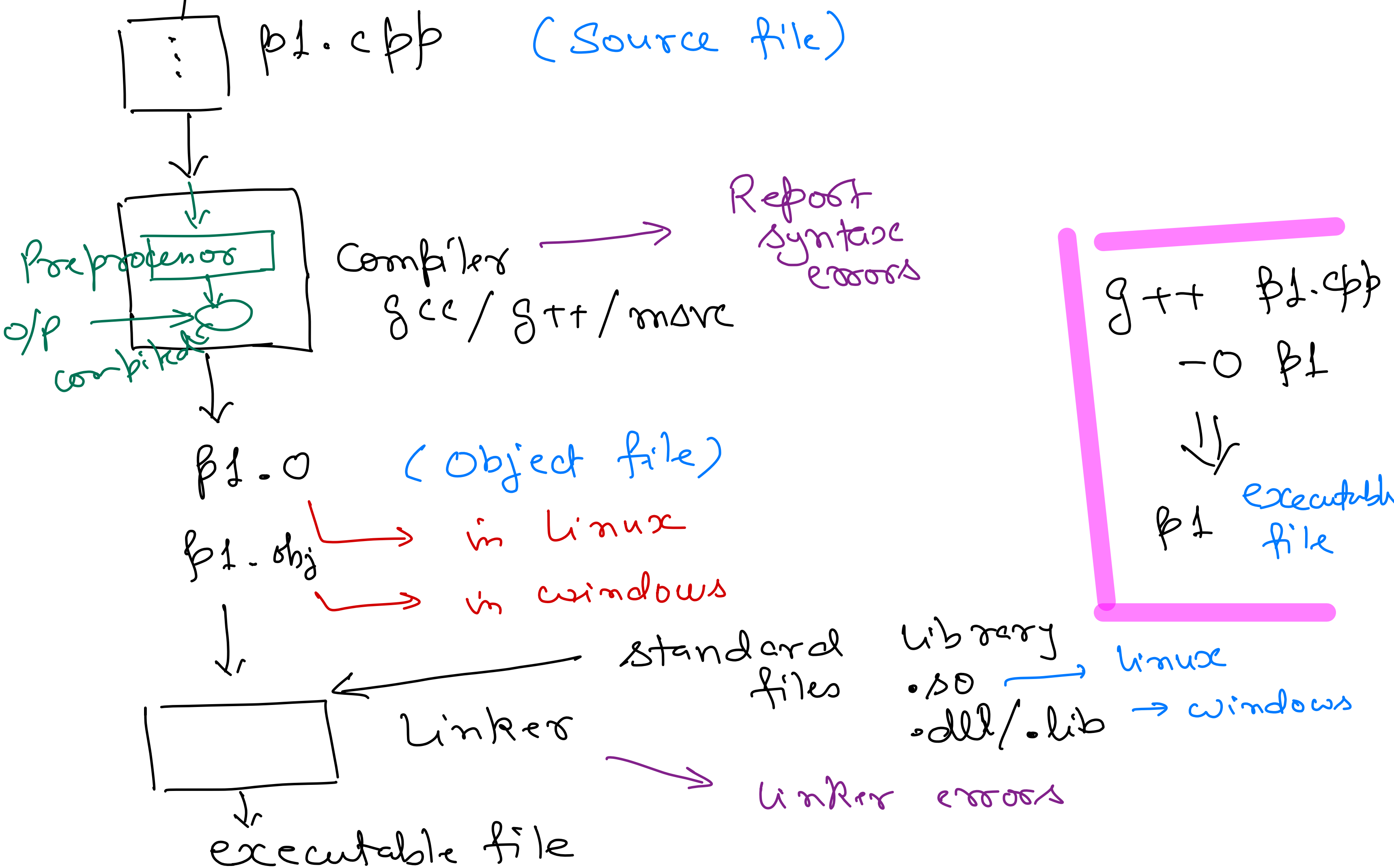
```
# include <iostream>

void  f1 ( );

int  main () {
    f1();
    return 0;
} /* ----*/
void  f1 () {
   std:: cout << "f1";
}
//. ---.
```

