## **7. Structuring Programs Using GLUT**

From Chapter 7 and below, check the folder below “code/” to find the C source files and associated HTMLs that show the modified part from their original source files.

GLUT library recommends event-driven programming to programmers.

Here, let's structure our programming accordingly.

After that, we will finally introduce various drawing-related functions according to the OpenGL programming style.

This section is explained on the assumption that eclipse is used.

After the middle stage, some source files will remain unchanged. When creating a new project, we copy the source files that have not changed from the original projects (files can be shared between projects, but in this class, we will not do so. Just copy them).

**Reference link**

·[OpenGL functions (functions that begin with the two-letter gl lowercase)](http://www.opengl.org/sdk/docs/man/)

·[OpenGLProgramming Guide Book (ver1.1, commonly known as red](http://glprogramming.com/red/index.html) book)

[·GLUT - The OpenGL Utility Toolkit](http://www.opengl.org/resources/libraries/glut/)

[·](http://www.opengl.org/resources/libraries/glut/)[Toolkit (GLUT) Programming Interface API Version 3](http://www.opengl.org/resources/libraries/glut/spec3/spec3.html)

### **07.01. Structural design of the program**

06-09-LogoOpenGL.c was still a single file. We divide it to prepare for future scale-up.

Before you split it, you have to think about which unit to be segmented.

How many groups of functions should we consider in 06-09-LogoOpenGL.c?

(Eclipse has a pane where you can list the functions.)

1. Header
2. CG Object creation
3. Camera setting
4. One frame drawing
5. Callback
6. OpenGL initialization
7. Main function

The above is an example of grouping.

A rewritten program based on this example would look like this:

[Program 07-01]

1. 07-01-Planning.c (Diff from 06-09)

**Exercise**

**07-01-ex1**: Function grouping

In the above configuration, list the functions that are likely to belong to the same group.

**07-01-ex2**: Definition of groups

Explain why each group is organized in the above configuration.

**07-01-ex3**: Source code differences

Is 07-01-Planning.c functionally different from 06-09-LogoOpenGL.c? If different, explain that part.

**07-01-ex4**: Display of function grouping

Please devise the display of Eclipse so that the program structure of 07-01-Planning.c is easy to understand.

### **07.02. File division**

We have already learned file division in section 04, but this time it is the actual programming with the divided files.

There are some correct answers below, but before you see them, do all the work by yourself.

The task is simple: Just split 07-01-Planning.c into multiple files.

Divide the original source file into 6 source files.

Wmm.

Mmm.

Yes?

The following is the result of splitting.

Don't look inside until you're done!

[Program 07-02]

1. 07-02-CommonHeaders.h (Diff from 07-01)
2. 07-02-Callback.c (Diff from 07-01)
3. 07-02-EmbededObjects.c (Diff from 07-01)
4. 07- 02-Initialization.c (Diff from 07-01)
5. 07-02-Projection.c (Diff from 07-01)
6. 07-02-Rendering.c (Diff from 07-01)07-01)
7. 07-02-MainFunction.c (Diff from07-01)

**Exercise**

**07-02-ex1**: Correspondence of functions in files

Describe which group in section 07-01 corresponds to which source file.

**07-02-ex2**: Reason for compilation error

The division was completed by the work in this section, but compilation does not pass in this state. Explain three reasons why the compilation doesn't work.

**07-02-ex3**: Compile error display

The division is completed by the work in this section, but compilation does not pass in this state. Check all the Eclipse error reports that show why the compilation doesn't work.

### **07.03. Correction to file splitting**

The work in section 07-02 is just splitting, so, naturally, the compilation does not pass.

Make the following corrections. This will bring you back to a state where you can make an executable file.

* Introduction of a common header
* function prototype declaration in the common header
* Setting link option

Since you learn all the necessary information and knowledge, do the above three elements by yourself (without looking at the source files below).

After making it all by yourself, see for yourself the difference in writing style from the following.

Don't look inside until you've done it!

[Program 07-03]

1. ic2-CommonHeaders.h (Diff from 07-02)
2. 07-03-Callback.c (Diff from 07-02)
3. 07-03-EmbededObjects.c (Diff from 07-02)
4. 07-03- Initialization.c (Diff from 07-02)
5. 07-03-Projection.c (Diff from 07-02)
6. 07-03-Rendering.c (Diff from 07-02)
7. 07-03-MainFunction.c (07-02)

**Exercise**

**07-03-ex1**: Why variable names are not needed

In function prototype declarations such as 07-03-CommonHeaders.h, they give function argument names. According to the C language specifications, the argument name is not required as long as the argument type is indicated. Explain why variable names are not needed due to the C compiler specifications.

