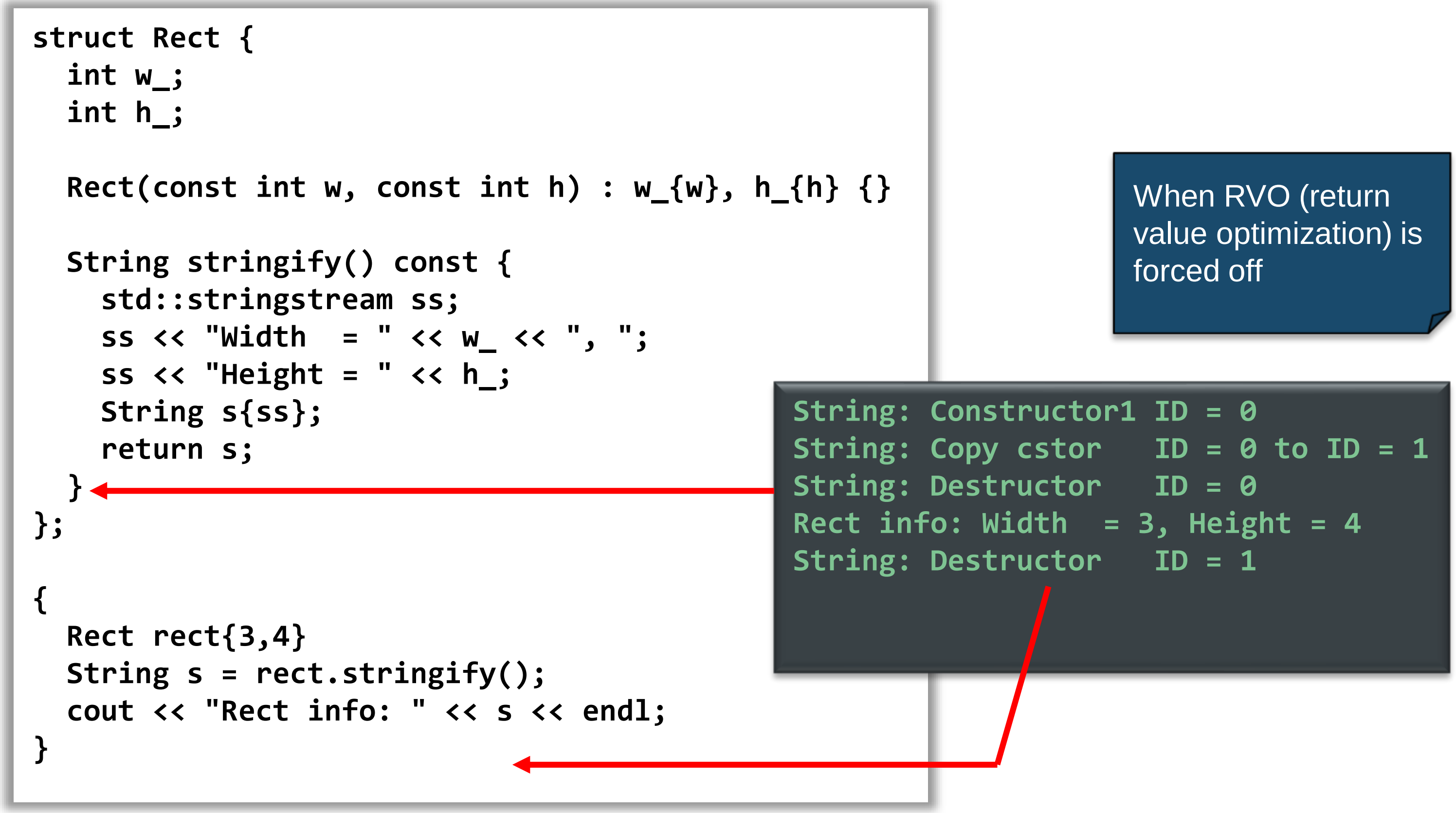
Returned Value: How it works? (3)

```
struct Rect {
  int w_;
  int h_;

  Rect(const int w, const int h) : w_{w}, h_{h} {}

  String stringify() const {
    std::stringstream ss;
    ss << "Width  = " << w_ << ", ";
    ss << "Height = " << h_;
    String s{ss};
    return s;
  }
};

{
  Rect rect{3,4}
  String s = rect.stringify();
  cout << "Rect info: " << s << endl;
}
```



```
String: Constructor1 ID = 0
Rectangle info Width = 3, Height = 4
String: Destructor   ID = 0
```

Returned Value: How it works? (4)

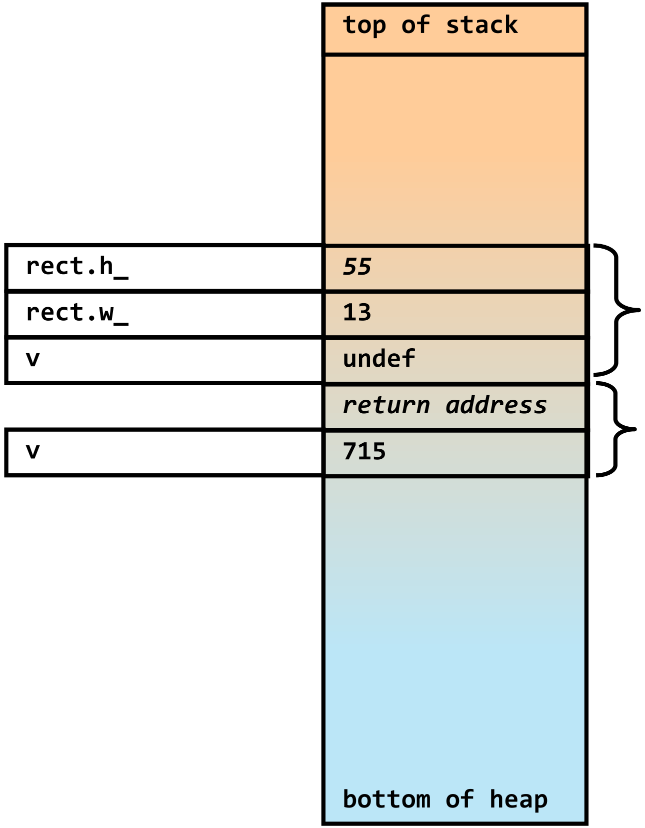


```
struct Rect {
  int w_;
  int h_;

  Rect(const int w, const int h) : w_{w}, h_{h} {}

  String stringify() const {
    std::stringstream ss;
    ss << "Width  = " << w_ << ", ";
    ss << "Height = " << h_;
    String s{ss};
    return s;
  }
};

{
  Rect rect{3,4}
  String s = rect.stringify();
  cout << "Rect info: " << s << endl;
}
```

```
String: Constructor1 ID = 0
String: Copy cstor   ID = 0 to ID = 1
String: Destructor   ID = 0
Rect info: Width  = 3, Height = 4
String: Destructor   ID = 1
```

When RVO (return value optimization) is forced off

Simple Return (1)