Header file

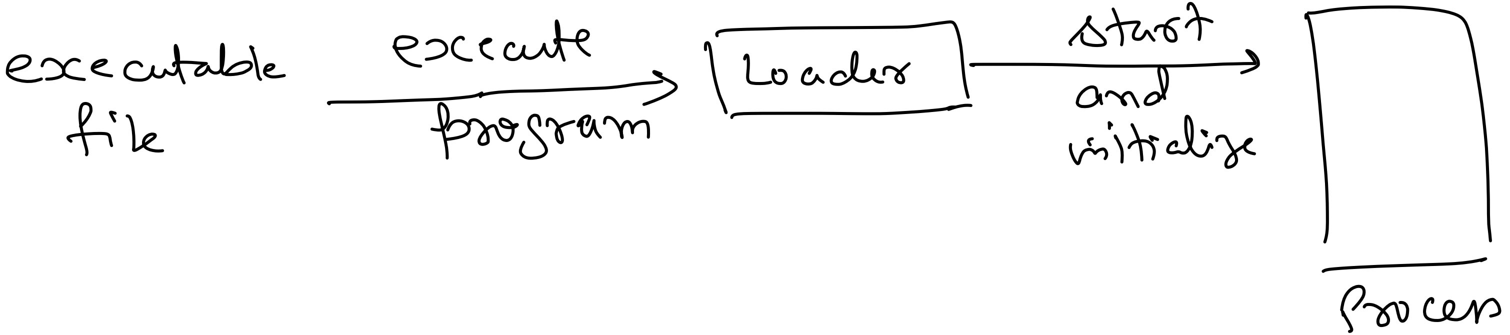
Preprocessor directive

Comment block

Single line comment



```
g++ p1.cpp
  -o p1
    ⇓
  p1
```



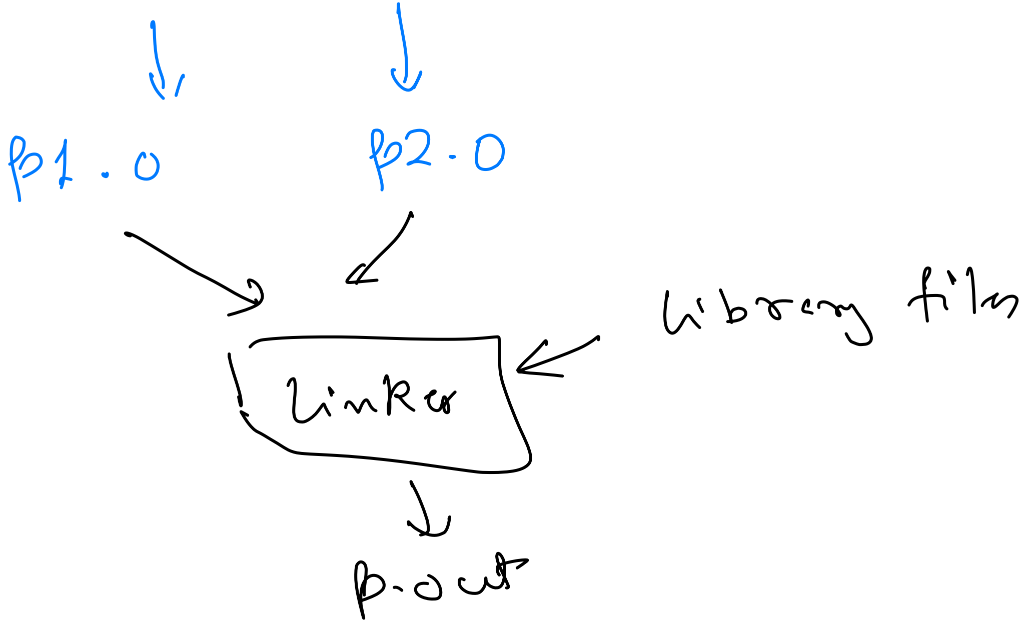
```
void f1();
int main() {
      f1();
      return 0;
}
```

