

**Introduction to Structured Programming**  
**Prof. Shankar Balachandran**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 61**

Welcome to the very last video for this course for this Mooc. So, we have stuck with the course for. So, long and. So, this is the place, when I do not want to teach something completely new. So, this is something that you have done, you have already done the programming, but when you write large piece of software, you may have to do a few things to be just careful and. So, that it becomes usable even later.

So, one thing that happens with the software is that, lot of people start contributing to code and your code that you wrote to be able to read by others and may be they want to change it later or you may even go and changed your own code to do a few other things later and so, on. So, you need to follow a few principles to do this. And one major thing that you have to do is, take a problem and divide that into smaller sub problems. So, that you can solve each of the sub problems separately and put them in different places.

So, it may be you have not seen this or you have not done it so, far, because you are writing smaller program so, far. But, this can really become a problem if you write lots of code. So, to motivate this whole thing, I am going to give a small programming assignment to you, it is not a very hard assignment. So, I want you to go and write a program to print these figures.

(Refer Slide Time: 01:30)

**Development Strategy 1 (Unstructured)**

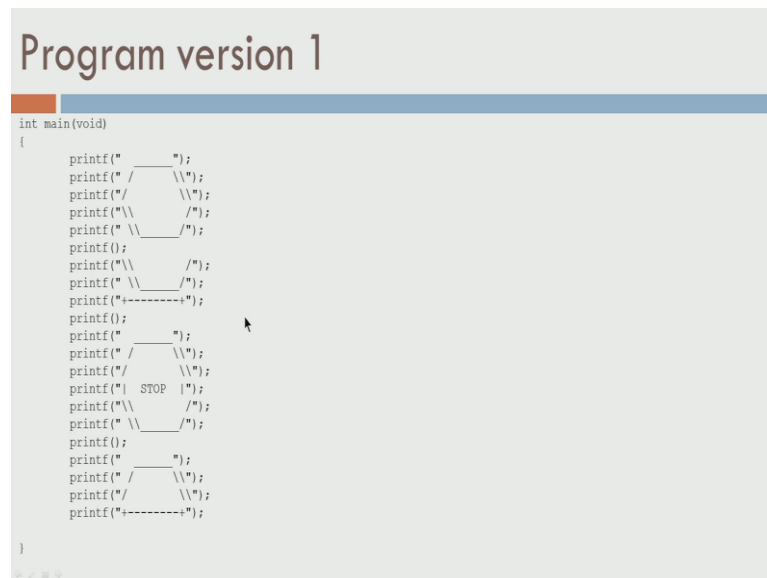
First version (unstructured):  
Create an empty program and `main()`.  
Copy the expected output into it, surrounding each line with `printf()`.  
Run it to verify the output.

The diagrams illustrate the unstructured development process. The first diagram shows a hexagon. The second diagram shows a trapezoid with a dashed line below it. The third diagram shows a hexagon with the word "STOP" inside. The fourth diagram shows a trapezoid with a dashed line below it.

So, there is a hexagon looking thing at the left and then, there is something like a cup, there is a stop sign and we will call this the hat. So, let us say this will call this the egg, the cup, the stop sign and the hat. Let us say, you want to draw these things. So, I am asking you to write a program which you will draw, you could do something very easy. So, in the first line draw dashes, in the second line draw this forward slash, give some space, draw a backward slash, you can do a lots of things.

So, let us look at the very first crack at this problem. The first version is an unstructured program, we will create an empty program and the main function. We will just copy the expected output, but surround each line with printf and run it to verify the output, we can do that very well.

(Refer Slide Time: 02:20)



```
int main(void)
{
    printf("      ");
    printf("/      \\");
    printf("/      \\");
    printf("\\      /");
    printf(" \\      /");
    printf();
    printf("\\      /");
    printf(" \\      /");
    printf("+-----+");
    printf();
    printf("      ");
    printf("/      \\");
    printf("/      \\");
    printf("| STOP |");
    printf("\\      /");
    printf(" \\      /");
    printf();
    printf("      ");
    printf("/      \\");
    printf("/      \\");
    printf("+-----+");
}
```

So, your program would look something like this, int main void, printf you have serious of printf, in each one you can see what you are trying to print. So, the only thing that is slightly different then what we will see in the output is for every back slash, we have to use this escape character called back slash itself. So, every back slash is actually supported by another back slash. The other thing that is missing is the serious of new lines that I have not explicitly added, but otherwise your program would look something like this.

