Header Files

The functions that make up a C program need not all be compiled at the same time; the source text of the program may be kept in several source and header files.

Example: One program, one source file

```c
#include <stdio.h>

#define MAXSIZE 100

typedef struct {
  char *name;
  int   score;
} DATA;

void getData( void );
void doCalc( void );
void print( void );

int main( void )
{
   getData();
   print();
   doCalc();
   print();
   return 0;
}

/* ================================  */
void getData( void )
{
   printf( " Function getData\n" );
}

/* ================================  */
void DoCalc( void )
{
   printf( " Function doCalc\n" );
}

/* ================================  */
void Print( void )
{
   printf( " Function print\n" );
}
```

The scope of a name is the part of the program within which the name can be used.

> For a **variable** declared at the beginning of a function, the scope is the function within the name is declared. Local variables of the same name in different functions are unrelated.

The same is true of the **formal parameters** of a function, which are in effect local variables.

The scope of a name defined with **#define** is from its point of definition to the end of the source file.

The scope of a name defined with **typedef** is from its point of definition to the end of the source file.

Let us consider dividing the previous program into several source files, as it might be if each of the components were considerably bigger.

- The **main** function would go in one file, which we call **leader.c**;
- **getData** and **doCalc** go into a second file, **part_A.c**;
- **print** goes into a third file, **part_B.c**.

There is one more thing to worry about: the definitions and declarations shared among the files. As much as possible, we want to centralize this, so that there is only one copy to get right and keep right as the program evolves. Accordingly we will place this common material in a header file, called **team.h,** which will be included as necessary.

The resulting program looks like this:

Header file **team.h**

```
#include <stdio.h>

#define MAXSIZE 100

typedef struct {
  char *name;
  int   score;
} DATA;

void GetData( void );
void DoCalc( void );
void Print( void );
```

Source file **leader.c**

```
#include "team.h"

int main( void )
{ // ...
}
```

Source file **part_A.c**

```
#include "team.h"

void GetData( void
{ // ...
}

void DoCalc( void )
{ // ...
}
```

Source file **part_B.c**

```
#include "team.h"

void Print( void )
{ // ...
}
```

There is a tradeoff between the desire that each file have access only to the information it needs for its job and the practical reality that it is harder to maintain more header files. Up to some moderate program size, it is probably best to have one header file that contains everything that is to be shared between any two parts of the program. For a much larger program, more organization and more headers would be needed.

File inclusion makes it easy to handle collections of

- #defines,
- type definitions, and
- prototype declarations.

Any source line of the form

#include "*filename*"

or

#inlclude <*filename*>

is replaced by the contents of the *filename*. If the *filename* is quoted, searching for the file typically begins where the source program was found.

An included file may itself contain #include lines.

#include is the preferred way to tie the declarations together for a large program. It guarantees that all the source files will be supplied with the same definitions and prototype declarations. When an included file is changed, all files that depend on it must be recompiled.

Conditional inclusion

If a project consists of several source and header file, it is recommended to use conditional inclusion as shown below:

```
#ifndef TEAM
#define TEAM

#include <stdio.h>

#define MAXSIZE 100

typedef struct {
  char *name;
  int   score;
} DATA;

void getData( void );
void doCalc( void );
void print( void );

#endif
```