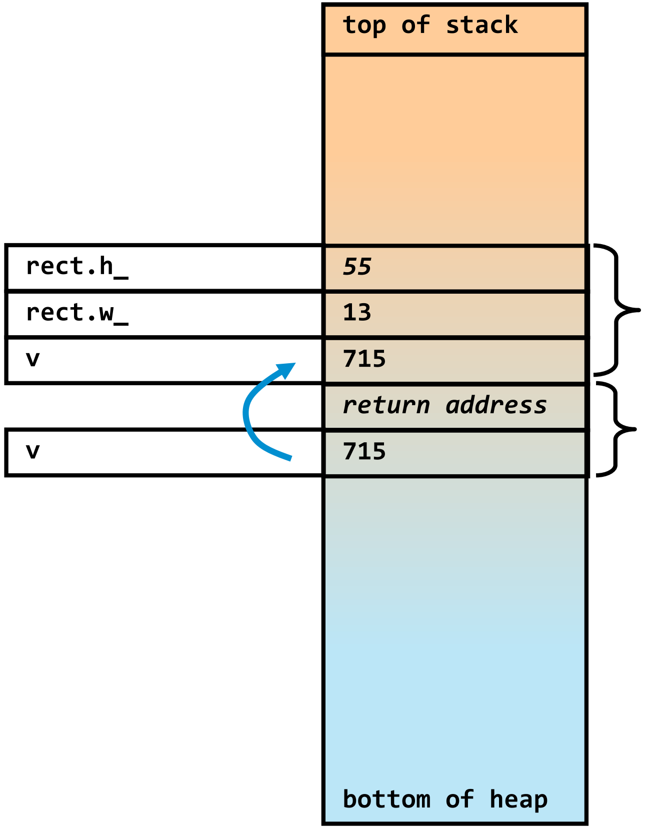
	top of stack
rect.h_	*55*
rect.w_	13
v	undef
	return address
v	715
	bottom of heap

```
class Rect{
  int w_;
  int h_;
}
```

```
int Myclass::get_area() {
  int v = w_ * h_;
  return v;
}
```

```
{
  Rect rect(13,55)
  int v = rect.get_area();
  cout << "Value = "  << v << endl;
}
```

Simple Return (2)



top of stack

rect.h_ 55

rect.w_ 13

v 715

return address

v 715

bottom of heap

```
class Rect{
  int w_;
  int h_;
}
```

```
int Myclass::get_area() {
  int v = w_ * h_;
  return v;
}
```

```
{
  Rect rect(13,55)
  int v = rect.get_area();
  cout << "Value = "  << v << endl;
}
```

Simple Return (3)

Parameter passing and return value follows strict rules called ABI.

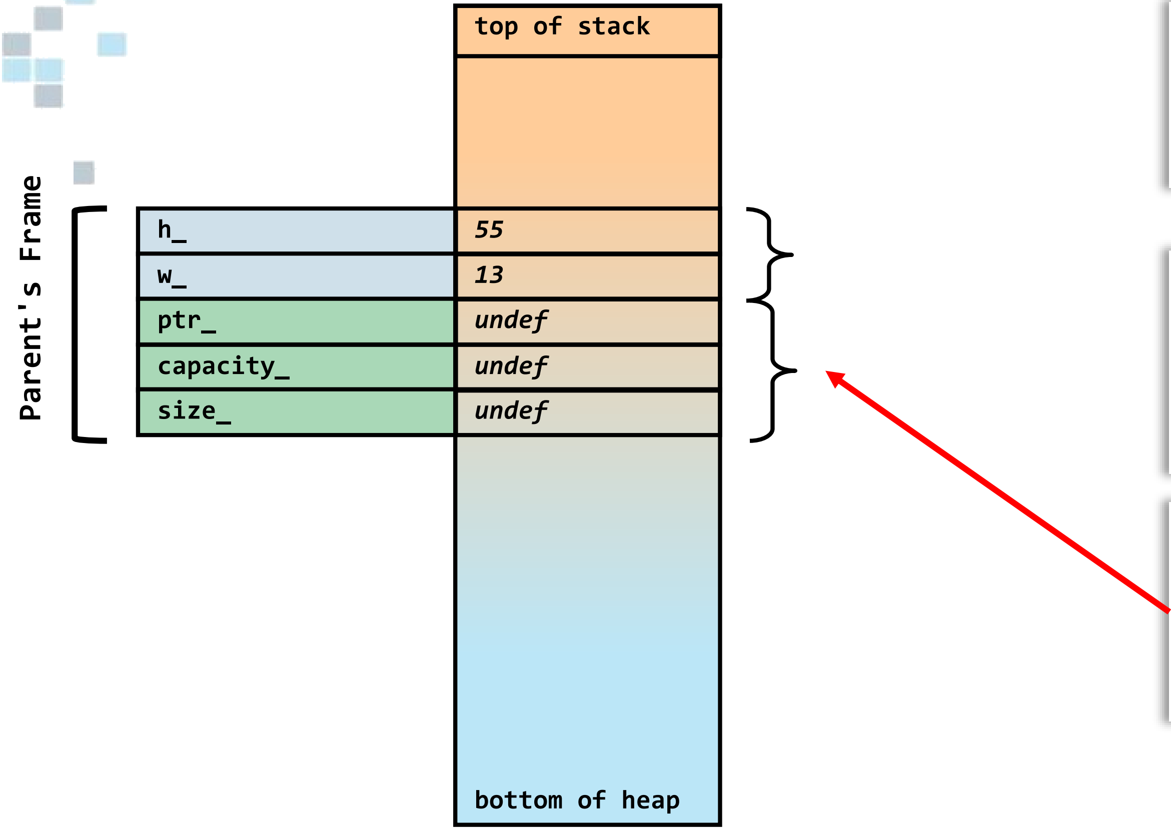
ARM spec:

1) First input parameter is *this, stored in r0.
2) The link register (lr) is the caller's return address.
3) Return value must be in r0 .

```
_get_area
  .fnstart
  @ args = 0, pretend = 0, frame = 0
  @ frame_needed = 0, uses_anonymous_args = 0
  @ link register save eliminated.
  ldr r3, [r0]        @ this_2(D)->w_, this_2(D)->w_
  ldr r0, [r0, #4]    @ this_2(D)->h_, this_2(D)->h_
  mul r0, r0, r3      @, this_2(D)->h_, this_2(D)->w_
  bx  lr   @
```

```
int Myclass::get_area() {
  int v = w_ * h_;
  return v;
}
```

Complex Return (1)

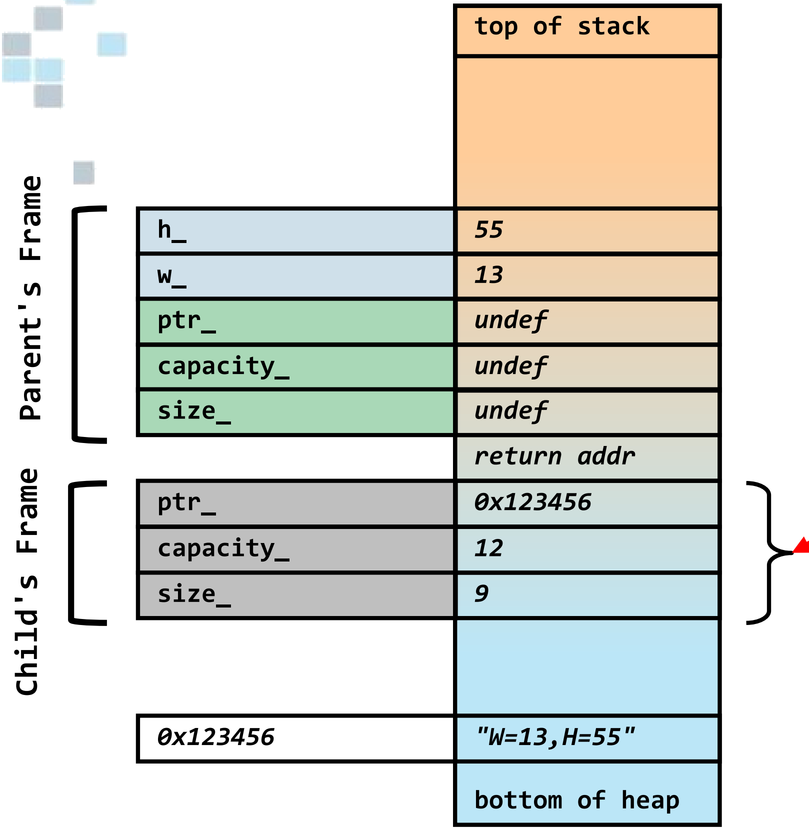


```
class Rect{
  int w_;
  int h_;
}
```

```
String Rect::stringify() {
 ...
  String s2{ss}
  return s2;
}
```

```
{
  Rect rect(13,55)
  String s1 = rect.stringigy();
  cout << "Rect :"  << s1 << endl;
}
```

