Make file- Multiple Source Files

1. Whenever we are working on a small project the entire code is contained in a single ".c" file.

2. If there is any change in the code the programmer recompiles that ".c" file and reruns the program.

3. If the project is very large the entire code is split into multiple code files.

Make file- Multiple Source Files

4. Suppose the code of one file is changed and you do not know in which file then all the files have to be recompiled so that the project runs correctly.

5. This process is very time consuming as it requires compiling those files also in which there was no change of code.

6. Also, if a single file in which the code was changed is not recompiled then the project will not work correctly.

Make file- Multiple Source Files

Consider an example to understand the situation

Problem: Design a calculator which can perform addition and subtraction.

Solution: Create three files

cal.c – the main file which takes the input and calls the addition and subtraction functions.

add.c – contains the function add() to perform the addition.

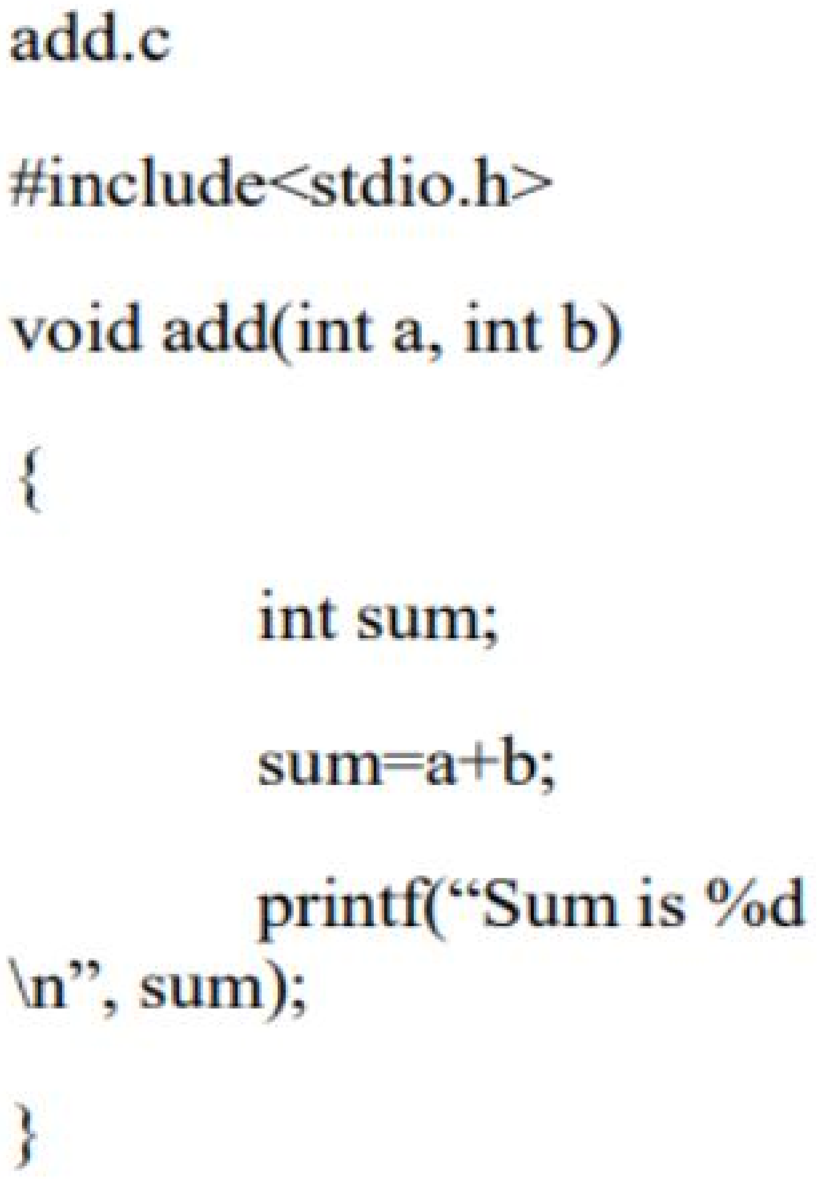
sub.c – contains the function sub() to perform the subtraction.

Make file- Multiple Source Files

cal.c

```
#include<stdio.h>

extern void add(int x, int y);

extern void sub(int x, int y);

int main()

{

        int x=5, y=3;

        add(x,y);

        sub(x,y);

        printf("Executed the
cal.c file\n");

}
```

Make file- Multiple Source Files



add.c

```
#include<stdio.h>

void add(int a, int b)

{

          int sum;

          sum=a+b;

          printf("Sum is %d
\n", sum);

}
```

Make file- Multiple Source Files

sub.c

```
#include<stdio.h>

void sub(int a, int b)

{

            int sub;

            sub=a-b;

            printf("Subtracted
value is %d\n",sub);

}
```

Make file- Multiple Source Files

To run compile all the three files individually

```
$ gcc -c cal.c
$ gcc -c add.c
$ gcc -c sub.c
$ gcc -o calculator cal.o add.o sub.o


$ ./calculator
```

Output:

```
Sum is 8
Subtracted value is 2
Executed the cal.c file
```

Make file- Multiple Source Files

Now suppose the programmer who manages the sub.c files makes some changes in the “printf” line.

(Make some changes on your own).

The manager of the project is aware that some changes have been done but don’t know to which file. So he has to recompile all the files.

```
$ gcc -c cal.c
```

```
$ gcc -c add.c
```

Make file- Multiple Source Files

(Note: add.c is recompiled which had no change made to it and sub.c is missed by the project manager by mistake)

Make file- Multiple Source Files

```
gcc -o calculator cal.o add.o sub.o
./calculator
```

Output:

```
Sum is 8
Subtracted value is 2
Executed the cal.c file
```

The output remains the same and the manager has no idea what went wrong and why no changes have been reflected in the output.

Makefile and the make command

The make command keeps track of the changes made in any of the files and recompiles only the desired file.

To tell make how the project is managed you are required to create a makefile.

The makefile has two important parts:

a) dependencies and b) rules.

Dependencies tell how every file of the project is related to the source files.

Rules tell how the target file is created.

Makefile and the make command

Create a makefile for our example:

1. nano makefile1

(Note: makefile1 is the name of the makefile)

Makefile and the make command

The content of makefile1 are as below

```
calculator:      cal.o    add.o   sub.o
        gcc      -o       calculator     cal.o    add.o   sub.o
cal.o:  cal.c
        gcc      -c       cal.c
add.o:  add.c
        gcc      -c       add.c
sub.o:  sub.c
        gcc      -c       sub.c
```

Makefile and the make command

Create a makefile for our example:

1. nano makefile1

(Note: makefile1 is the name of the makefile)

All the spaces in the file are "tab". The makefile makes calculator dependent on cal.o, add.o and sub.o.

The rule says that whenever there is change in any of these execute

gcc -o calculator cal.o add.o sub.o

Makefile and the make command

To invoke the make command write:

```
make -f makefile1
```

```
./calculator
```

Makefile and the make command

This time do some changes in the add.c file

```
touch add.c
```

```
make -f makefile
```

Output:

```
gcc -c add.c
gcc -o calculator cal.o add.o sub.o
```

Automatically only the affected files are recompiled.

```
./calculator
```

Make file

dbm_close

This routine closes a database opened with dbm_open and must be passed a dbm pointer returned from a previous call to dbm_open .