Inheritance

In Object Oriented Programming,
we know that the user of a class can use the functionalities of the class
without knowing / bothering about the implementation.

OOP also provides a mechanism so that,
user can create his own class which **inherits** functionalities of a pre-existing class
without knowing / bothering about the implementation of the original class.

This is known as **inheritance.**
The pre-existing class is called **base-class**.
The newly created class is called the **derived class**.

Suppose we already have a class B with 20 public member functions.

class B is defined inside header B.h

B.h

```
class B
{
   public:
   void func1(int);
   int func2(void);
   ...
   ...
   void func20(void);
   private:
   ...
};
```

B.h gives only the declarations of member functions

We do not know anything about the implementation of the member functions `func1() ... func20()`

We can still use the class by <u>including the header</u> and <u>creating object</u>.

TestB.cpp

```
#include "B.h"
int main()
{
   B t;
   t.func1(5);
   t.func2();
   ...
   t.func20();
};
```

Suppose we already have a class B with 20 public member functions.

class B is defined inside header B.h

B.h gives only the declarations of member functions

We do not know anything about the implementation of the member functions `func1() ... func20()`

B.h

```
class B
{
   public:
   void func1(int);
   int func2(void);
   ...
   ...
   void func20(void);
   private:
   ...
};
```

We can still use the class by including the header and creating object.

TestB.cpp

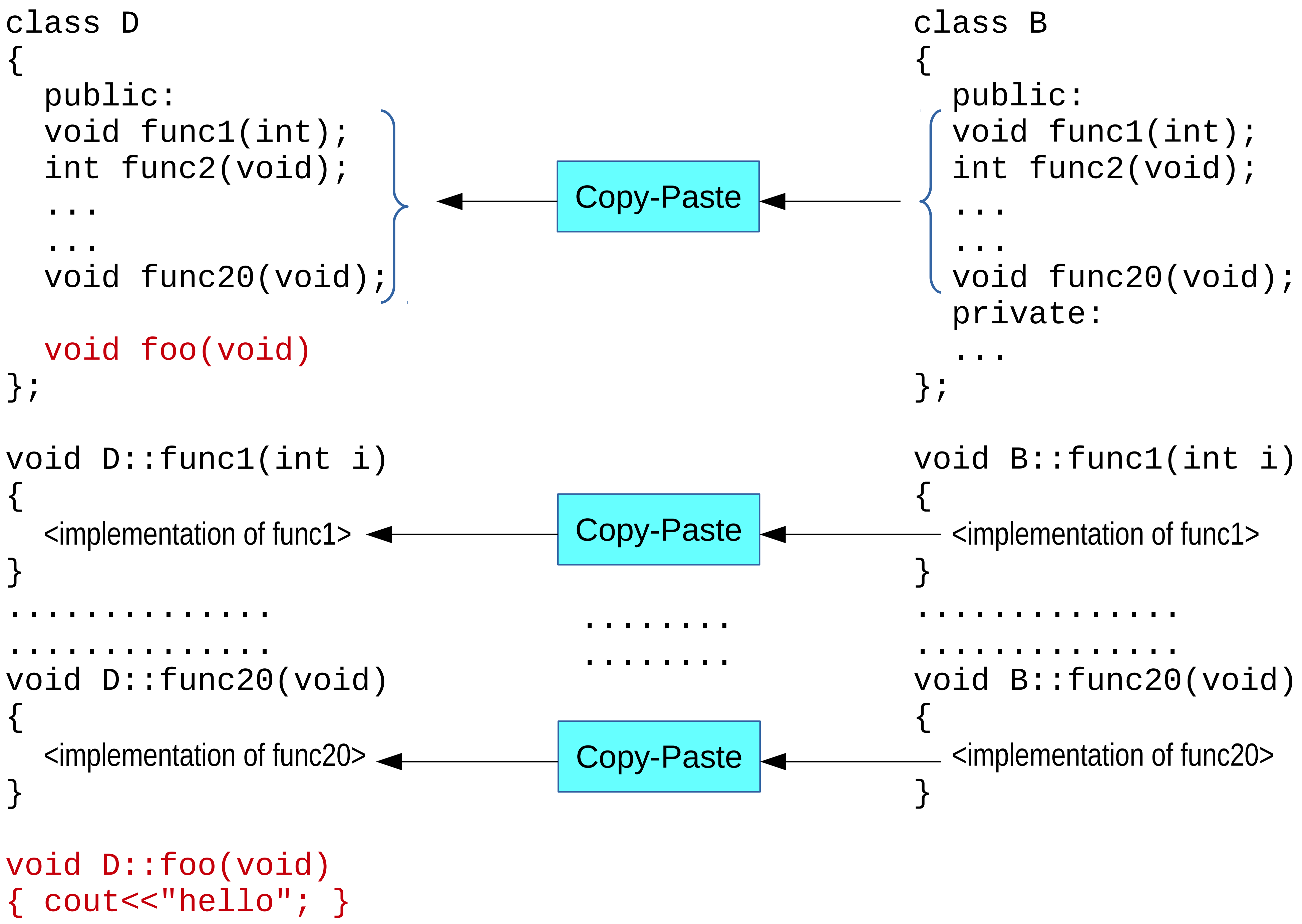
```
#include "B.h"
int main()
{
   B t;
   t.func1(5);
   t.func2();
   ...
   t.func20();
};
```

Now we want to create a new class D s.t.