Complex Return (2)



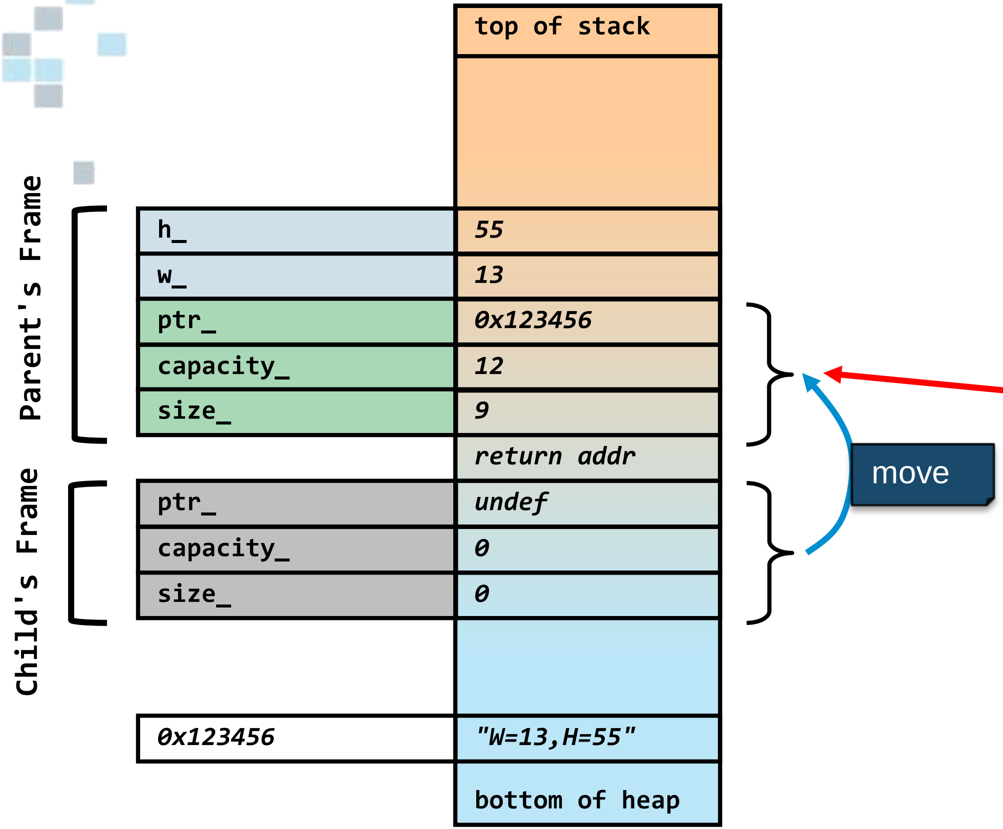
		top of stack
Parent's Frame	h_	*55*
	w_	*13*
	ptr_	*undef*
	capacity_	*undef*
	size_	*undef*
		return addr
Child's Frame	ptr_	*0x123456*
	capacity_	*12*
	size_	*9*
	0x123456	*"W=13,H=55"*
		bottom of heap

```
class Rect{
  int w_;
  int h_;
}
```

```
String Rect::stringify() {
 ...
  String s2{ss}
  return s2;
}
```

```
{
  Rect rect(13,55)
  String s1 = rect.stringigy();
  cout << "Rect :"  << s1 << endl;
}
```

Complex Return (3)



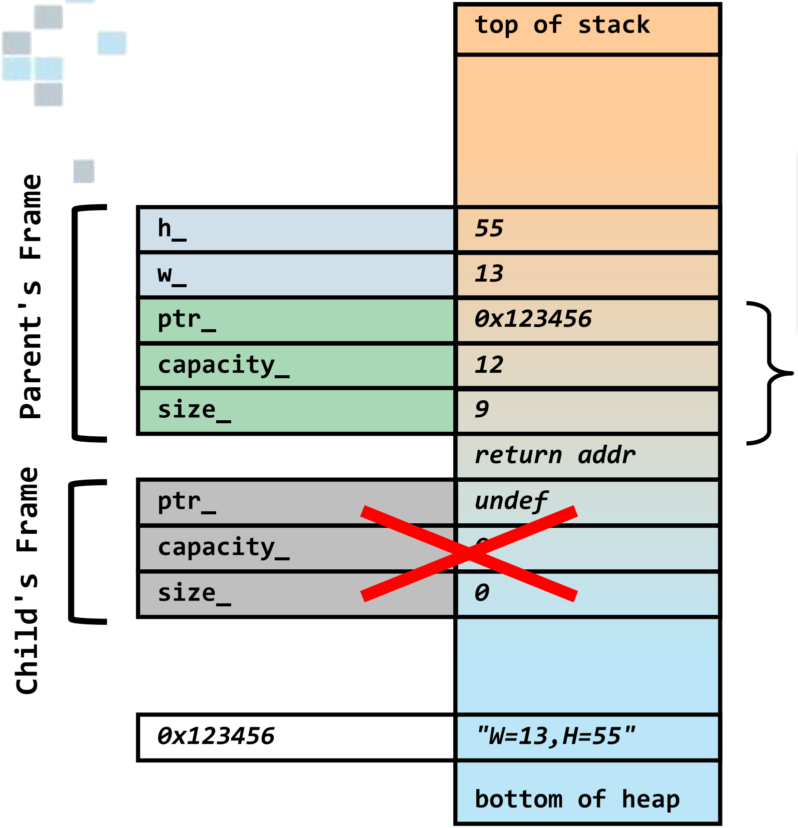
	top of stack
h_	*55*
w_	*13*
ptr_	*0x123456*
capacity_	*12*
size_	*9*
	return addr
ptr_	*undef*
capacity_	*0*
size_	*0*
0x123456	*"W=13,H=55"*
	bottom of heap

```
class Rect{
  int w_;
  int h_;
}
```

```
String Rect::stringify() {
 ...
  String s2{ss}
  return s2;
}
```

```
{
  Rect rect(13,55)
  String s1 = rect.stringigy();
  cout << "Rect :"  << s1 << endl;
}
```

Return Value Optimization (RVO)



	top of stack
h_	55
w_	13
ptr_	0x123456
capacity_	12
size_	9
	return addr
ptr_	undef
capacity_	0
size_	0
0x123456	"W=13,H=55"
	bottom of heap

Return value is constructed directly in the parent frame

```
class Rect{
  int w_;
  int h_;
}
```

```
String Rect::stringify() {
 ...
  String s2{ss}
  return s2;
}
```

```
{
  Rect rect(13,55)
  String s1 = rect.stringigy();
  cout << "Rect :"  << s1 << endl;
}
```