So, this is what I will call an unstructured program. So, you have not put too much thought into it, you are asked to print something like this, you just write a serious of printf statements and this will print some. It will print what do you want, but it is not a

very nice thing to do.

(Refer Slide Time: 03:08)

## Method Decomposition

Step 1:  
Identify the structure of the output.

So, the first thing that we do in any engineering step is take a problem and divided that into sub problems. The same thing applies to writing software. Can I take this problem and divided that into sub problems. So, the main thing that comes with this is what is called decomposition. Can you take the problem and decompose that into smaller sub problems and identify, if there is any structure to the problem itself. So, let us go and look at this.

(Refer Slide Time: 03:34)

## High-Level Structure

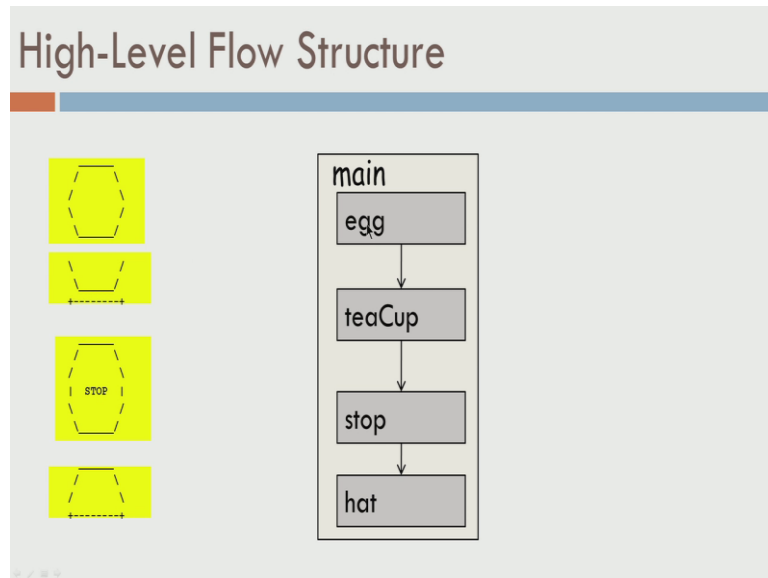
The structure of the output:  
initial "egg" figure  
second "tea cup" figure  
third "stop sign" figure  
fourth "hat" figure

This structure can be represented by methods:  
egg  
teaCup  
stopSign  
hat

So, there is some a structure. So, the first figure will call that the egg, the second figure

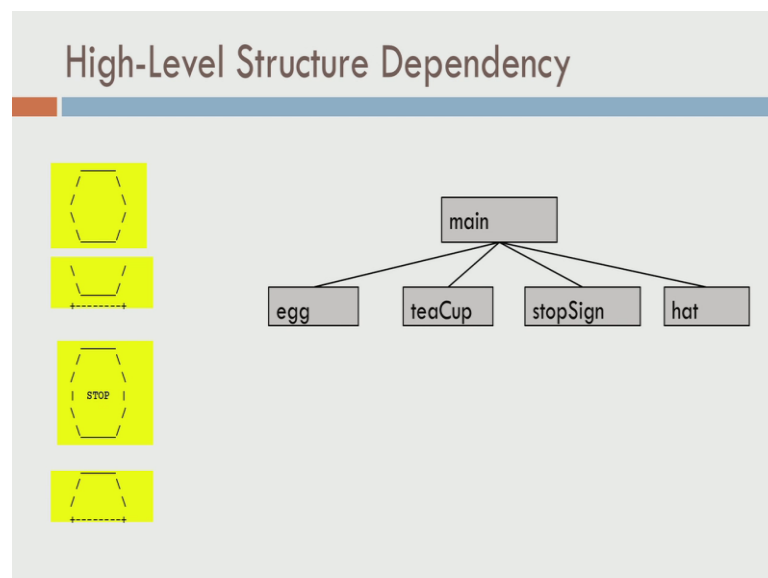
will call that tea cup, the third figure is will call that the stop sign and the fourth figure will call it a hat. So, now I can think of this problem as drawn an egg, draw tea cup, draw a stop sign, draw a hat. That is slightly better than draw some line and draw some other line and draw some other line and so, on. So, this is slightly better.

(Refer Slide Time: 04:05)



Let us see, how this might look like. In your main program, you will have four function calls draw egg, draw tea cup, draw stop sign, draw hat and you would have to write four functions for each one of them egg, tea cup, stop and hat. So, this is something that you can do and it is something that you can write at this level anyway.

