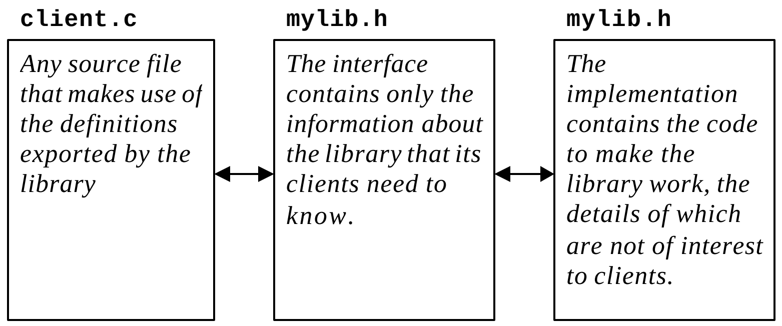
Libraries and Interfaces

- An **interface** is the boundary between the implementation of a library and programs that use that library (i.e. its *clients*) .

- The purpose of an interface is to provide each client with the information it needs to use the library without revealing the details required by the implementation.

- In C, an interface is represented by a **header file**, which traditionally has the same name as the file that implements it with the **`.c`** extension replaced by **`.h`**.

  For example, you created a collection of functions that you want to make available to clients as a library. You need to create two files:
  - an interface (`mylib.h`)
    - contains only the functions prototypes
  - corresponding implementation (`mylib.c`)



client.c
Any source file that makes use of the definitions exported by the library
mylib.h
The interface contains only the information about the library that its clients need to know.
mylib.h
The implementation contains the code to make the library work, the details of which are not of interest to clients.

- Putting the prototypes in the interface makes them available to clients and is called **exporting** those functions.

- Interfaces can export :
  - function prototypes
  - data types
  - constants

- In computer science the term **package** is used to describe the software that defines a library. That is:
  - a **`.h`** file, and
  - corresponding **`.c`** file

Example: Random Numbers

In order to generate a random number in your program, you must use the standard library functions: `rand(), srand()` and `time().`

You must include the following libraries in your program:

```
#include <stdio.h>  //for standard input output
#include <stdlib.h> //for using the random functions
#include <time.h>  // for using the time function
```

The following program illustrates the use of these functions:

```
int main()
{
     int x, y, z, i;
     int loop;

     srand(time(NULL)); /* this basically turns on the
                          random generator  */

     printf("Please Enter How many Random Numbers you would”
            “ like to generate\n");

     scanf("%d", &loop);

     for(i = 1; i<=loop; i++){

     // rand is a function that will return an integer.
        x = rand(); //will find random number btw 0 and 32767
        y = rand()%100;  //will find random number btw 0 and 99
        z = ((rand()%100)-50); /* will find random number
                                  btw –50 and 49 */

        printf(" %d   %d   %d\n", x, y, z);
     }

     return 0;

}
```

A user defined library

The `random.h` interface

```
/*
 * File: random.h
 * --------------
 * This interface provides several functions for
 * generating random numbers.
 */

#ifndef _random_h
#define _random_h

/*
 * Function: RandomInteger
 * Usage: n = RandomInteger(low,high);
 * ----------------------------------
 * This function returns a random integer in the
 * range low to high, inclusive.
 */

int RandomInteger(int low, int high);

/*
 * Function: RandomReal
 * Usage: d = RandomReal(low,high);
 * -------------------------------
 * This function returns a random real number in
 * the half open interval [low,high).
*/

double RandomReal(double low, double high);
```

```
/*
 * Function: RandomChance
 * Usage: if (RandomChance(p)) ...;
 * -------------------------------
 * This function returns TRUE (1) with the probability
 * indicated by p, which should be a floating point
 * number between 0 and 1. For example, calling
 * RandomChance(.30) returns TRUE 30 percent of the
 * time.
 */

int RandomChance(double p);

/*
 * Function: Randomize
 * Usage: Randomize();
 * -------------------
 * This function initializes the random-number
 * generator so that its results are unpredictable.
 * If this function is not called, the other
 * functions will return the same values on each run.
*/

void Randomize(void);

#endif
```