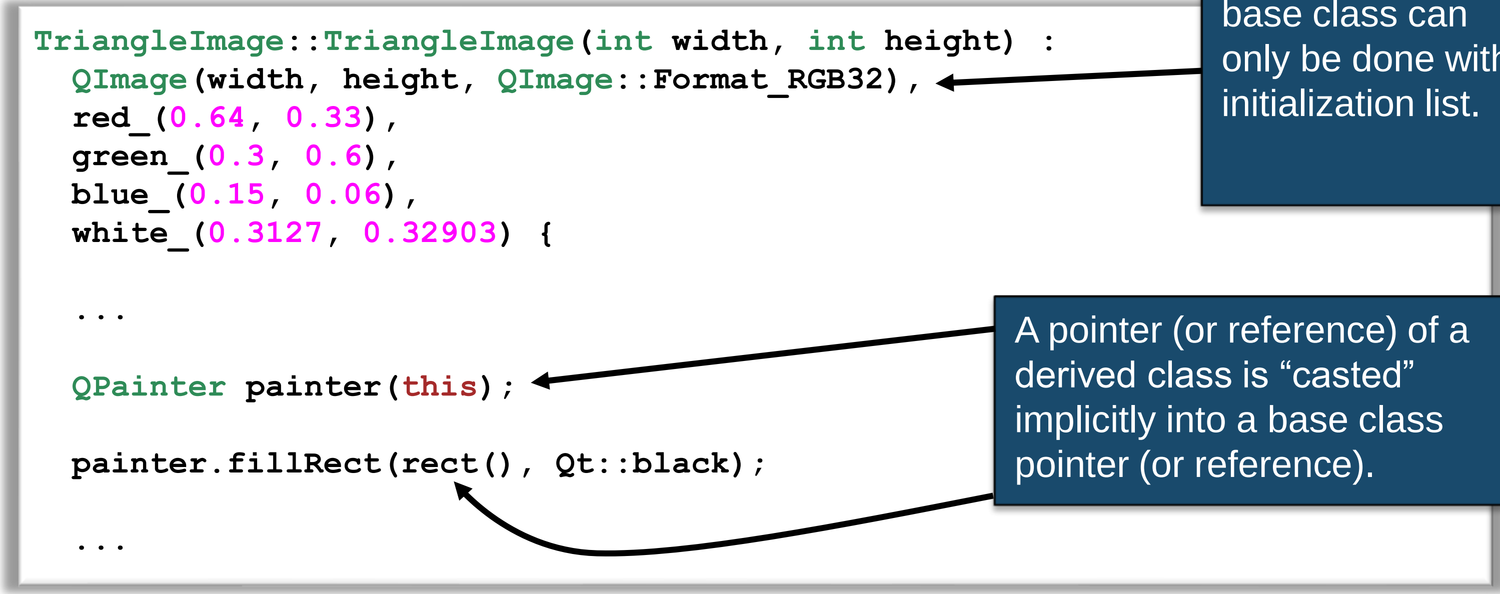
Derived Class & Inheritance

Derived Class Example (1)

```
class TriangleImage: public QImage {
public:
  TriangleImage(int width, int height);
  ~TriangleImage() = default;

private:
  QRgb xy_to_rgb(const QPointF &current);
  QPointF red_;
  QPointF green_;
  QPointF blue_;
  QPointF white_;
  QPolygonF triangle_;
  QRectF  bbox_;
};
```

TriangleImage is an extension of QImage

Derived Class Example (2.)



```
TriangleImage::TriangleImage(int width, int height) :
  QImage(width, height, QImage::Format_RGB32),
  red_(0.64, 0.33),
  green_(0.3, 0.6),
  blue_(0.15, 0.06),
  white_(0.3127, 0.32903) {

  ...

  QPainter painter(this);

  painter.fillRect(rect(), Qt::black);

  ...
```

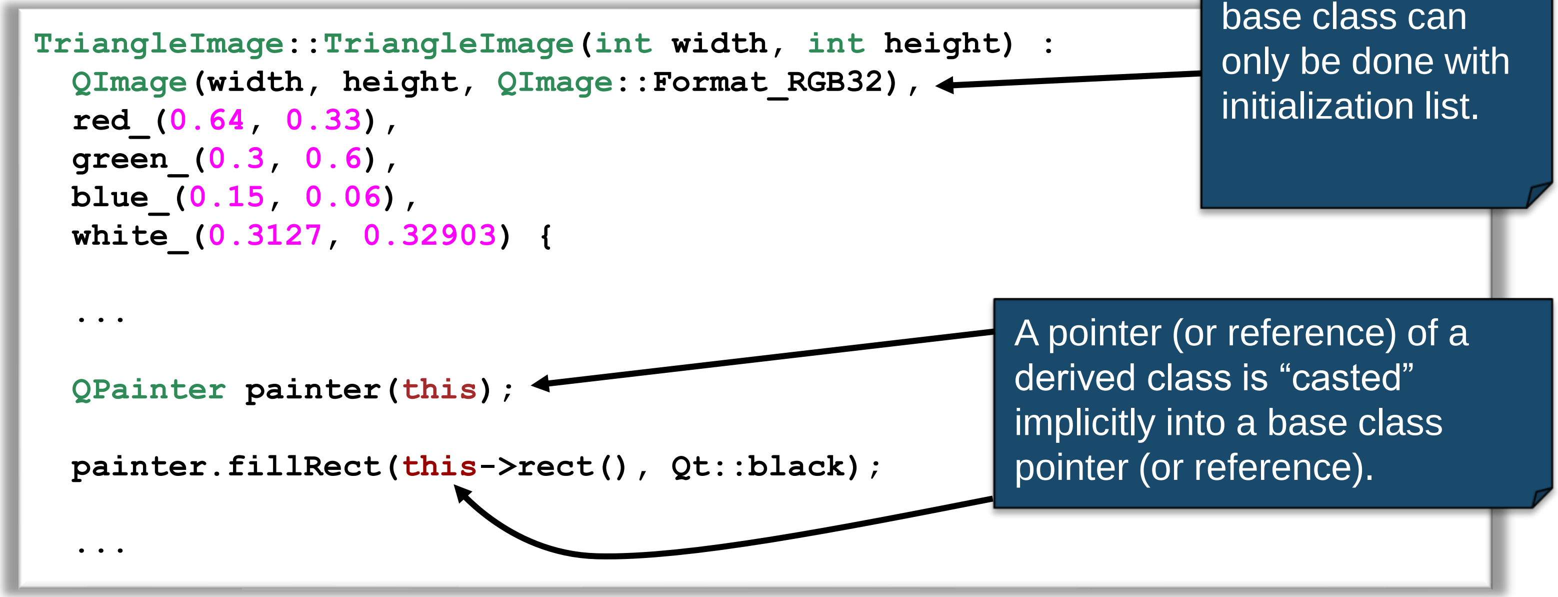
Initialization of the base class can only be done with initialization list.

A pointer (or reference) of a derived class is “casted” implicitly into a base class pointer (or reference).

Note from QT documentation
1) QPainter::QPainter(QPaintDevice *device)
2) QImage inherits QPaintDevice

Note from QT documentation
1) QRect QImage::rect() const
2) void QPainter::fillRect(const QRect &rectangle, const QColor &color)

Derived Class Example (2..)



```
TriangleImage::TriangleImage(int width, int height) :
  QImage(width, height, QImage::Format_RGB32),
  red_(0.64, 0.33),
  green_(0.3, 0.6),
  blue_(0.15, 0.06),
  white_(0.3127, 0.32903) {

  ...

  QPainter painter(this);

  painter.fillRect(this->rect(), Qt::black);

  ...
```

Initialization of the base class can only be done with initialization list.

A pointer (or reference) of a derived class is “casted” implicitly into a base class pointer (or reference).

