כדי לצור תוכניות מודולריות ב- C נרכז את כל הפונקציות והמשתנים המשותפים להן בקובץ אחד ונייצא את המישק של מודול זה בקובץ h.

לדוגמא: נאמר שבתכנית קיימים שלשה מודולים: x, y, foo. לכל אחד מהם יהיה קובץ C המכיל את הפונקציות והמשתנים הפנימיים, וקובץ h המכיל את הפונקציות והמשתנים הגלובליים אותם מעוניינים לייצא למודולים אחרים. פונקציות ומשתנים שלא יופיעו בקובץ h לא יהיו בשמוש בקבצים אחרים.

הקובץ foo.c יראה כך

**/\* foo.c \*/
/\* Import needed interfaces of other modules: \*/
#include "x.h"
#include "y.h"
/\* Implements this interface: \*/
#include "foo.h"**

**int var1;
static int var2;**

**void Fun1(int \*p) { ... }
static void Fun2(void) { ... }**

הקובץ foo.h יראה כך:

**/\* foo.h \*/
extern int var1;
extern void Fun1(int \*);**

המשתנה var2 והפונקציה Fun2 הוגדרו עם מילת המפתח static ולכן הם מקומיים לקובץ foo.c. המשתנה var1 והפונקציה Fun1 הם גלובלים. כל קובץ (למשל x.c ו- y.c) יוכל להשתמש בהם על ידי הוספת #include "foo.h" בראשו. הקובץ foo.h מכיל את ההגדרות הכוללות את מילת המפתח extern הנחוצות לתכנת הקשור כדי לחולל את קוד המכונה הנכון בשמוש במשתנים ופונקציות מקובץ אחר.

מה קורה אם התכנת שכח לכתוב #include “foo.h” בקובץ מסויים ומנסה להשתמש בפונקציה Fun1 עם פרמטר שגוי? מבחינת הקומפיילר, אם פונקציה לא מוגדרת היא מקבלת פרמטר integer ומחזירה integer, לכן אם כתבתי Fun1(9) בלי שכללתי את קובץ ההגדרות foo.h לא תתקבל שגיאת קומפילציה למרות שהפונקציה אמורה לקבל מצביע ל integer. הגרסה הבאה לקובץ foo.h מונעת מצבים אלה. חשבו מדוע.

**/\* foo.h – improved \*/
#define var1 Foo\_var1
#define Fun1 FooFun1**

**extern int var1;
extern void Fun1(int \*);**