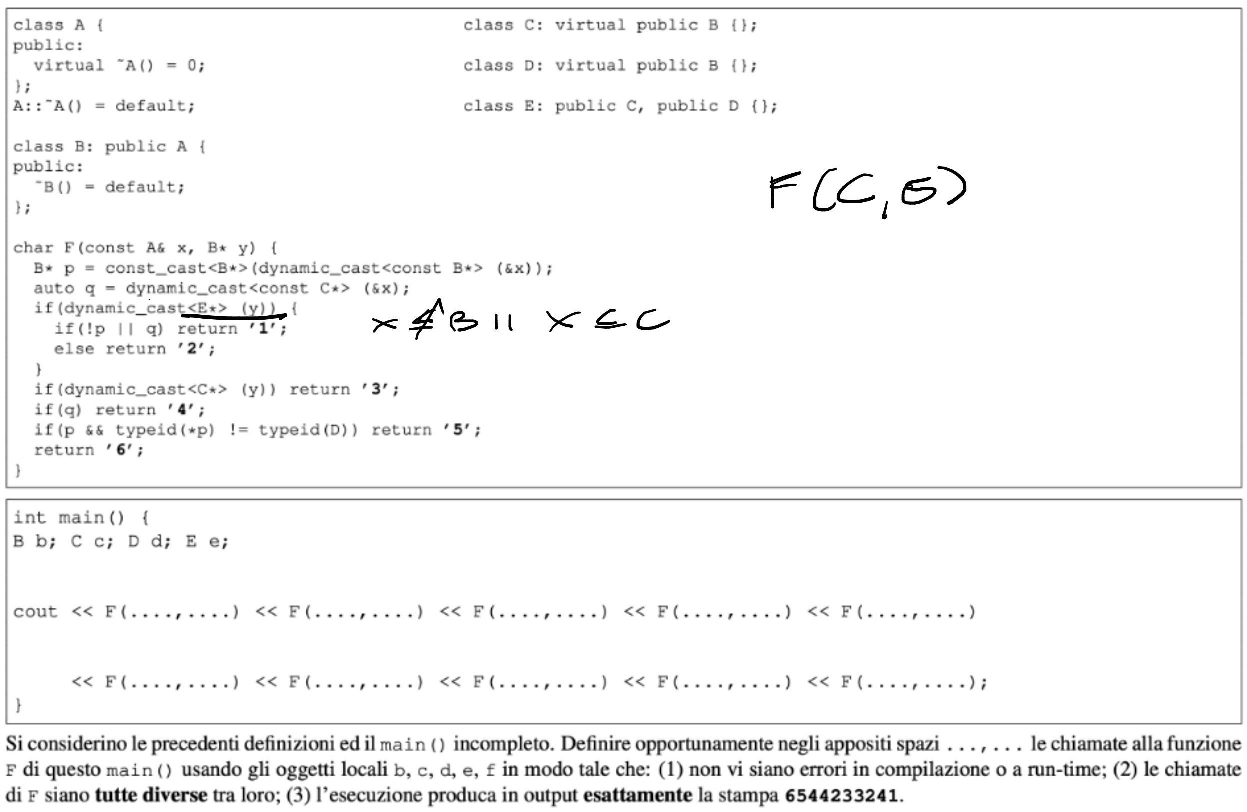


```
class A {                                   class C: virtual public B {};
public:
  virtual ~A() = 0;                         class D: virtual public B {};
};
A::~A() = default;                          class E: public C, public D {};

class B: public A {
public:
  ~B() = default;
};

char F(const A& x, B* y) {
  B* p = const_cast<B*>(dynamic_cast<const B*> (&x));
  auto q = dynamic_cast<const C*> (&x);
  if(dynamic_cast<E*> (y)) {
    if(!p || q) return '1';
    else return '2';
  }
  if(dynamic_cast<C*> (y)) return '3';
  if(q) return '4';
  if(p && typeid(*p) != typeid(D)) return '5';
  return '6';
}
```