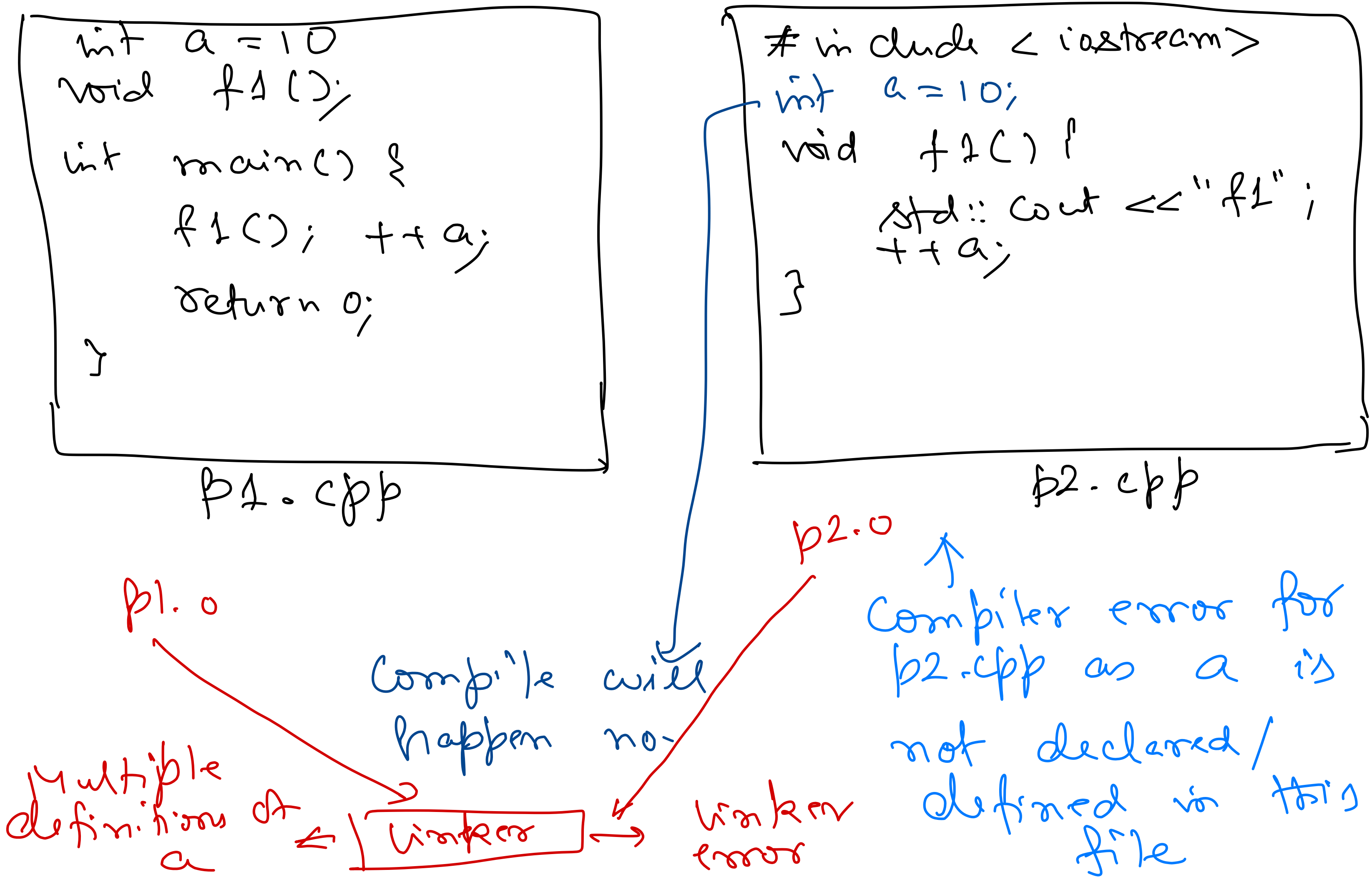
p1.cpp

```
#include <iostream>

void f1() {
     std::cout << "f1";

}
```

p2.cpp

```
$ g++ p1.cpp p2.cpp -o p.out
```





```
int  a = 10
void  f1 ();
int  main() {
     f1(); ++a;
     return 0;
}
```