(Refer Slide Time: 04:25)



So, you have these four things. So, main depends on the function would depend on the function egg, would depend on the function tea cup, stop sign and hat. So, that is another way to write the program, but this is still not the best thing.

(Refer Slide Time: 04:40)

### Program version 2

```
void main() {
    egg();
    teaCup();
    stopSign();
    hat();
}

void egg() {
    printf("      ");
    printf(" /_____\");
    printf("/      \");
    printf("\\      /");
    printf(" \\_____/");
    printf();
}

void teaCup() {
    printf("\\\\      /");
    printf(" \\_____/");
    printf("+-----+");
    printf();
}
...

```

So, let us go and look at what is happening. Your main program becomes much simpler, the task that you doing in main is I have to do these four tasks in this order, that is all you care about. But, you do not care about how exactly egg is drawn, how tea cup is drawn and so, on, because you have delegated this to a function. Remember, when we did functions I talked about delegations and we did exactly that now. So, when you call this function egg, this function egg has all the printf statements.

Of course, the new lines are missing I did intentionally added, because you can see the structure of the output here clearly. So, series of printf statements, this is slightly better. What you have is from the main program the intend is clear, you want to do draw egg, draw tea cup, draw stop sign and draw hat, but the functions are themselves still ugly, you have only the serious of printf statements. So, this is one level of decomposition.

From, something which is just a sequence of printf statements, you have four functions calls, the main program is clear or the main method is clear, but the internal functions are still a bit ugly. So, now let us see whether we can decompose the problem in little further.

(Refer Side Time: 05:50)

```
Program version 2, cont'd.

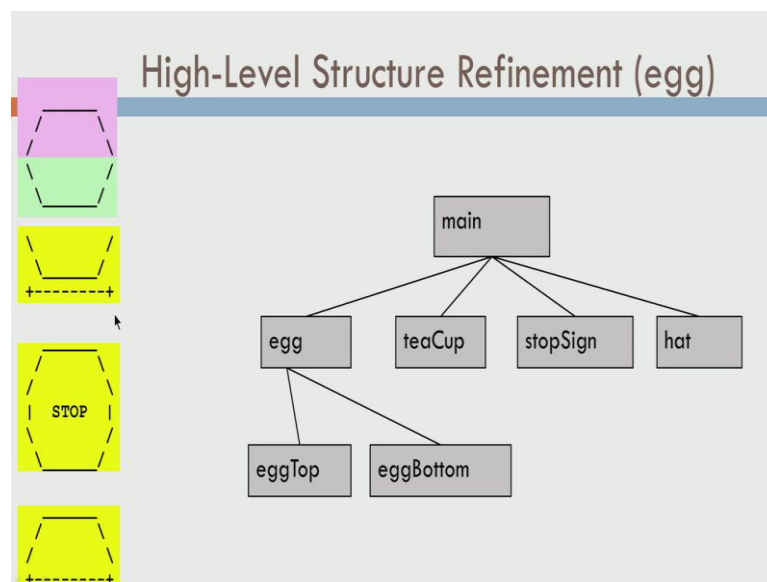
...

void stopSign() {
    printf(" ");
    printf(" / \\\");
    printf("/ \\\");
    printf("|  STOP  |");
    printf("\\ \\\");
    printf(" \\ \\\");
    printf(" /");
    printf();
}

void hat() {
    printf(" ");
    printf(" / \\\");
    printf("/ \\\");
    printf("+-----+");
}
}
```

So, this is actually just finishes the program with all these four egg, tea cup, stop sign and hat. Let us see, if there is something better that we can do about this whole thing, is there some other structures to drawing this or should I be drawing or writing a series of printf statement for each one of these functions. So, let us look at the egg to begin with.

(Refer Slide Time: 06:10)