The `random.c` implementation

```
/*
 * File: random.c
 * --------------
 * This file implements the random.h interface.
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "random.h"

/*
 * Function: RandomInteger
 * -----------------------------------
 * This function begins by using rand to select an
 * integer in the interval [0,RAND_MAX] and then
 * converts it to the desired range by applying the
 * following steps:
 *
 * 1. Normalize the value to a real number in interval[0,1)
 * 2. Scale the resulting value to the appropriate
 *    range size
 * 3. Truncate the scaled value to an integer
 * 4. Translate the integer to the appropriate starting
 *    point.
 */
int RandomInteger(int low, int high)
{
   int k;
   double d;

   d = (double) rand() / ((double)RAND_MAX + 1);
   k = (int) (d*(high – low + 1));
   return (low + k);
}
```

```
/*
 * Function: RandomReal
 * --------------------
 * The implemetation of RandomReal is similar to that of
 * RandomInteger, without the truncation step.
*/

double RandomReal(double low, double high)
{
   double d;

   d= (double) rand()/(double)RAND_MAX + 1;
   return (low + d * (high – low));
}


/*
 * Function: RandomChance
 * ----------------------
 * This function uses RandomReal to generate a real
 * number in the interval [0,1) and then compares that
 * value to p.
 */

int RandomChance(double p)
{
   return(RandomReal(0,1) < p);
}

/*
 * Function: Randomize
 * -------------------
 * This function operates by setting the random number
 * seed to the current time.
*/

void Randomize(void)
{
   srand((int) time(NULL));
}
```

Constructing a client program: Craps

```
/*
 * File : craps.c
 * ------------------
 * This program plays the casino game called craps,
 * which is played using a pair of dice. At the
 * beginning of the game, you roll the dice and compute
 * the total.
 * If your first roll is 7 or 11 you win with what
 * gamblers call a “natural”. If your first roll is 2,
 * 3, or 12 you lose by “crapping out”. In any other
 * case, the total from the first roll becomes your
 * “point”, after which you roll the dice until you
 * roll your point again, in which case you win, or
 * until you roll a 7, in which case you lose. Other
 * rolls, including 2,3, and 12 have no effect during
 * this phase of the game.
 * /

#include <stdio.h>
#include “random.h”

#define TRUE 1
#define FALSE 0

/* Function prototypes */
int TryToMakePoint(int point);
int RollTwoDice(void);
```

```
/* Main program */

int main ()
{
   int point;

   Randomize();
   printf("This program plays a game of craps.\n");
   point = RollTwoDice();
   switch(point){
      case 7: case 11:
         printf("That's a natural. You win.\n");
         break;
      case 2: case 3: case 12:
         printf("That's craps. You lose.\n");
         break;
      default:
         printf("Your point is %d.\n", point);
         if (TryToMakePoint(point))
            printf("You made your point. You win.\n");
         else
            printf("You rolled a seven. You lose.\n");
   }
}
```

```
/*
 * Function: TryToMakePoint
 * ------------------------
 * This function is responsible for the part of the
 * game during which you roll the dice repeatedly until
 * you either make your point or roll a 7. The function
 * returns true or false.
 */

int TryToMakePoint(int point)
{
   int total;

   while (TRUE) {
      total = RollTwoDice();
      if (total == point) return (TRUE);
      if (total == 7) return (FALSE);
   }
}

/*
 * Function: RollTwoDice
 * ---------------------
 * This function rolls two dice and returns their sum.
 * As part of the implementation the result is
 * displayed on the screen.
 */

int RollTwoDice (void)
{
   int d1, d2, total;

   printf("Rolling the dice ... \n");
   d1 = RandomInteger(1,6);
   d2 = RandomInteger(1,6);
   total = d1 + d2;
   printf("You rolled %d + %d = %d\n", d1,d2, total);
   return total;
}
```