- D also has the member functions func1()...func20()
- Additionally, D has another member function foo() (we shall define foo() ourselves)
- We shall use class D by creating objects and calling the member functions

```
<definition of class D>
<definition of D::foo() >
int main()
{
   D x;
   x.func1(5); x.func2();
   ...
   x.foo();
};
```

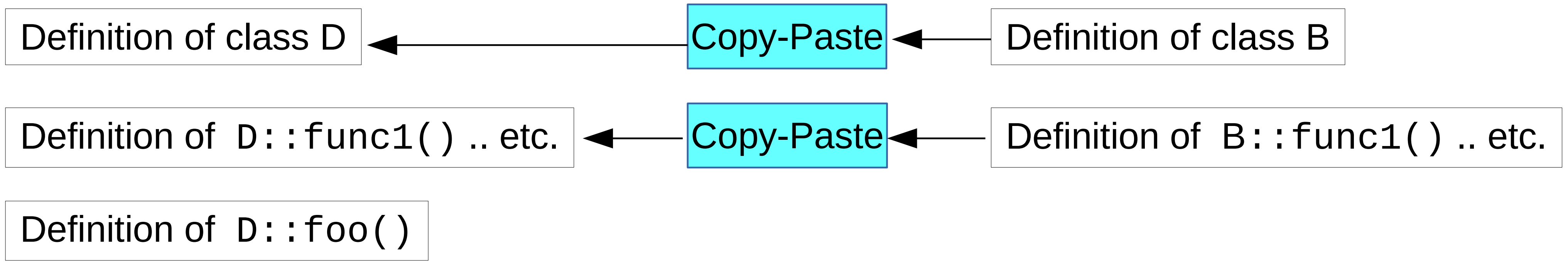
A crude solution (DO NOT do this)



```
class D
{
  public:
  void func1(int);
  int func2(void);
  ...
  ...
  void func20(void);

  void foo(void)
};

void D::func1(int i)
{
  <implementation of func1>
}
..............
..............
void D::func20(void)
{
  <implementation of func20>
}

void D::foo(void)
{ cout<<"hello"; }
```

```
........
........
```

```
class B
{
  public:
  void func1(int);
  int func2(void);
  ...
  ...
  void func20(void);
  private:
  ...
};

void B::func1(int i)
{
  <implementation of func1>
}
..............
..............
void B::func20(void)
{
  <implementation of func20>
}
```



Problem 1:

Source code for B::func1() etc. may not be available.
It could be proprietary and may come as a precompiled binary file.
Developer of class B may not allow you to know the actual implementation.

Even when you have access to the code,

Problem 2:

Either you need to understand how those functions work,
or you have a chunk of code in your own program and you have no idea what it is doing.

A better solution: Derive a class D from the base class B and inherit its members.

class B is defined
inside header B.h

B.h

```
class B
{
  public:
  void func1(int);
  int func2(void);
  ...
  ...
  void func20(void);
  private:
  ...
};
```

We may define our class D
as follows

D.h

```
#include "B.h"
class D : public B
{
  public:
  void foo(void);
};
```

We need to define our
own member foo()

D.cpp

```
void B::foo(void)
{ cout<<"hello"; }
```

Finally we can use
class D as intended

```
#include "D.h"
int main()
{
  D x;
  x.func1(5);
  x.func2();
  ...
  x.foo();
};
```

We do not need to know anything about the implementation of the base class

class B is defined
inside header B.h

B.h

```
class B
{
  public:
  void func1(int);
  int func2(void);
  ...
  ...
  void func20(void);
  private:
  ...
};
```

We may define our class D
as follows

D.h

```
#include "B.h"
class D : public B
{
  public:
  void foo(void);
};
```

We need to define our
own member foo()

D.cpp

```
void B::foo(void)
{ cout<<"hello"; }
```

Finally we can use
class D as intended

```
#include "D.h"
int main()
{
  D x;
  x.func1(5);
  x.func2();
  ...
  x.foo();
};
```

Syntax for
public inheritance

Public members (both data and function) of B automatically becomes public members of D

Private members of B are NOT inherited

We do not need to know anything about the implementation of the base class

In C++ we have 3 kinds of inheritance - public, private and protected. For all these kinds, **Private members of base class are never inherited**.

We may add new data members or member functions to the derived class

Public Inheritance

To derive a class <der> from a base class <base> we may use the syntax

```
class <der> : public <base>
{
   ....
   ....
};
```

All the public members of base class become public members of derived class.

Private Inheritance

To derive a class <der> from a base class <base> we may use the syntax

```
class <der> : private <base>
{
   ....
   ....
};
```

All the public members of base class become private members of derived class.