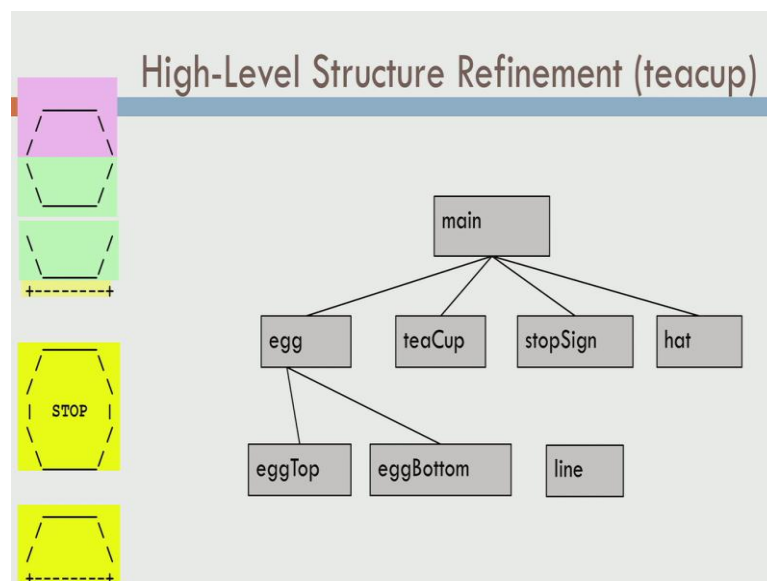


So, the egg there is something in the top half and something in the bottom half. You cannot really draw the top half with the bottom half, code or anything like that, there is no reuse that you can do. So, the top half has to be drawn and the bottom half has to be drawn separately. So, this will require a printf of a line followed by forward slash and the back slash and forward by another line with the forward slash back slash and in this, it

would be a reverse.

So, you cannot reuse anything. So, let us assume that there is a function which can draw the top of the egg, there is another function that can draw the bottom half of the egg. So, we will call those functions egg top and egg bottom, that is the way we have identify this. Now, let us go and look at this next one which is supposed to be for drawing the cup. If you notice, cup has something which is similar to the bottom of the egg, but there is something else also. The saucer here is just a line we need something to do there.

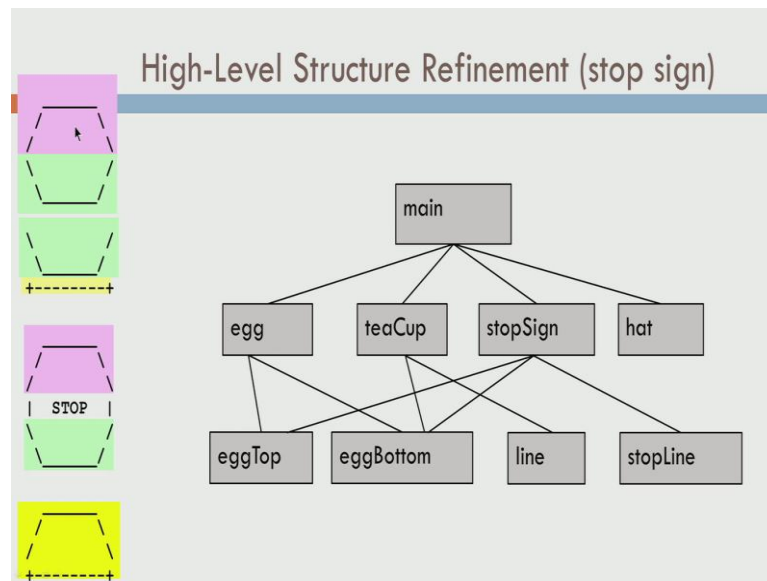
(Refer Slide Time: 07:18)



So, if you want that. So, the tea cup actually can reuse the egg bottom, drawing the egg bottom. So, if you draw the egg bottom and if you draw a line with the pluses on both ends that is actually. So, if you know how to draw bottom of an egg, you can actually draw a tea cup. Only thing you have to do is, draw the bottom of the egg and draw a line underneath. So, tea cup requires, you to know how to draw the bottom of the egg and bottom of the line.

If you have already written that as the function, you can call the same function once more from tea cup, there is no harm. Now, let us go and look at the stop sign, the stop sign is also interesting.

(Refer Slide Time: 08:00)