F(C, E)

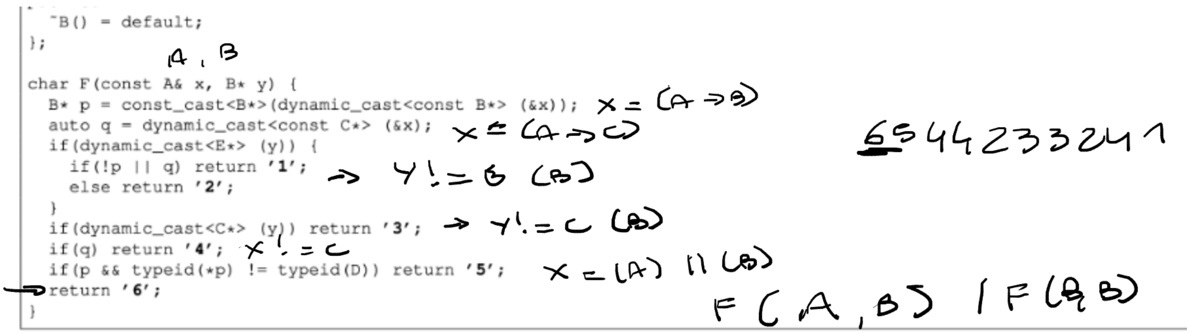
$x \nleq B \,||\, x \leq C$

```
int main() {
B b; C c; D d; E e;


cout << F(....,....) << F(....,....) << F(....,....) << F(....,....) << F(....,....)


     << F(....,....) << F(....,....) << F(....,....) << F(....,....) << F(....,....);
}
```

Si considerino le precedenti definizioni ed il `main()` incompleto. Definire opportunamente negli appositi spazi `...,...` le chiamate alla funzione `F` di questo `main()` usando gli oggetti locali `b`, `c`, `d`, `e`, `f` in modo tale che: (1) non vi siano errori in compilazione o a run-time; (2) le chiamate di `F` siano **tutte diverse** tra loro; (3) l'esecuzione produca in output **esattamente** la stampa **6544233241**.



```
  ~B() = default;
};

char F(const A& x, B* y) {
  B* p = const_cast<B*>(dynamic_cast<const B*> (&x));
  auto q = dynamic_cast<const C*> (&x);
  if(dynamic_cast<E*> (y)) {
    if(!p || q) return '1';
    else return '2';
  }
  if(dynamic_cast<C*> (y)) return '3';
  if(q) return '4';
  if(p && typeid(*p) != typeid(D)) return '5';
  return '6';
}
```

A , B

x = (A → B)

x ≤ (A → C)

→ Y != B (B)

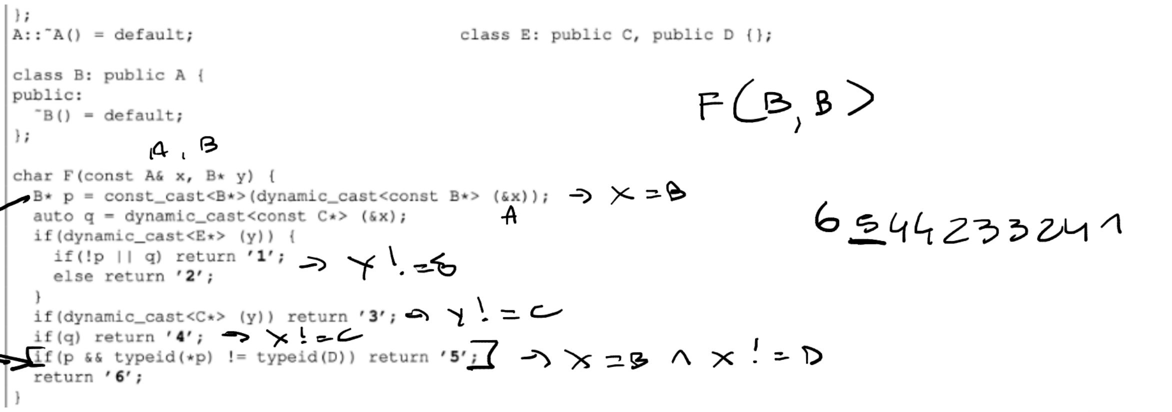
→ Y != C (B)

x != C

x = (A) || (B)