Note from QT documentation
1) QPainter::QPainter(QPaintDevice * device)
2) QImage inherits QPaintDevice

Note from QT documentation
1) QRect QImage::rect() const
2) void QPainter::fillRect(const QRect &rectangle, const QColor &color)

Derived Class Example (3)

```
{
  QImage triangle_image(triangle_width, triangle_height, QImage::Format_RGB32);
  QPainter painter(&triangle_image);
  painter.fillRect(triangle_image.rect(), Qt::black);
  for (int y = 0; y < triangle_image.height(); ++y) {
    …
     triangle_image.setPixel(x, y, qRgb(255, 255, 255));
  }
  image_widget_->setPixmap(QPixmap::fromImage(triangle_image));
}
```

```
{
  TriangleImage triangle_image(triangle_width, triangle_height);
  image_widget_->setPixmap(QPixmap::fromImage(triangle_image));
}
```

call with `TriangleImage` is perfectly legal, no need for a special cast

Inheritance & Polymorphism

- Inheritance allows a derived class to “inherit” (public and protected) members and methods of the base class.
- Polymorphism allows processing a derived class objects as if they were object of the base class.

```
class TriangleImage : public QImage {
...
};
```

```
QPainter painter(this);
```

- Inheritance & Polymorphism are the pillars of object oriented programming

Derived Class Summary

- Key Points
  - Derived class must follow the “is a” rule.
  - Use public derivation 99% of the time
  - Member access rule in public deviation

private	can’t be accessed
protected	protected
public	public

```
class TriangleImage : public QImage {
...
};
```

```
QPainter painter(this);
```

- Key Benefits
  - Increase reuse
  - Allow extension without changing existing code

Private vs. Protected (1)

```
class Shape {
 private:
  int color_;
 public:
  Shape() {
  }
  int get_color() const { return color_;}
  void draw() const {
    std::cout << "draw a shape" << std::endl;
  }
};

class Circle : public Shape {
 private:
  Point center_;
  int radius_;
 public:
  Circle(int x, int y, int radius) :
    center_(x, y) {
   radius_ = radius;
  }
  void set_color(int color) {
    color_ = color;
};
```

```
shell> g++ ....
shapes.cpp: In member function ‘void Circle::set_color(int)’:
shapes.cpp:20:7: error: ‘int Shape::color_’ is private
   int color_;
       ^
shapes.cpp:42:5: error: within this context
     color_ = color;
     ^
```

Private vs. Protected (2)

```
class Shape {
 protected:
  int color_;
 public:
  Shape() {
  }
  int get_color() const { return color_;}
  void draw() const {
    std::cout << "draw a shape" << std::endl;
  }
};

class Circle : public Shape {
 private:
  Point center_;
  int radius_;
 public:
  Circle(int x, int y, int radius) :
    center_(x, y) {
   radius_ = radius;
  }
  void set_color(int color) {
    color_ = color;
};
```

```
int main() {
  Circle c0(0, 0, 10);
  c0.color_ = 0xff;
}
```

```
shell> g++ ....
shapes.cpp: In function ‘int main()’:
shapes.cpp:20:7: error: ‘int Shape::color_’ is protected
   int color_;
       ^
shapes.cpp:64:6: error: within this context
   c0.color_ = 0xff;
      ^
```

022

Virtual Functions

Virtual Functions: Introduction (1)

```
class Shape {
 private:
  int color_;
 public:
  Shape() {
  }
  int get_color() const { return color_;}
  void draw() const {
    std::cout << "draw a shape" << std::endl;
  }
};

class Circle : public Shape {
 private:
  Point center_;
  int radius_;
 public:
  Circle(int x, int y, int radius) :
    center_(x, y) {
   radius_ = radius;
  }
};
```

```
shell> ./shapes.exe
draw a shape
draw a shape
draw a shape
```

shapes: vector of polymorphic pointers to Shape *

```
class Triangle : public Shape {
 private:
  Point p0_;
  Point p1_;
  Point p2_;
 public:
  Triangle(int x0, int y0, int x1,
           int y1, int x2, int y2) :
    p0_{x0, y0}, p1_{x1, y1}, p2_{x2, y2} {}
};

int main() {
  std::vector<Shape *> shapes;

  shapes.push_back(new Circle(0, 0, 10));
  shapes.push_back(new Circle(5, 5,  6));
  shapes.push_back(new Triangle(0, 0, 0, 1, 2, 1));

  for(auto &shape_ptr : shapes) {
    shape_ptr->draw();
  }
}
```

Virtual Functions: Introduction (2)

```
class Shape {
 private:
  int color_;
 public:
  Shape() {
  }
  int get_color() const { return color_;}
  virtual void draw() const = 0;


};
```

```
shell> ./shapes.exe
draw a circle
draw a circle
draw a triangle
```

pure virtual function

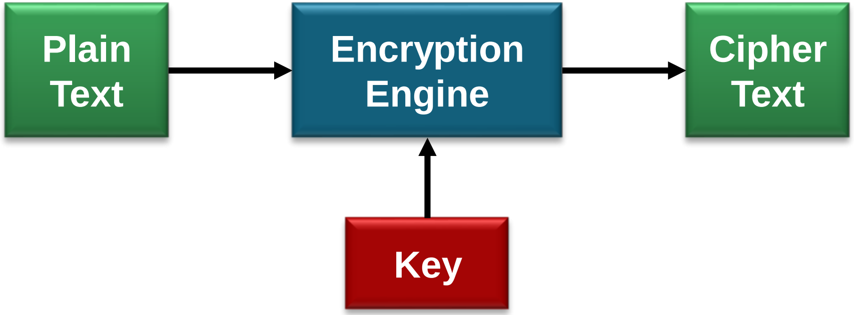
```
class Circle : public Shape {
 private:
  Point center_;
  int radius_;
 public:
  Circle(int x, int y, int radius) :
    center_(x, y) {
   radius_ = radius;
  }
  void draw() const override {
    std::cout << "draw a circle" << std::endl;
  }
};
```

virtual override function

```
class Triangle : public Shape {
 private:
  Point p0_;
  Point p1_;
  Point p2_;
 public:
  Triangle(int x0, int y0, int x1,
           int y1, int x2, int y2) :
    p0_{x0, y0}, p1_{x1, y1}, p2_{x2, y2} {}
  void draw() const override {
    std::cout << "draw a triangle" << std::endl;
  }
};

int main() {
  std::vector<Shape *> shapes;

  shapes.push_back(new Circle(0, 0, 10));
  shapes.push_back(new Circle(5, 5,  6));
  shapes.push_back(new Triangle(0, 0, 0, 1, 2, 1));

  for(auto &shape_ptr : shapes) {
    shape_ptr->draw();
  }
}
```

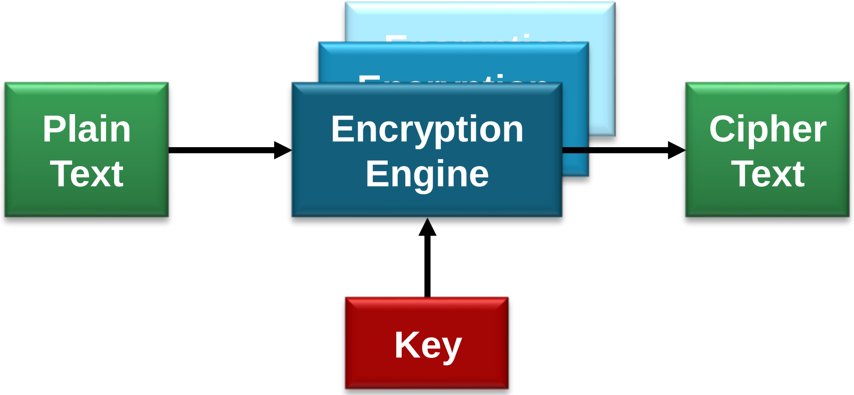
Virtual Functions (1)

- Typical scenario: add a new way of processing your data
  - Different algorithm
  - Optimized algorithm
- Example: encryption engine



Virtual Functions (2)

- Target: add encryption algorithms with minimal changes on the code base



Virtual Function Example (1)

```
class EncryptEngine {
private:
  EncryptDES encrypter_;
public:
  void encrypt(const string &msg) {
    encrypter_->encrypt(msg);
  }
  void decrypt(const string &cipher) {
    encrypter_->decrypt(cipher);
  }
  void crypt_block(...);
  void crypt_binary(...);
  void crypt_short_msg(...);
  void change_key(...);
  void set_encrypter(...);
  void erase_key(...);
};
```

```
class EncryptDES {
private:
  int key_;
public:
  void encrypt(const string &msg) {
     cout << “DES crypt " << msg << endl;
  }
  void decrypt(const string &cipher) {
     cout << “DES decrypt " << cipher << endl;
  }

};
```

How can I add a new encryption algorithm with limited changes on the code base?