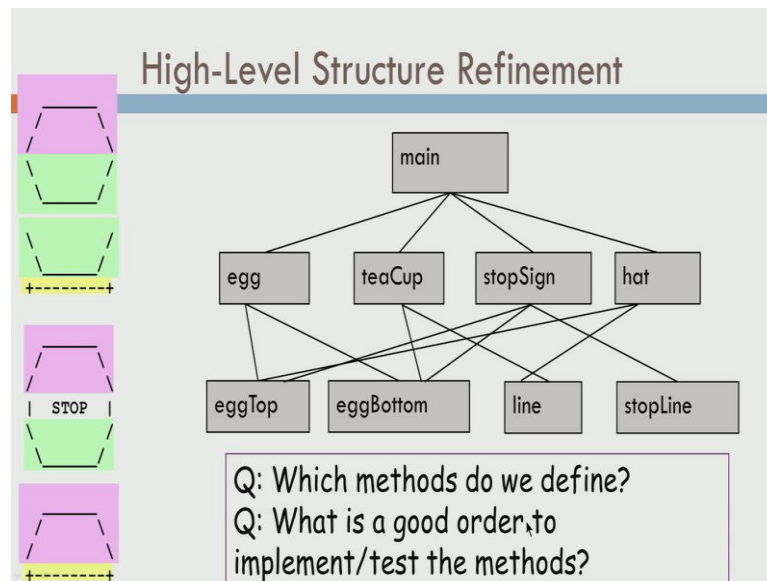


Because, the top half of the stop sign looks like the top of the egg. The bottom top of the stop sign looks like the bottom half the egg and also the top of the cup and that is a line in between which has the stop. So, for drawing a stop sign if you draw an egg top and if you print this line and if you draw egg bottom, you have a stop sign. So, we will call this line, the stop line. So, this is called just a line, this is called a stop line and for drawing a stop sign you need egg top, then you need to draw stop line and we need to do egg bottom, you have that.

Then, let us look at the last one which I called the hat. The hat seems to be the top of the egg and the bottom of the tea cup. So, I already wrote those two things, let us I have functions to do those, I have top of the egg drawn and bottom of the tea cup. So, I have these two. So, I do not have to rewrite code anymore.



(Refer Slide Time: 09:01)



So, we have these two and I already have the methods. So, now if you look at this whole structure, main program requires you to know how to draw an egg, tea cup, stop sign and hat. Because, those are the four functions calls we are doing. Egg in term will call egg top and egg bottom in that order, tea cup will call egg bottom and line in that order, stop sign will call egg top, stop line and egg bottom in that order and hat will call egg top and line in that order.

So, for writing egg all you have to do is write egg top and egg bottom and make two function calls, but somebody ask to still write egg top and egg bottom and line and stop line and so, on. So, now there are two questions one can ask. What are the methods we should be defining and what is a good order in which we have to implement and test this methods? So, clearly main depends on these four and these four depend on these four, it dictates that we have to know how to implement these four.

Without building this, there is no point in building these and without building these, you cannot build your overall diagram which is on the left side, this egg followed by cup followed by stop followed by hat, you cannot draw that unless you have drawn, you know how to draw these four structures.

(Refer Slide Time: 10:21)

## Structured Program version

```
void main() {
    egg();
    teaCup();
    stopSign();
    hat();
}

// Draws the top half of an egg figure.
void eggTop() {
    printf("      ");
    printf(" /_____\n");
    printf("/          \n");
}

// Draws the bottom half of an egg figure.
eggBottom() {
    printf("\\_____/");
    printf("\\_____/");
}

// Draws a complete egg figure.
egg() {
    eggTop();
    eggBottom();
    printf();
}

...

```

So, we will start each one of them. So, main requires of call to egg, tea cup, stop sign and hat. This is just like what we had in the version 2 of the program. Only that we have a version which draws top of the egg and we draw version which draws bottom of the egg. If you have these two, I am ready to draw the egg itself. What does the egg need? You need top of the egg, you need bottom of the egg and we need to draw the egg by calling egg top and egg bottom.

So, forget this printf for a while now. It is actually a mistake it should have a back slash n there. So, printf back slash n. It will draw a line after the egg is drawn. So, if you look at the sequence, main calls egg, egg calls egg top which will draw this, then it calls egg bottom which will draw this and it is supposed to be printing a new line. Again in these cases, the new lines are intentionally left out. So, that it does not clutter the program write now.

(Refer Slide Time: 11:17)

## Revised Version Continued

```
...
// Draws a line of dashes.
void line() {
    printf("+-----+");
}

// Draws a teacup figure.
void teaCup() {
    eggBottom();
    line();
    printf();
}

// Draws a stop sign figure.
void stopSign() {
    eggTop();
    printf("| STOP |");
    eggBottom();
    printf();
}

// Draws a figure that looks sort of like a hat.
void hat() {
    eggTop();
    line();
}
}
```

Then, let us look at how to draw a line. So, you do a plus followed by dashes followed by a plus, this is for our saucer. To draw a tea cup, we need the egg bottom on the line and. So, there will give us the cup and saucer. To draw a stop sign, we need the egg top and egg bottom, but we need a stop line in between. So, this top line is needed only for the stop sign, it is not needed anywhere else. So, instead of calling a function, I could just put the printf statement here itself, because it is not used by more than one function.