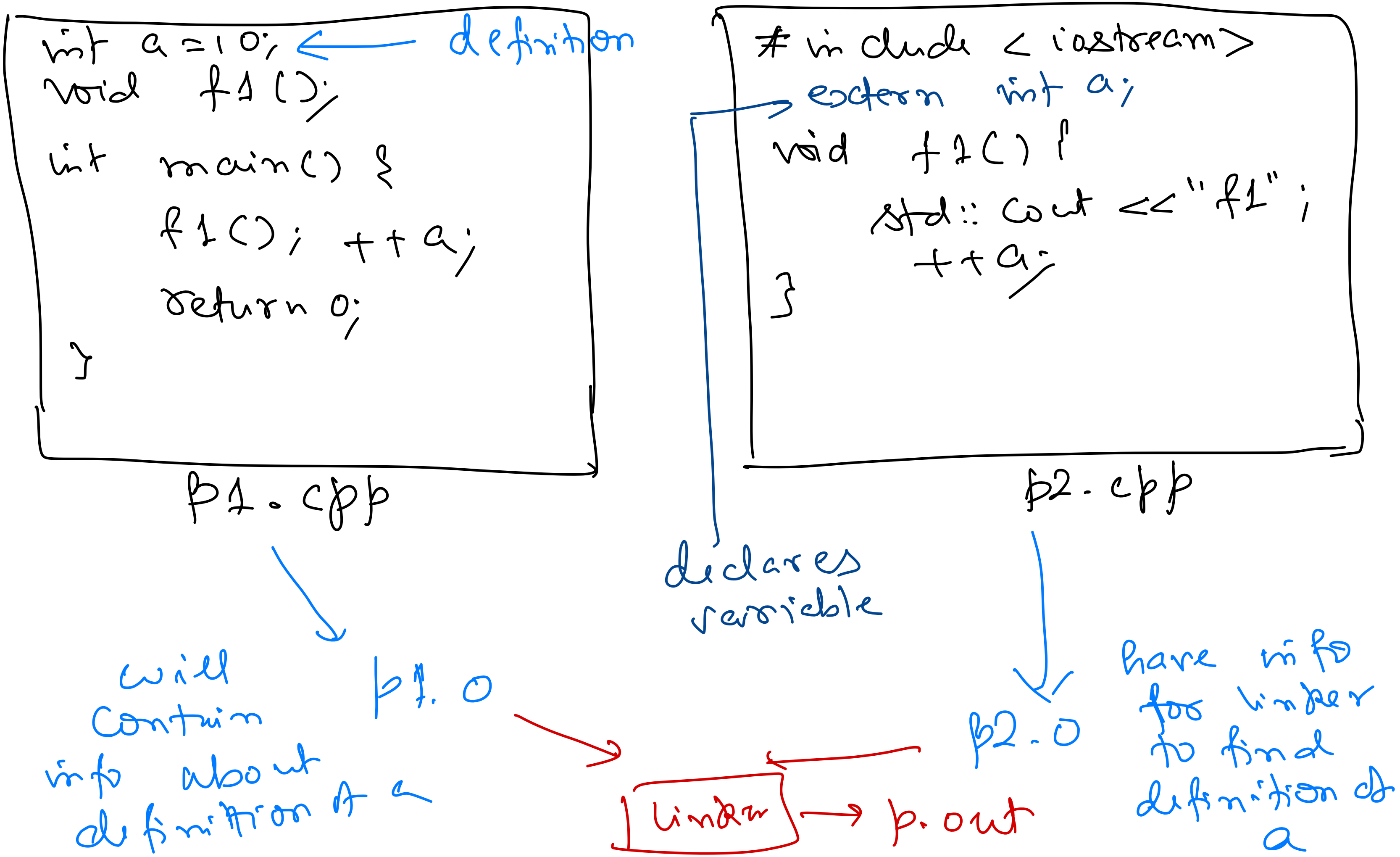
p1.cpp

```
#include <iostream>
int  a = 10;
void  f1() {
     std::cout << "f1";
     ++a;
}
```