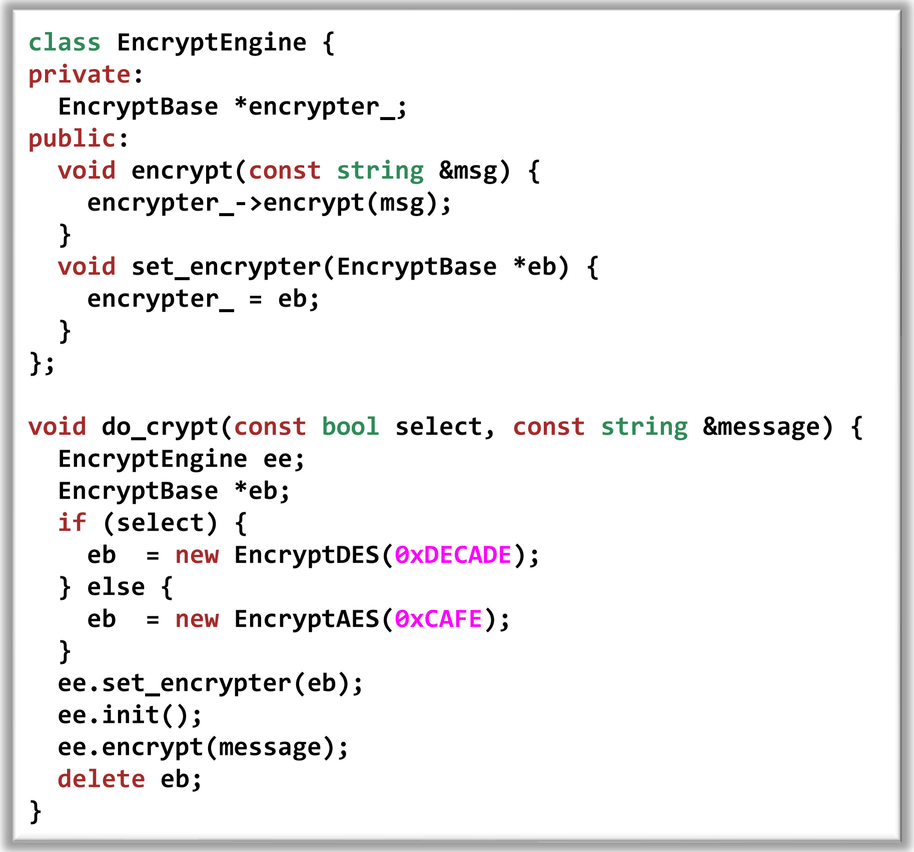
Virtual Function Example (2)

```
class EncryptEngine {
private:
  EncryptDES *encrypter1_;
  EncryptAES *encrypter2_;
  int selector_;
public:
  void encrypt(const string &msg) {
    switch(selector_) {
    case 1:
      encrypter1_->encrypt(msg);
      break;
    case 2:
      encrypter2_->encrypt(msg);
      break;
    }
  }
};
```

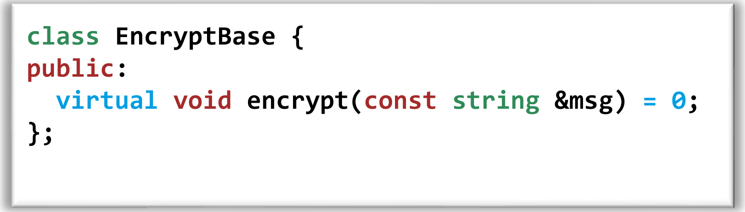
```
class EncryptDES {
private:
  int key_;
public:
  void encrypt(const string &msg) {
     cout << “DES crypt " << msg << endl;
  }
};
```

```
class EncryptAES {
private:
  int key_;
public:
  void encrypt(const string &msg) {
     cout << “AES crypt " << msg << endl;
  }
};
```

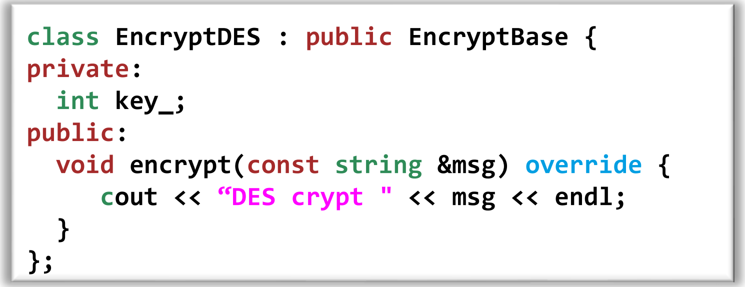
Virtual Function Example (3)



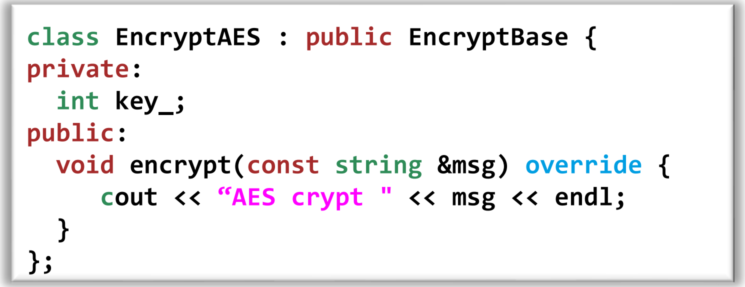
```
class EncryptEngine {
private:
  EncryptBase *encrypter_;
public:
  void encrypt(const string &msg) {
    encrypter_->encrypt(msg);
  }
  void set_encrypter(EncryptBase *eb) {
    encrypter_ = eb;
  }
};

void do_crypt(const bool select, const string &message) {
  EncryptEngine ee;
  EncryptBase *eb;
  if (select) {
    eb  = new EncryptDES(0xDECADE);
  } else {
    eb  = new EncryptAES(0xCAFE);
  }
  ee.set_encrypter(eb);
  ee.init();
  ee.encrypt(message);
  delete eb;
}
```



```
class EncryptBase {
public:
  virtual void encrypt(const string &msg) = 0;
};
```



```
class EncryptDES : public EncryptBase {
private:
  int key_;
public:
  void encrypt(const string &msg) override {
     cout << “DES crypt " << msg << endl;
  }
};
```



```
class EncryptAES : public EncryptBase {
private:
  int key_;
public:
  void encrypt(const string &msg) override {
     cout << “AES crypt " << msg << endl;
  }
};
```

Virtual Function Example (4a)

```
class EncryptEngine {
private:
  EncryptBase *encrypter_;
public:
  void encrypt(const string &msg) {
    encrypter_->encrypt(msg);
  }
  void set_encrypter(EncryptBase *eb) {
    encrypter_ = eb;
  }
};

void do_crypt(const bool select, const string &message) {
  EncryptEngine ee;
  EncryptBase *eb;
  if (select) {
    eb  = new EncryptDES(0xDECADE);
  } else {
    eb  = new EncryptAES(0xCAFE);
  }
  ee.set_encrypter(eb);
  ee.init();
  ee.encrypt(message);
  delete eb;
}
```

pointer upcasting is legal, no need for a special cast

```
class EncryptBase {
public:
  virtual void encrypt(const string &msg) = 0;
};
```

```
class EncryptDES : public EncryptBase {
private:
  int key_;
public:
  void encrypt(const string &msg) override {
     cout << “DES crypt " << msg << endl;
  }
};
```

```
class EncryptAES : public EncryptBase {
private:
  int key_;
public:
  void encrypt(const string &msg) override {
     cout << “AES crypt " << msg << endl;
  }
};
```