So, stop sign is required, this top line is required only for the stop sign, nothing else. Hat, you do not need to do anything new at all, egg top you have already drawn, you know how to draw a line. So, you do these. So, now you all look at the structure, the program is much cleaner. You have reused the code that we have written for egg top and egg bottom in several places. So, egg top is used in drawing an egg and hat, egg bottom is useful in drawing the egg and the tea cup and egg top and egg bottom are useful in stop sign.

So, now this is the much nicely structured and nicely decomposed program. So, for instance if you make one small mistake in the space in the egg top, if you cut and pasted these to this structure multiple times, you have to go and change this multiple times in your program. But, if there is a small mistake in egg top, all you have to do is fix it here, automatically all the functions which call it will have the fixed versions.

(Refer Slide Time: 12:49)

## Practice

- Organize the program into multiple files
- Each file should be a coherent collection of related functions
- One file, e.g., main.c should define main(), which should call the functions defined in the other files.

So, this is the setup. So, even though I said this is what you should do. There is some more things that you have to do. So, first thing is taking a problem and dividing that in to sub problems and writing functions for each one of them, we have already done that. We need to do something more, the thing is we have the. So, called basic shapes. So, I am going to call egg top, egg bottom and line, I am going to call these things as basic shapes.

Using basic shapes, you build the shapes and using the shapes, you draw a complete picture, this is the complete picture. So, these are basic shapes, these are shapes and this is the complete picture and how to we do that. So, I want to be able to separate the concern of each one of them. Drawing the basic shapes and being able to do something there is difference from being able to draw their shapes and being able to draw shapes is different from being able to draw the picture.

So, I am going to do this in a bottom of fashion. So, you go and look at this, we will go and develop this in a bottom fashion. We will try and write these things first, methods for these things and then write methods for these things and then this, but we are also going to do something more which we have not seen. So, far.

(Refer Slide Time: 14:08)

## Header Files

- Code files other than `main.c` should be split into two parts:
  - `code.c`
    - all actual function definitions
    - all function prototypes that are "private," i.e., not used by any other file
  - `code.h`
    - all typedef's and struct's
    - function prototypes for all functions in `code.c` used by other files, e.g., `main.c`

So, the first thing we are going to do is. So, for we have been using the. So, called header file, we are not really developed our own header files. So, we use `stdio.h` and `string.h` and so, on. So, you need to separate the concerns, this is the terms people used in Software Engineering separate concerns. So, one you are mixing the program, what the program is supposed to be doing with what the interface of the functions are.

So, the implementation and the declaration are all in the same file. So, far, but we can separate these concerns. So, use the `code.h` or give a file name `dot h` to do all the prototypes and so, on and write file name `dot c` to write the program. So, the implementation usually goes into `dot c` files and the declaration is going to `dot h` files.

(Refer Slide Time: 15:01)

## Header File Structure

```
#ifndef CODE_NAME
#define CODE_NAME
...structs, enums, prototypes...
#endif
```

- The `#ifndef/#define/#endif` instructions are a standard idiom that makes sure that this file is loaded at most once, no matter how many files `#include` it.

Let us say, we adopt that and we are going to do something with the header files. So, for every problem there is some solving the problem itself and all the functions or other things that you need for it. So, we will see how to do that.

(Refer Slide Time: 15:10)

### Example Header File (basicshapes.h)

```
#ifndef BASIC_SHAPES_H
#define BASIC_SHAPES_H

void eggTop();
void eggBottom();
void line();
#endif
```

So, the header files usually have this format. So, there is hash if not def of code name, then define code name, put all your structures and prototypes and so, on, end if. So, let us not very about this if not def defined in end if, but all the declarations go with in these three structures, hash if not def, hash define, hash end if. So, what it tells you is, if something called code name is not defined, define it now, define all these things and end it.

So, let see how this look likes for basic shapes. I am creating a file called basic shapes dot h, this takes care of only the declarations for all the functions which have for basic shapes, egg top, egg bottom and line are three functions. So, add the declarations for these three functions in a file called basic shapes and. So, whenever I use a file name, I will use the same thing for the if not def. If not def basic shapes h define basic shape h.

If this has been not defined. So, for, define this variable now, define all these function prototypes, end it. So, this is the declaration for the basic shapes.

(Refer Slide Time: 16:24)

## Using Your Header (.h) Files

- Use `#include "..."` to include any header files you defined
  - Example: `#include "basicshapes.h"`
- Put your header files in the same folder as your C files
- Use `#include <...>` to include standard C library header files
  - Example: `#include <stdlib.h>`

The description. So, will come later. So, if you want to use the dot h file, you do hash include basic shape dot. This is like hash include stdio dot h. So, put your header file in the same folder as your C files and use hash include, for example, in stdlib. So, I will say you how to write basic shapes dot c.