F(A, B) | F(B, B)

$\underline{6}544233241$



```
};
A::~A() = default;                          class E: public C, public D {};

class B: public A {
public:
  ~B() = default;
};

char F(const A& x, B* y) {
  B* p = const_cast<B*>(dynamic_cast<const B*> (&x));
  auto q = dynamic_cast<const C*> (&x);
  if(dynamic_cast<E*> (y)) {
    if(!p || q) return '1';
    else return '2';
  }
  if(dynamic_cast<C*> (y)) return '3';
  if(q) return '4';
  if(p && typeid(*p) != typeid(D)) return '5';
  return '6';
}
```

F(B, B)

A , B

→ x = B

A

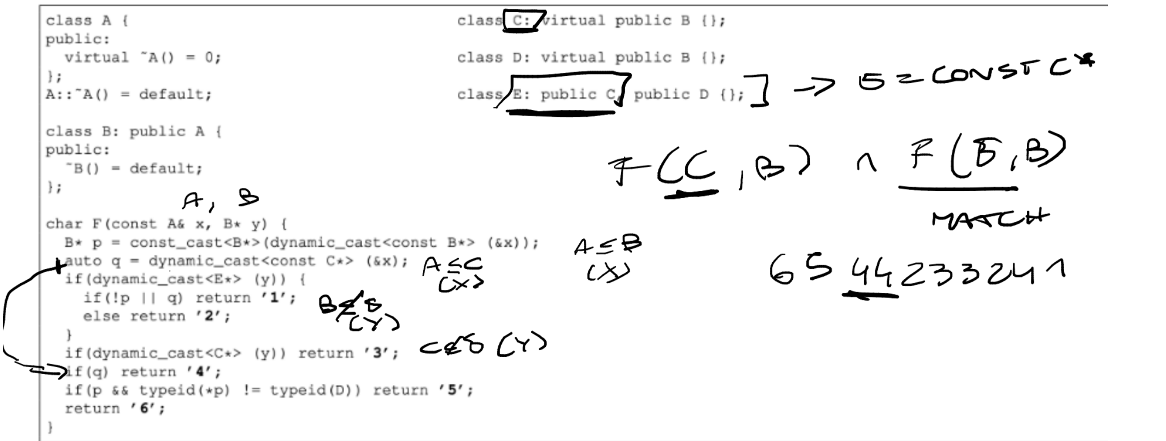
$6\underline{5}44233241$

→ Y != B

→ y != C

→ x != C

→ x = B ∧ x != D



```
class A {                                   class C: virtual public B {};
public:
  virtual ~A() = 0;                         class D: virtual public B {};
};
A::~A() = default;                          class E: public C, public D {};

class B: public A {
public:
  ~B() = default;
};

char F(const A& x, B* y) {
  B* p = const_cast<B*>(dynamic_cast<const B*> (&x));
  auto q = dynamic_cast<const C*> (&x);
  if(dynamic_cast<E*> (y)) {
    if(!p || q) return '1';
    else return '2';
  }
  if(dynamic_cast<C*> (y)) return '3';
  if(q) return '4';
  if(p && typeid(*p) != typeid(D)) return '5';
  return '6';
}
```

→ E = CONST C*

F(C, B) ∧ F(E, B)