p2.cpp

Compile will happen no-

Compiler error for p2.cpp as a is not declared / defined in this file

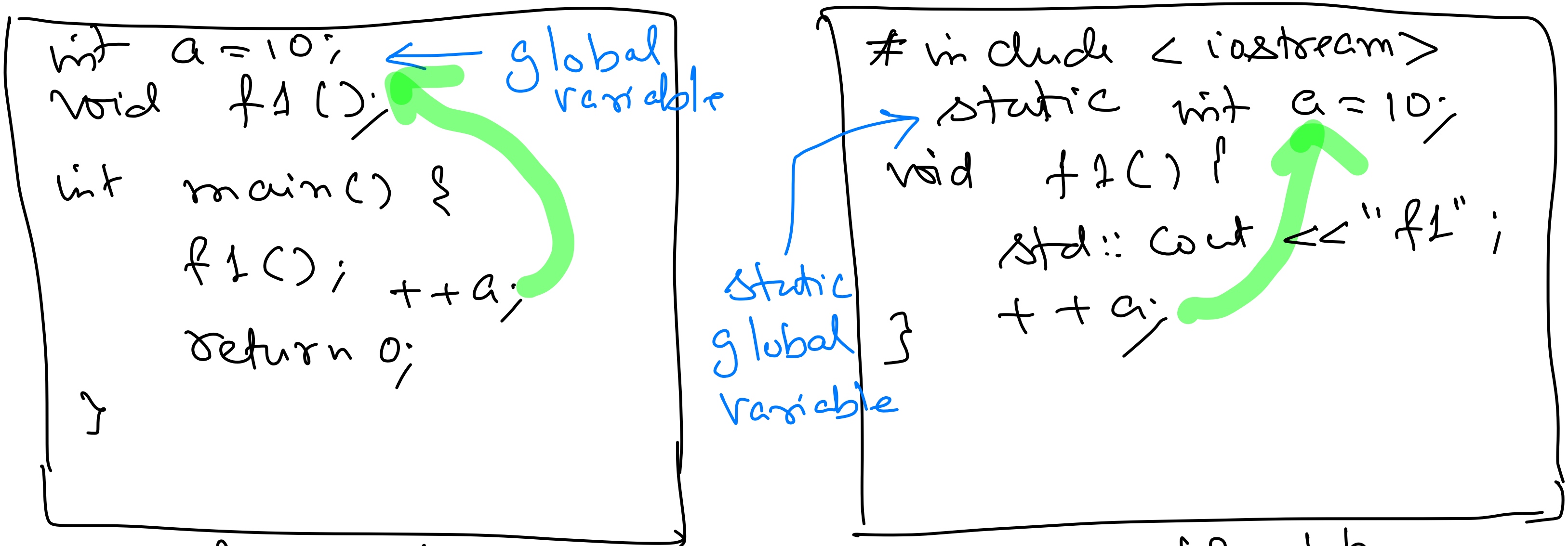


```
int a=10;
void f1();

int main() {
    f1(); ++a;
    return 0;
}
```

```
#include <iostream>
extern int a;
void f1() {
    std::cout << "f1";
    ++a;
}
```

will contain info about definition of a

have info for linker to find definition of a



```
int a=10;
void f1();

int main() {
    f1();
    ++a;
    return 0;
}
```

p1.cpp

```
#include <iostream>
static int a=10;
void f1() {
    std::cout << "f1";
    ++a;
}
```

p2.cpp

		scope	lifetime	initial value
global	→	entire program	program	0
static global	→	source file only	program	0

```
#include <iostream>
void  f1();
int  main() {
        f1();
        f1();
        return 0;
}
void  f1() {
        int  a = 10;
        ++a;
        std::cout << a;
}
```

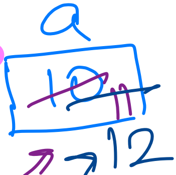
O/P:

11 11

```
#include <iostream>
void  f1();
int  main() {
        f1();
        f1();
        return 0;
}
void  f1() {
static  int  a = 10;
        ++a;
        std::cout << a;
}
```

O/P

11 12



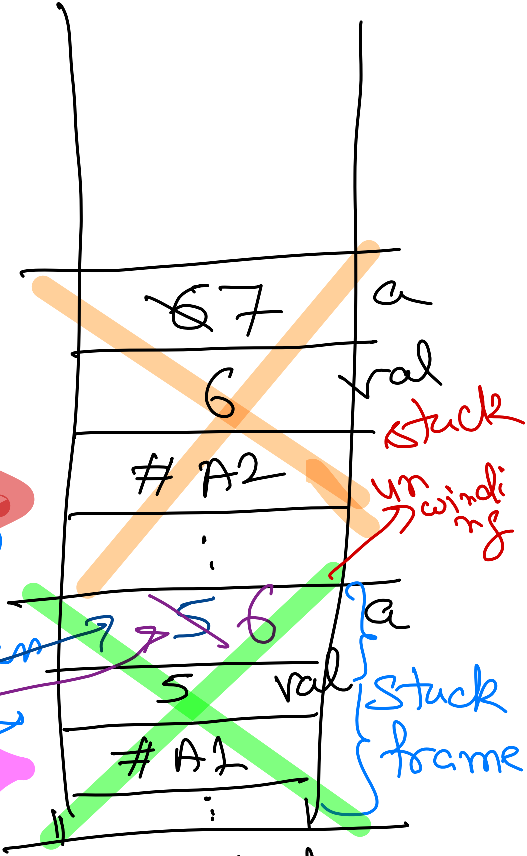
	scope	lifetime	initial value
local	statement block	statement block	garbage
static local	statement block	program	0