(Refer Slide Time: 16:42)

## basicshapes.c

```
#include "basicshapes.h"
#include <stdio.h>
// Draws the top half of an an egg figure.
void eggTop() {
    printf("          ");    printf("\n");
    printf(" /_____\");    printf("\n");
    printf("/          \");    printf("\n");
}
// Draws the bottom half of an egg figure.
void eggBottom() {
    printf("\          /");    printf("\n");
    printf(" \   ___/");    printf("\n");
}
// Draws a line of dashes.
void line() {
    printf("+-----+");    printf("\n");
}
```

So, basic shapes dot c will be a C program, this has the implementation. This include stdio and also include basic shape and it has the function descriptions for egg top, egg bottom and line. So, basic shapes dot h and basic shapes dot c, you have something that can draw basic shapes and nothing else.

(Refer Slide Time: 17:07)

## Using Your Code (.c) Files

- Put your code (.c) files in the same directory as the other files in your project.
- Use your IDE's "add to project" command to add the code files to your project.
- Do **not** include code files, i.e., do not write `#include "basicshapes.c"` in any file.

Now, you want to be able to draw shapes with it. So, let us move up.

(Refer Slide Time: 17:13)

## shapes.h

```
#ifndef SHAPES_H
#define SHAPES_H

#include "basicshapes.h"
void egg();
void teaCup();
void stopSign();
void hat();
#endif
```

I am going to write a file called shapes dot h, which has all the things required for drawing the shapes. So, the declarations for drawing the shapes and this is the description of those functions.



(Refer Slide Time: 17:23)

```
shapes.c
52
#include <stdio.h>
#include "shapes.h"
// Draws the top half of an an egg figure.
void egg() {
    eggTop();
    eggBottom();
    printf("\n");
}
// Draws a teacup figure.
void teaCup() {
    eggBottom();
    line();
    printf("\n");
}
```

So, shapes dot c has shapes dot h, because it needs those descriptions and how is the egg implemented. The implementation is in this file, egg top, egg bottom and printf, tea cup egg bottom, line and printf and so, on.

(Refer Slide Time: 17:43)

```
shapes.c (contd.)
53
// Draws a stop sign figure.
void stopSign() {
    eggTop();
    printf("| STOP |");
    eggBottom();
    printf("\n");
}
// Draws a figure that looks sort of like a hat.
void hat() {
    eggTop();
    line();    printf("\n");
}
```

This is what you need for drawing the shapes. So, now we have four files basic shapes dot h, basic shapes dot c, shapes dot h and shapes dot c. So, shapes dot h has only the declarations, it also includes basic shape dot h and so, on. So, now we have four files, you cannot compile just one file now, you have to create what is called a project. I will show you, how the project is done in the demo.

(Refer Slide Time: 18:10)

```
drawFigures.c
54
#include "shapes.h"

void main() {
    egg();
    teaCup();
    stopSign();
    hat();
}
```

Finally, the top most thing I am going to call the draw figures dot c, draw figures needs shapes dot h and. So, now, egg, tea cup, stop sign, hat are all provided by shapes dot c. Shapes dot c in turn depends on basics shapes dot c. So, this is the overall structure, we have five files, draw figures, basic shapes dot h in c and shapes h in c. So, I will already return this here. So, you can see basic shapes dot h, it has a declarations, basic shapes dot c which has the description, shapes dot h which has the declarations, shapes dot c which has the description and draw shapes dot c which has the description for the main program.

I have included all of these into our project called draw shapes. So, you can do that by saving file new project. So, I did file new project I created a file by name, draw shapes. I included these five files into the project. Now, I can compile the project and what it does is it takes all these files and compile them together and when I run it, it actually runs the main program. So, if you noticed the individual once do not have main routines.