MATCH

A, B

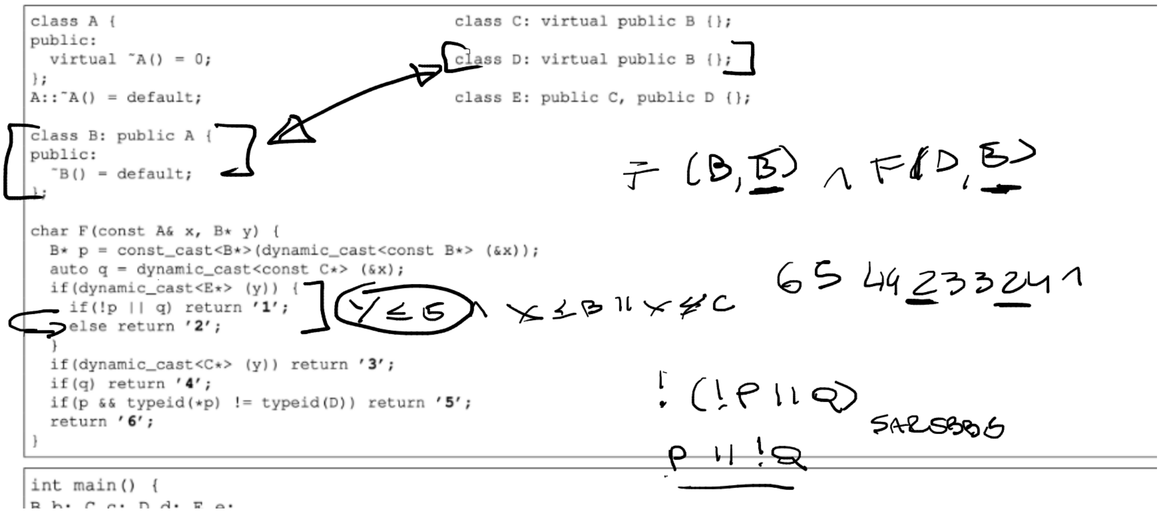
A ≤ C (x)

A ≤ B (x)

B ≰ B (Y)

C ≰ B (Y)

$65\underline{44}233241$



```
class A {                                   class C: virtual public B {};
public:
  virtual ~A() = 0;                         class D: virtual public B {};
};
A::~A() = default;                          class E: public C, public D {};

class B: public A {
public:
  ~B() = default;
};

char F(const A& x, B* y) {
  B* p = const_cast<B*>(dynamic_cast<const B*> (&x));
  auto q = dynamic_cast<const C*> (&x);
  if(dynamic_cast<E*> (y)) {
    if(!p || q) return '1';
    else return '2';
  }
  if(dynamic_cast<C*> (y)) return '3';
  if(q) return '4';
  if(p && typeid(*p) != typeid(D)) return '5';
  return '6';
}
```

```
int main() {
B b; C c; D d; E e;
```

$F(B,\underline{B}) \wedge F(D,\underline{B})$

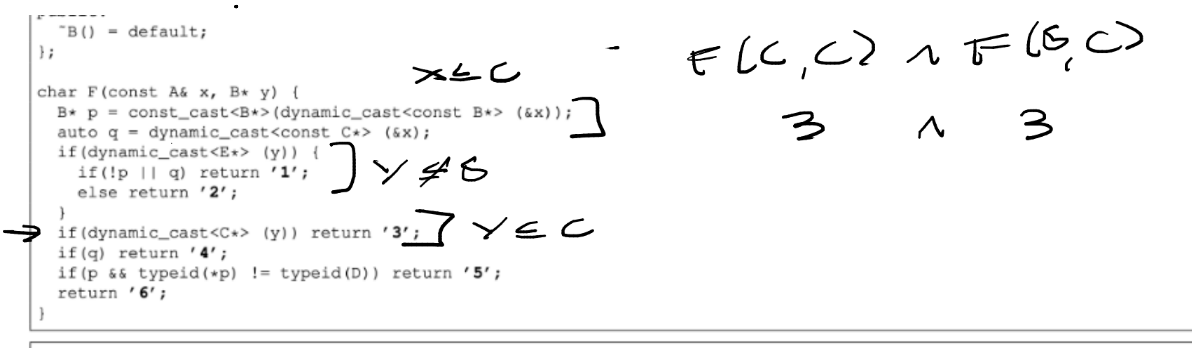
$Y \leq E \wedge X \leq B \,||\, X \notin C$