Modular Development

Program structured as a single module:

program.c

```
main()
{
   ProcA();
   ProcB();
}

void ProcA(void)
{
   ...
}

void ProcB(void)
{
   ...
}
```

Program divided into separate modules:

main.c

```
#include “module1.h”
#include “module2.h”

main()
{
   ProcA();
   ProcB();
}
```

module1.h

```
void ProcA(void);
```

module2.h

```
void ProcB(void);
```

module1.c

```
#include “module1.h”

void ProcA(void)
{
   ...
}
```

module2.c

```
#include “module2.h”

void ProcB(void)
{
   ...
}
```

Using Files in C

To read or write a file as part of a C program, you must use the following steps:

1. Declare a file pointer variable.

```
FILE *infile, *outfile;
```

2. Open the file.

```
infile = fopen(“phonebook.dat”,”r”);
outfile = fopen(“newbook.dat”,”w”);
```

3. Transfer the data.

   Read data from the file:
   `fscanf, fgetc, getc` etc.

   Write data to the file:
   `fprintf, fputc, putc` etc.

4. Close the file.

```
fclose (infile);
```

Character I/O

- The simplest approach to file processing is to go through files character by character.
- To read a single character you can use the function `getc`:
  `int getc (FILE *infile);`
- To write a single character you can use the function `putc`:
  `int putc (int c, FILE *infile);`
- Example: Copy one file to another by calling the following function:

```
void CopyFile(FILE *infile, FILE *outfile)
{
   int ch;

   while( (ch = getc(infile)) != EOF){
      putc(ch, outfile);
   }
}
```

Updating a file

Suppose you want to modify the contents of an existing file. The process of changing a file is called updating the file and is not as simple as it might seem.

The most common way to update a file consists of the following steps:

1. Open the original file for input.
2. Open a temporary file for output with a different name.
3. Copy the input file to temporary file, performing any updates as you go.
4. Close both files.
5. Delete the original file.
6. Rename the temporary file so that it once again has the original name.

Example

```
/*
 * This program copies a program from one file to
 * another, removing all comments .
 */
#include<stdio.h>
#include<string.h>
#define TRUE 1
#define FALSE 0
void CopyRemovingComments (FILE *, FILE *);

int main()
{
   char filename[20], *temp;
   FILE *infile, *outfile;

   printf("This program removes comments from a file.\n");

   while (TRUE) {
      printf("File name: ");
      scanf("%s", filename);
      infile = fopen(filename,"r");
      if (infile != NULL) break;
      printf("File %s not found. Try again.\n", filename);
   }

   temp = tmpnam(NULL);
   outfile = fopen(temp, "w");
   if (outfile == NULL)
      printf("Error: Can't open temporary file.\n");
   else {
      CopyRemovingComments(infile,outfile);
      fclose(infile);
      fclose(outfile);
      if (remove(filename) != 0|| rename(temp,filename) != 0)
        printf("Unable to rename temporary file.");
   }
}
```

```
void CopyRemovingComments (FILE *infile, FILE *outfile)
{
   int ch, nch;
   int commentFlag;

   printf("Inside Copy function\n");

   commentFlag = FALSE;
   while ( ( ch = getc(infile) ) != EOF) {
     if (commentFlag) {
       if (ch == '*'){
          nch = getc(infile);
          if (nch == '/')
             commentFlag = FALSE;
          else
             ungetc(nch,infile);
       }
      } else {
          if (ch =='/') {
              nch = getc (infile);
              if (nch == '*')
                 commentFlag = TRUE;
              else
                 ungetc(nch, infile);
          }
          if (!commentFlag) putc (ch,outfile);
      }
    }
}
```

Formatted I/O

The `printf` function comes in three different forms:

`printf(`*control string, …*`);`
`fprintf(`*output stream, control string, …*`);`
`sprintf(`*character array, control string, …*`);`

The `scanf` function comes in three different forms:

`scanf(`*control string, …*`);`
`fscanf(`*input stream, control string, …*`);`
`sscanf(`*character array, control string, …*`);`