Virtual Function Example (4b)

```
class EncryptEngine {
private:
  EncryptBase *encrypter_;
public:
  void encrypt(const string &msg) {
    encrypter_->encrypt(msg);
  }
  void set_encrypter(EncryptBase *eb) {
    encrypter_ = eb;
  }
};

void do_crypt(const boo                              ) {
  EncryptEngine ee;
  EncryptBase *eb;
  if (select) {
    eb  = new EncryptDES(0xDECADE);
  } else {
    eb  = new EncryptAES(0xCAFE);
  }
  ee.set_encrypter(eb);
  ee.init();
  ee.encrypt(message);
  delete eb;
}
```

encrypter->encrypt() automatically dispatch to DES::encrypt() or AES::encrypt()

```
class EncryptBase {
public:
  virtual void encrypt(const string &msg) = 0;
};
```

```
class EncryptDES : public EncryptBase {
private:
  int key_;
public:
  void encrypt(const string &msg) override {
     cout << “DES crypt " << msg << endl;
  }
};
```

```
class EncryptAES : public EncryptBase {
private:
  int key_;
public:
  void encrypt(const string &msg) override {
     cout << “AES crypt " << msg << endl;
  }
};
```

Virtual Function Example (4c)

```
class EncryptEngine {
private:
  EncryptBase *encrypter_;
public:
  void encrypt(const string &msg) {
    encrypter_->encrypt(msg);
  }
  void set_encrypter(EncryptBase *eb)
    encrypter_ = eb;
  }
};

void do_crypt(const bool select, const string &message) {
  EncryptEngine ee;
  EncryptBase *eb;
  if (select) {
    eb  = new EncryptDES(0xDECADE);
  } else {
    eb  = new EncryptAES(0xCAFE);
  }
  ee.set_encrypter(eb);
  ee.init();
  ee.encrypt(message);
  delete eb;
}
```

`= 0` indicates pure virtual function, i.e. no implementation in the base class

When a class has at least one pure virtual function, it is called an abstract class.

```
class EncryptBase {
public:
  virtual void encrypt(const string &msg) = 0;
};
```

```
class EncryptDES : public EncryptBase {
private:
  int key_;
public:
  void encrypt(const string &msg) override {
     cout << “DES crypt " << msg << endl;
  }
};
```

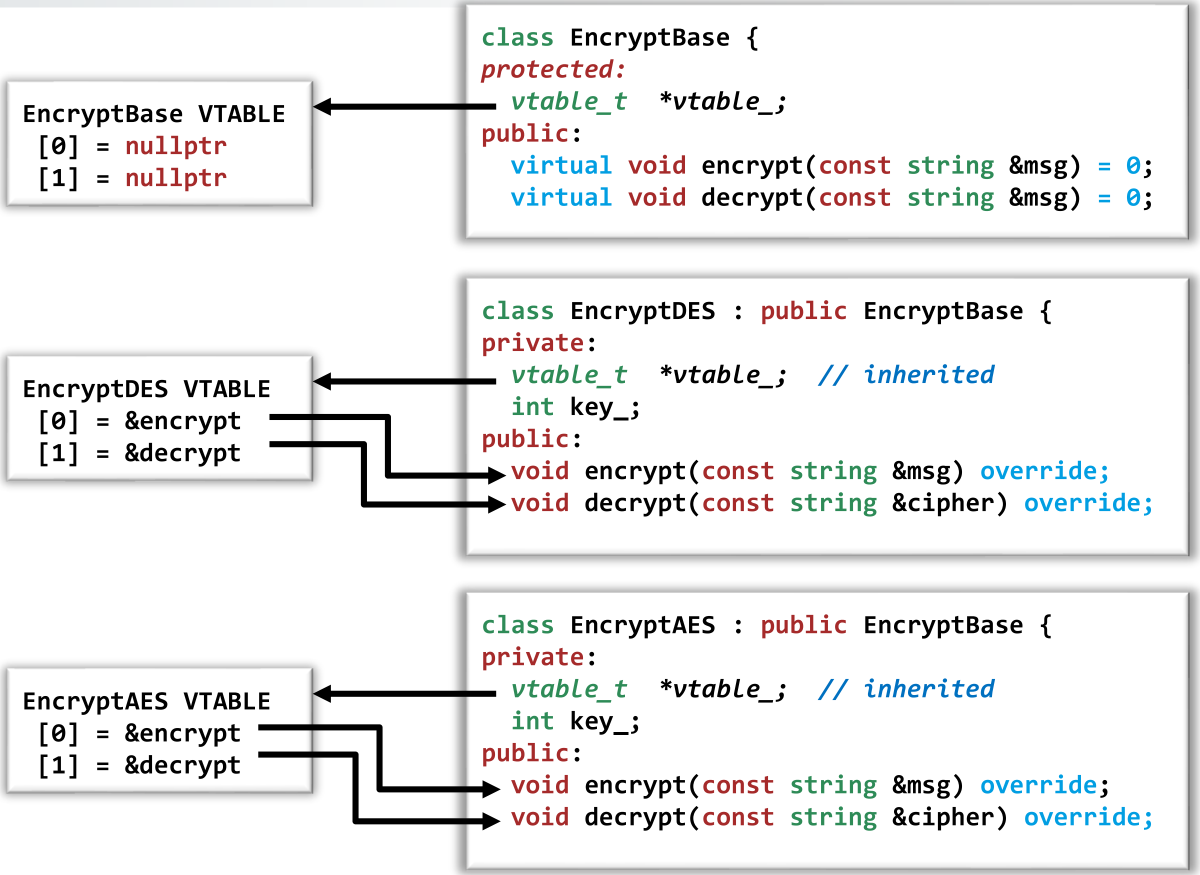
```
class EncryptAES : public EncryptBase {
private:
  int key_;
public:
  void encrypt(const string &msg) override {
     cout << “AES crypt " << msg << endl;
  }
};
```

Virtual Function: How ? (1)

```
using vtable_t =
  void (*)(const string &);
```

One Virtual Table per class

One extra pointer for each instance of the class



```
EncryptBase VTABLE
 [0] = nullptr
 [1] = nullptr
```

```
class EncryptBase {
protected:
  vtable_t  *vtable_;
public:
  virtual void encrypt(const string &msg) = 0;
  virtual void decrypt(const string &msg) = 0;
```

```
EncryptDES VTABLE
 [0] = &encrypt
 [1] = &decrypt
```

```
class EncryptDES : public EncryptBase {
private:
  vtable_t  *vtable_;  // inherited
  int key_;
public:
  void encrypt(const string &msg) override;
  void decrypt(const string &cipher) override;
```

```
EncryptAES VTABLE
 [0] = &encrypt
 [1] = &decrypt
```