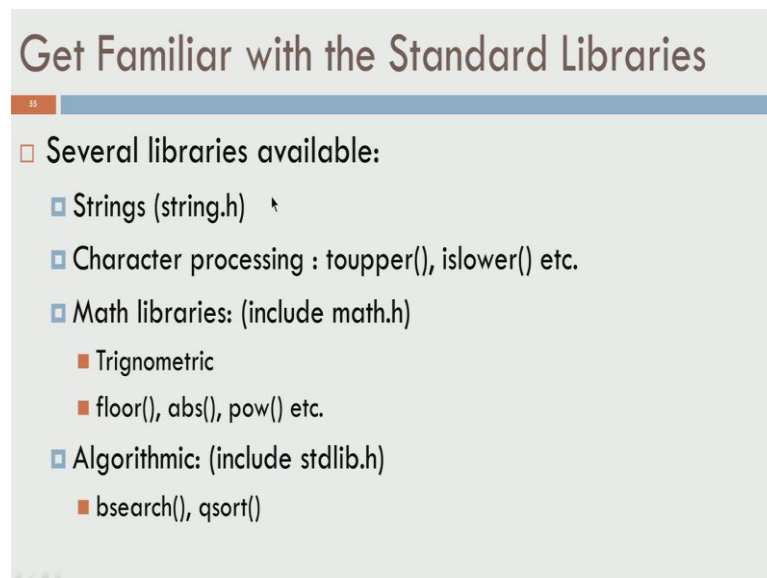
We do not have main routines in any of these. The main routine is only in this one. So, for drawing shapes dot c and basic shapes dot c and shapes dot c, they are also compiled and they are compiled along with draw shapes dot c, you can see the result of the print out here. So, you can see the egg, the tea cup, the stop sign and this that came from actually having taken the problem and dividing that into sub problems, identifying the structure which is common and being able to reuse and so, on.

So, for whatever reason if I two cups tomorrow, I can actually call the cups twice. So, I

can call tea cup and a tea cup, it will draw two tea cups one below the other. So, this idea of being able to taking a problem, dividing that into sub problems is a necessary skill. You probably did that with functions itself, but separating the concerns of implementation from the. So, called interface or just the declaration is the another thing.

So, once you have something if you divide that into multiple files, you can quickly go and see which file has the basic issue and which file has the problem and so, on. So, for example, tomorrow I may not want to draw egg and stop sign and so, on, but I want to draw two cups and three cups, I can write a different program which will take the cups and draw them explicitly. So, this notion of dividing the problem is decomposition and organizing them into multiple header files in C files is a very necessary skill, when you build large and large pieces of software.

(Refer Slide Time: 20:58)



So, I want to end this course with one small piece of advice. So, there are lots of things that we did in the course, but we never wrote or explicitly use libraries for many of them. C comes with a lot of libraries. These are standard libraries that come with any standard C package. So, your compiler package these things for you, thinks to operate on the strings, things to operate on characters. For example, you take a lower character and convert that to upper case, you can check whether a character is lower case or upper case, things like that.

There are math libraries included from math dot h, trigonometric functions, floor, absolute value, power and things like that are all available from math libraries. It is not

just that these are the functions that are there. Even some algorithmic things are available. For example, from `stdlib` you learn binary search, you do not have to write library search yourself, you can call the binary search function available from `stdlib`.

So, you learnt how to do various sorting algorithms. So, `qsort` for instance is an algorithm which has something called quick sort. So, you can call `qsort` on an array of integers for instance. So, you have to learn how to do it though, I am simplifying a few things here, but there are sorting, searching functions, mathematical libraries and the other things that are already available out there. If it is available as a part of your package, learn how to use it.

So, at some point all we have to do is make appropriate function calls, include appropriate header files, make appropriate function calls and compile with appropriate `flux`. So, I am simplifying a lot of things, but learn how to use the basic libraries and not write things from scratch, every time you write a program. So, this brings us to the end of this module and. In fact, to the end of this course and I hope you enjoyed the course as much as we enjoyed creating it and this has been probably a rough and tuff, 9 or 10 weeks that you have been through with lots of programming assignments and assessments and so, on.

But, we hope that this course helped you, in some way or the other to learn either from scratch or at least be able to if we have already done C programming, at least cleared a lot of concepts for you and taught a few new things like data structures and algorithms and so, on. So, for doing this course we took help from quite a few of you people. So, of course, we were the faces that you saw, but there is a big team that was behind it.

We have 10 TAS who are all behind this. So, they were actually responding to you on the forums. There is a team from Google which created the platform and there is also this local team, the NPTEL team in charge of video recording scheduling and. So, many other things. So, there are probably 30 or 40 people who got involved in creating the contents for this course and delivering it to you. So, I personally want to place on record that we three thank each and every one of them.

So, we cannot possibly list all of them by name, but I want to thank all of them to make this course, the success it is now. And I am hoping that you guys will tune in later for other courses that NPTEL has to offer. So, all the best for your exams and I hope to see you in some other course at a later point of time.

Bye bye.