6 5 44 <u>2</u>3 3 <u>2</u>4 1

!(!p || q)

SARSBBG

<u>p || !q</u>



```
  ~B() = default;
};

char F(const A& x, B* y) {
  B* p = const_cast<B*>(dynamic_cast<const B*> (&x));
  auto q = dynamic_cast<const C*> (&x);
  if(dynamic_cast<E*> (y)) {
    if(!p || q) return '1';
    else return '2';
  }
  if(dynamic_cast<C*> (y)) return '3';
  if(q) return '4';
  if(p && typeid(*p) != typeid(D)) return '5';
  return '6';
}
```

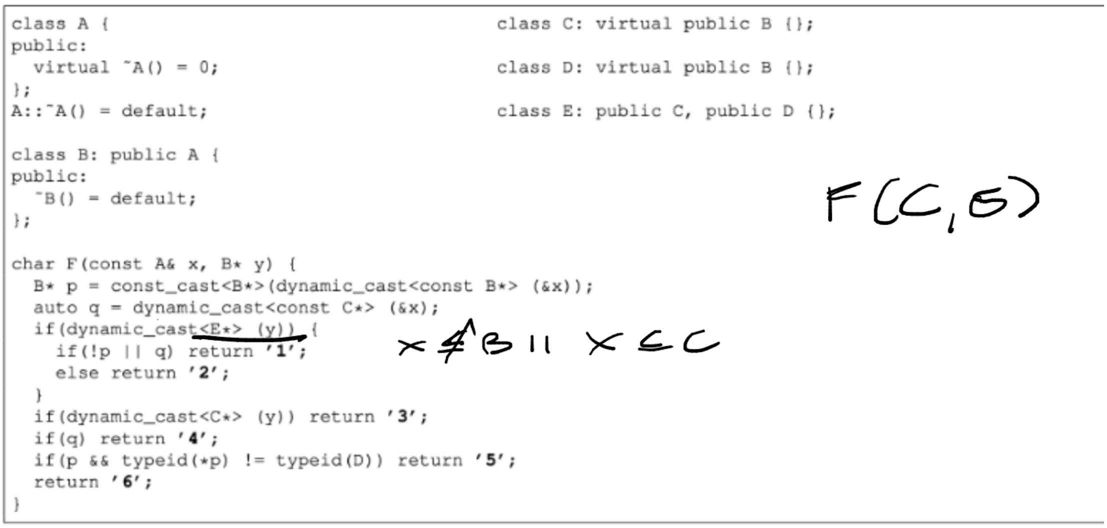
$x \leq C$

$F(C,C) \wedge F(E,C)$

$3 \wedge 3$

$Y \notin E$

$Y \in C$



```
class A {                                   class C: virtual public B {};
public:
  virtual ~A() = 0;                         class D: virtual public B {};
};
A::~A() = default;                          class E: public C, public D {};

class B: public A {
public:
  ~B() = default;
};

char F(const A& x, B* y) {
  B* p = const_cast<B*>(dynamic_cast<const B*> (&x));
  auto q = dynamic_cast<const C*> (&x);
  if(dynamic_cast<E*> (y)) {
    if(!p || q) return '1';
    else return '2';
  }
  if(dynamic_cast<C*> (y)) return '3';
  if(q) return '4';
  if(p && typeid(*p) != typeid(D)) return '5';
  return '6';
}
```

$F(C,E)$

$x \notin B \,||\, x \in C$