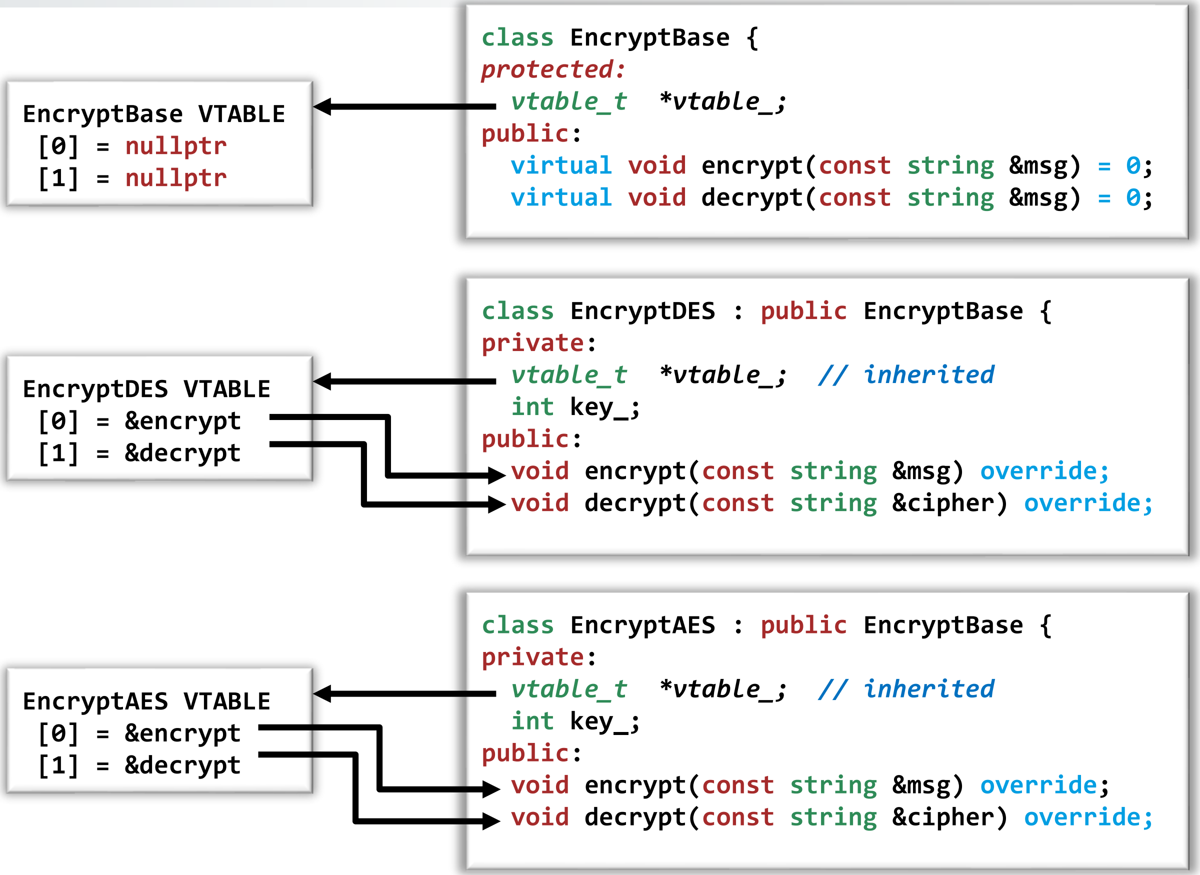
```
class EncryptAES : public EncryptBase {
private:
  vtable_t  *vtable_;  // inherited
  int key_;
public:
  void encrypt(const string &msg) override;
  void decrypt(const string &cipher) override;
```

Virtual Function: How ? (2)

```
using vtable_t =
  void (*)(const string &);
```

```
EncryptBase *eb;
eb  = new EncryptDES(0xDECADE);
eb->encrypt(msg)
```

```
EncryptBase *eb;
eb  = new EncryptDES(0xDECADE);
(eb->vtable_[0])(msg)
```



```
EncryptBase VTABLE
 [0] = nullptr
 [1] = nullptr
```

```
class EncryptBase {
protected:
  vtable_t   *vtable_;
public:
  virtual void encrypt(const string &msg) = 0;
  virtual void decrypt(const string &msg) = 0;
```

```
EncryptDES VTABLE
 [0] = &encrypt
 [1] = &decrypt
```

```
class EncryptDES : public EncryptBase {
private:
  vtable_t   *vtable_;  // inherited
  int key_;
public:
  void encrypt(const string &msg) override;
  void decrypt(const string &cipher) override;
```

```
EncryptAES VTABLE
 [0] = &encrypt
 [1] = &decrypt
```

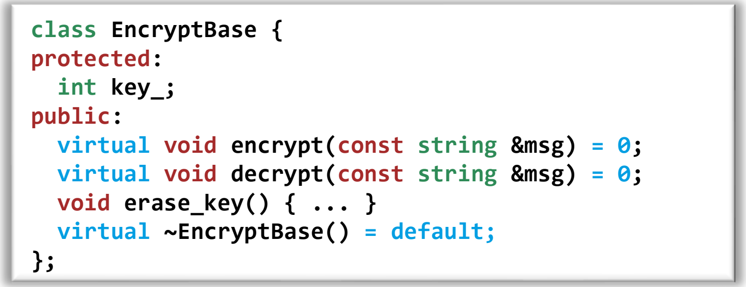
```
class EncryptAES : public EncryptBase {
private:
  vtable_t   *vtable_;  // inherited
  int key_;
public:
  void encrypt(const string &msg) override;
  void decrypt(const string &cipher) override;
```

Virtual Function: Addendum (1)

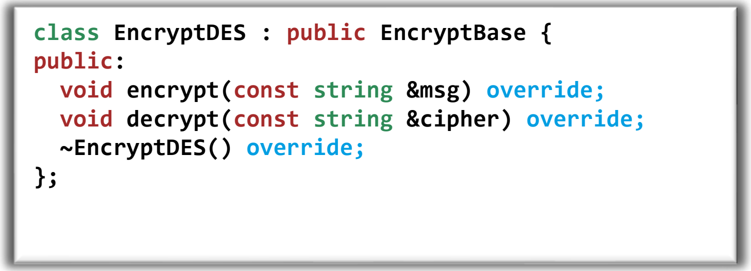
- Only virtual functions appear in the vtables and therefore we be subject to dynamic dispatch.
- When a function is declared virtual, it is implicitly virtual for all identically function appearing in derived class.
  - Recommendation: use one of `virtual`, `override` or `final` keywords for clarity
- Destructor of the base class must be virtual
  - Otherwise, you may have potential memory leakage! Why?

    => only the base class storage will be released.

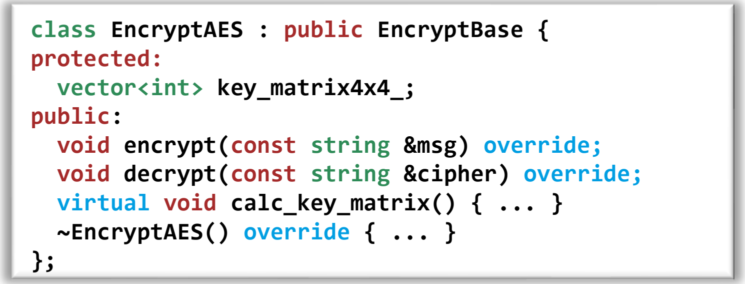
Virtual Function: Addendum (2)



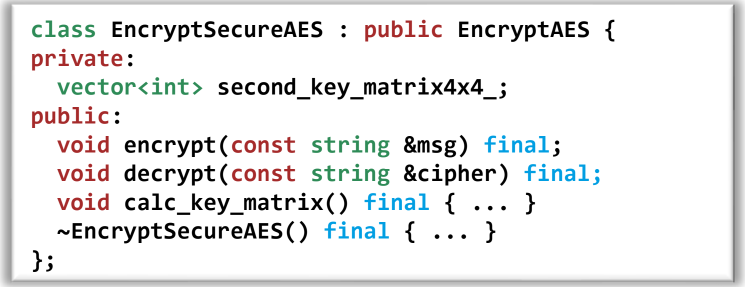
```
class EncryptBase {
protected:
  int key_;
public:
  virtual void encrypt(const string &msg) = 0;
  virtual void decrypt(const string &msg) = 0;
  void erase_key() { ... }
  virtual ~EncryptBase() = default;
};
```



```
class EncryptDES : public EncryptBase {
public:
  void encrypt(const string &msg) override;
  void decrypt(const string &cipher) override;
  ~EncryptDES() override;
};
```



```
class EncryptAES : public EncryptBase {
protected:
  vector<int> key_matrix4x4_;
public:
  void encrypt(const string &msg) override;
  void decrypt(const string &cipher) override;
  virtual void calc_key_matrix() { ... }
  ~EncryptAES() override { ... }
};
```



```
class EncryptSecureAES : public EncryptAES {
private:
  vector<int> second_key_matrix4x4_;
public:
  void encrypt(const string &msg) final;
  void decrypt(const string &cipher) final;
  void calc_key_matrix() final { ... }
  ~EncryptSecureAES() final { ... }
};
```