В языке С используется раздельная компиляция файлов: сначала каждый файл компилируется по отдельности и затем результаты (объектные модули) объединяются линковщиком в исполняемый модуль.

first.cpp

```
int a;

int main() {
  a = 5;

  return 0;
}
```

second.cpp

```
void Show(int val) {
  printf("%d", val);
}
```

Компиляция

first.obj + second.obj

Линковка

first.exe

Если у нас есть только один файл, то работа с переменными происходит следующим образом:

```
int a;

void Show(int val) {
   printf("%d", val);
}

int main() {
  a = 5;

  Show(a);

  return 0;
}
```

Выделим в памяти место для хранения переменной типа **int**

1000

Поместим число 5 в то место, где хранится переменная **a**

1000

Подготовим параметры функции: из того места, где хранится переменная **a**, возьмем число 5 и поместим его в стек.
Сохраним адрес возврата в стеке.
Сделаем переход на код функции.

В случае нескольких файлов они не видят переменные, объявленные в другом месте. Например, переменная a объявлена в первом файле и поэтому она недоступна во втором. При компиляции второго файла (а она проходит отдельно) компилятор не имеет информации о данной переменной и поэтому выдаст ошибку.

first.cpp

```
int a;

int main() {
  a = 5;

  return 0;
}
```

second.cpp

```
void Show(int val) {
  printf("%d", val  *  a);
}
```

Компиляция

identifier "a" is undefined

Спецификатор хранения extern подсказывает компилятору, что данная переменная объявлена где-то в другом месте и дает информации о ее типе.

Теперь компилятор может скомпилировать второй файл, подразумевая, что настоящий адрес переменной a будет затем подставлен линковщиком.

first.cpp

```
int a;

int main() {
  a = 5;

  return 0;
}
```

second.cpp

```
extern int a;

void Show(int val) {
  printf("%d", val  *  a);
}
```

first.cpp

```
int a;

int main() {
  a = 5;

  return 0;
}
```

Выделим в памяти 4 байта для хранения переменной типа **int**

Поместим число 5 в то место, где хранится переменная **a**

Сообщим компилятору, что переменная **a** имеет тип **int** и определена где-то в другом месте

Поместим число 25 в пока неизвестное место (подставим ???? вместо адреса и пометим это место в коде модуля)

second.cpp

```
extern int a;

void Show(int val) {
   a = 25;
}
```

Линковка: вместо ???? подставим реальный адрес переменной a, взятый из модуля first.cpp

first.cpp

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

second.cpp

```
extern int a;

void Show(int val) {
   a = 25;
}
```

Если переменная a не нигде не объявлена, то мы получим ошибку при линковке нашей программы.

Линковка: вместо ???? пытаемся подставить реальный адрес переменной a, но его нигде нету: ошибка линковки

Error LNK2019 unresolved external symbol a

first.cpp

```
int a;

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

second.cpp

```
int a;

void Show(int val) {
    a = 25;
}
```

Линковка: в проекте дважды объявлена переменная a - ошибка линковки

Error LNK2005 "int a" (?a@@3HA) already defined

Спецификатор хранения static делает переменную локальной для данного модуля (файла) и позволяет использовать одноименные глобальные переменные в разных модулях. Следующая программа корректно скомпилируется и слинкуется.

first.cpp

```
int a;

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

second.cpp

```
static int a;

void Show(int val) {
   a = 25;
}
```

В этом примере компилятор не имеет информации о функции Show при компиляции первого файла. Поэтому мы получим ошибку компиляции.

first.cpp

```
int a;

int main() {
  a = 5;

  Show(a);
  return 0;
}
```

second.cpp

```
void Show(int val) {
  printf(“%d”, val);
}
```

Сейчас мы даем компилятору информацию о функции, определяя ее прототип. Эта программа корректно скомпилируется.

first.cpp

```
int a;

void Show(int val);

int main() {
  a = 5;

  Show(a);
  return 0;
}
```

second.cpp

```
void Show(int val) {
  printf("%d", val);
}
```

Часто в больших проектах создают специальные заголовочные файлы, в которых перечисляют нужные различным модулям прототипы функций:

first.cpp

```
#include “funct.h”

int a;

int main() {
  a = 5;

  Show(a);
  return 0;
}
```

second.cpp

```
void Show(int val) {
  printf(“%d”, val);
}
```

funct.h

```
void Show(int val);
```