

# Modular Programming

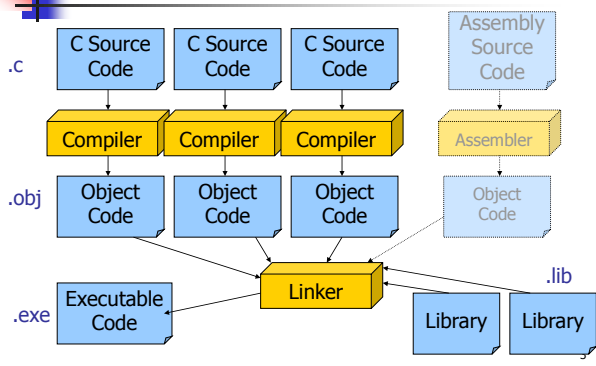


## Intro

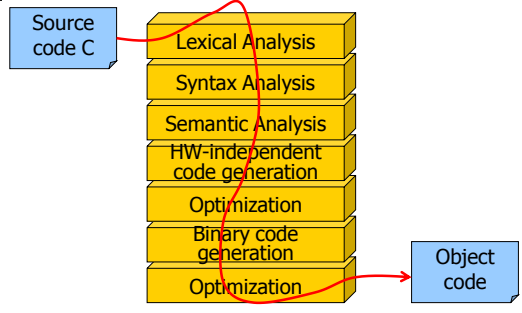
- It is not possible to create complex programs using a single source file:
  - Compiling is slow
  - Difficult reuse of functions
  - Impossible collaboration among different developers
  - Hard to keep track of modifications
- Programs are usually distributed on many source files.

2

## Compiling and Linking C code



## Compiler Structure



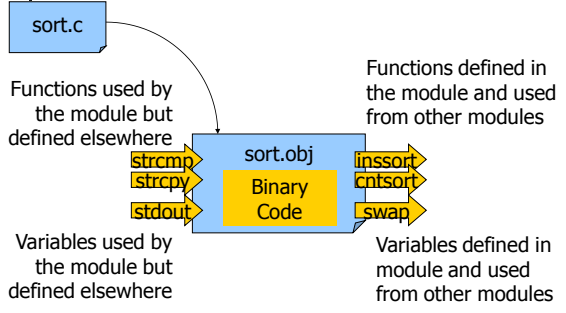
4

## Object code module

- It contains binary code corresponding to a source C module
- It contains external references to functions and variables declared in other modules
- It contains functions and variables used by other modules
- Only one module contains function main().

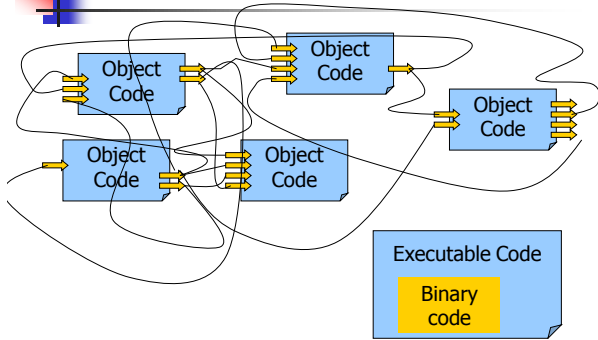
5

## Object Module



6

## Linker

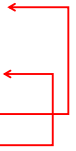


7

## Modularity

Each module in C:

- Defines some "private" functions
- Defines some "public" functions
- Defines some "private" variables
- Defines some "public" variables
- Calls some "extern" functions
- Uses some "extern" variables



8

## Function call (I)

```
void InsertionSort(int A[], int n) ;
/* ... */
void InsertionSort(int A[], int n)
{
    int i, j, key ;
    /* ... */
}
extern void InsertionSort(int A[], int n) ;
/* ... */
main()
{
    int v[10] ;
    /* ... */
    InsertionSort(v, 10) ;
    /* ... */
}
```

9

## Function Call (II)

- All functions in a module (file) are public by default
- Module using an external function must declare its *prototype* (with keyword **extern**)
- Linker will realize the link between modules.

10

## Private functions

- The keyword **static** is used to specify which functions must not be public, so they are just usable within the module.

```
void InsertionSort(int A[], int n) ;
void Deletevector(int A[], int n) ;
void Insertionsort(int A[], int n)
{ . . . }
static void DeleteVector(int A[], int n)
{ . . . }
```

11

## Global Variables

```
int n_elem ;
double vett[MAX] ;
static int temp ;
extern int n_elem ;
extern double vett[MAX] ;
/* ... */
{ for(i=0; i<n_elem; ++i)
  vett[i]++ ;
```

12

## Global Variables

- A global variable in a module is visible by all the other modules by default.
- Module using an external variable must declare its *prototype* (with keyword **extern**)
- The keyword **static** is used to specify which variables must not be public, so they are just visible within the module.
- Local variables (within functions) cannot be shared.

13

## Problems

- The function declaration must be exactly the same on both modules!

```
extern int f(int x); int f( double x ) {
    . . .
    y = f ( z ) ;
    . . .
}
```

**Error!**

14

## Header files

- A header file (file .h) is used to collect definitions of "extern" functions and variables
- The Developer of module M exporting functions/variables must create a header file
- All other modules using module M have to include the header of M (**#include**).

15

## Example of header file

```
extern void InsertionSort(int A[], int n);
extern int n_elem;
extern double vet[MAX];

#include "sort.h" /* used to check consistency */
int n_elem;
double vet[MAX];
static int temp;
void InsertionSort(int A[], int n)
{ . . . }
static void Cancelvector(int A[], int n);
{ . . . }
```

16

## Example of header file (II)

```
extern void InsertionSort(int A[], int n);
extern int n_elem;
extern double vet[MAX];
```

```
#include "sort.h" /* import declarations */
main()
{
    int v[10];
    /* ... */
    InsertionSort(v, 10);
    /* ... */
    for(i=0; i<n_elem; ++i)
        vet[i]++;
}
```

17

## Notes

- Each module usually exports some functions/variables and it imports some others
- It is best practice to include in each module its own header file
- It is better to limit the number of shared variables
- Also include **#define** in header files
- Groups of logically related modules can share a single header file

18