⇓

value of static local variable is retained between different calls to same function.

strtok() => uses static local variable

```
# include <iostream>

 int  f1( int  val);

int   main() {
int x = f1 (5); #A1
   x = f1(6); #A2
      return 0;

}
void   f1( int  val) {
   int   a  = val;
   ++a;
   std::  cout  <<  a;
   return  a;
}
```



Stores value of actual arguments, return address + meta data + memory for local variables

Call stack

Used to make function calls.

↓

to return value from function,
normally CPU registers

Exercise

Trace function call using call stack

```
someFunc (4);

int someFunc ( int n) {
if((n == 1) || (n==2))
        return 1;

return someFunc (n-1)
          +
       someFunc (n-2);
}
```

Memory for : global, static global and static local is allocated in data segment

---

```
int ~~sumInt~~ sum ( int a, int b) {
        return a+b;
 }

float ~~sumFloat~~ sum ( float a, float b) {
        return a + b;
}
```

→ function overloading (compile time polymorphism)

Function overloading

- → function names must be same.
- → there must be a difference in argument list.
  - → number of arguments
  - → type of argument(s).

```
int a = 5, b = 10;
float f1 = 1.0, f2 = 2.5;

sum (a, b);  →  sum (int, int)
```

C++ function names are mangled.

name mangling

⇓

generate unique names for each function

```
int sum(int a, int b){   ⇒ sum@i@i
        :
 }
float sum( float a, float b){   ⇒ sum@f@f
        :
 }
```

```
f1( 5);       ⟹   o/p : 5  0

f1 ( 5, 10);  ⟹   o/p: 5 10
```

```
void  f1( int  a,   int b = 0) {
    std::cout << a
              << b;
}
```

default value to function argument

void f2(int ~~a = 0~~, int b){

};

↓

if an argument is taking default value then all arguments to right of it must also take default values.

```
void f1 (int a) {
      :
}
void f1 (int a, int b = 0) {
      :
}
```

```
f1(10);
```

X → ambigious

```
int  main() {
    int  a = 5,  b = 10;

    int t;
    t = a;  a = b;  b = t;

   return 0;
}
```

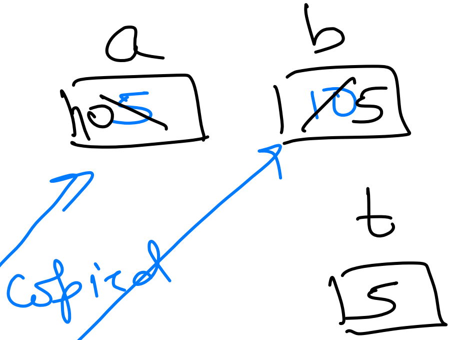
Swap values of two int

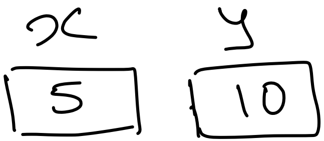
```
void   swap (int  a,  int  b) {
       int   t;
       t = a;   a = b;   b = t;
}

int   main() {
      int   x = 5,   y = 10;
      std:: cout  <<  x  <<  y;
      swap (x , y)
      std:: cout  <<  x  <<  y;
      return;
}
```

→ o/p 5 10

→ o/p 5 10

Call by value.



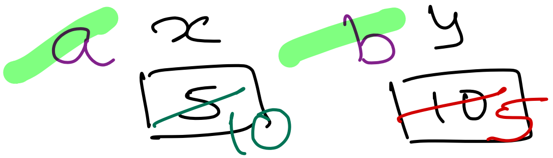


Call by value → copy of actual argument is made in formal argument & any changes to formal arg do not reflect back to actual.

Pass by reference → alias to an existing variable.

```
void swap (int &a, int &b) {
        int t;
        t = a;   a = b;   b = t;
}
```

reference variables



```
int   main( ) {
      int  x = 5,  y = 10;
      std:: cout << x << y;      o/p : 5  10
      swap (x,  y);
      std:: cout << x << y;      o/p : 10  5
      return 0;
}
```