**07-03-ex2**: Meaning of prototype declaration

Using theic2\_BootWindow() function as an example, explain how the prototype declaration works between 07-03-MainFunction.c and 07-03-Initialization.c at compile time. Explain from the perspective of the compiler.

**07-03-ex3**: Breakpoints in multiple places

Set breakpoints at the beginning of each function block of main() function, ic2\_DrawFrame() function, and ic2\_OpenGLLogo() function, and do not step to the breakpoint of ic2\_OpenGLLogo()..

(Difficult to try in the environment of 2020 due to eclipse)

### **07.04. Addition of event (keyboard)**

The GLUT library provides a function (callback registration function) that can register callbacks for some events.

[Callback registration](http://www.opengl.org/resources/libraries/glut/spec3/node45.html#SECTION00080000000000000000)

Here, let's make the program terminate when the 'q' key is pressed.

The event is that "the 'q' key was pressed."

The callback is "terminating the program".

First, let's design the operation after being called back.

Here, the behavior after being called back is "end the program".

There is an exit() function as a function to forcibly terminate the program from any point.

(How do you know the existence of the exit() function? It's the same as learning a new verb in a natural language, and if you need it, you have to ask someone who knows it.)

1. What is the grammar of the exit() function?
2. What is the function of the exit() function?  
   (Where can I find the official information?)  
   (The exit() function is a C library.)  
   (That should be on the man page.)

The next thing to think about is pairing events and callbacks.

To do this, we will gather the information we need to register the events and callbacks.

1. What kind of event is issued on the act of pressing the 'q' key?
2. What is the glut library function that can register a callback function for that event?
3. Arguments are specified for the callback functions that can be registered. What are their numbers and types, and what do they mean?

With this kind of information, we can make changes to your program.

Which part of the program should we modify?

At first, put a mark on the comment line where you think it will be there.

Then look at the example below to see if your expectations were correct.

Don't look inside until you're done!

[Program 07-04] (Unpublished part is unchanged)

1. ic2-CommonHeaders.h (Diff from 07-03)
2. 07-04-Callback.c (Diff from 07-03)
3. 07-04-Initialization.c (Diff from07-03)

See modified part of 07-04-Callback.c to check the function that summarizes the operations in call-back.

See modified part of 07-04-Initialization.c to check where the event and callback are registered as a pair.

Callbacks for events where a key is pressed can be registered with the glutKeyboardFunc() function.

This part only needs to be registered.

Actually, event detection starts after the glutMainLoop() function is activated.

(Where is the glutMainLoop() function in the program source?)

**Exercise**

**07-04-ex1**: Event types

Examine the events that can be handled by the GLUT library and list them.

**07-04-ex2**: Function name

Callback registration function takes a "function" as its argument. Explain why this is possible as a C language concept.

**07-04-ex3**: Arguments at the time of callback

The number and types of arguments of the functions that can be registered with the callback registration function are fixed. Explain why they have to do this as a C language concept.

**07-04-ex4**: Keyboard function

Explain the argument of the ic2\_NormalKeyInput() function, with paying attention to the glutKeyBoardFunc() function as well (specify the source of the material. In the case of the material on the Internet, there are two sources. To show).

**07-04-ex5**: Checking the arguments of the keyboard function

Actually check the arguments of the ic2\_NormalKeyInput() function using the breakpoint function of eclipse.

(Difficult to implement in the environment of 2020 due to eclipse)

### **07.05. Add event (enlarge / reduce logo)**

As a practice, if 's' key is pressed, the logo will be made small, and large if 'S' key is pressed. Let's create such a program.

The first is the design of the operation after being called back.

We prepare a user-defined function ic2\_OpenGLLogo() and describe its operation in it.

The size of the logo is controlled by the argument s of ic2\_OpenGLLogo(float s).

Set it to 0.5 for 's' and 0.95 for 'S'.

Be aware that we just specify the size of the logo.

In other words, we don't “draw” the logo here.

(The actual drawing of CG is continuously requested by the glutPostRedisplay() function. See 06.04.)

(Of course, you can even draw here, but one of the basic programming policies is the policy of "same work in the same part." The program will be smarter if you follow it.)

Set this value where you are calling ic2\_OpenGLLogo(float s).

Since the place where ic2\_OpenGLLogo(float s) is called and the place where the key event of 's' / 'S' can be picked up are different, prepare a global variable logoscale so that the value can be passed.

(Unfortunately, there aren't many quick ways to pass a value during this time other than a global variable.)

When a callback occurs, just set the logoscale to a value of 0.5 for 's' and 0.95 for 'S'.

It is necessary to modify each part as the global variable is manipulated.

[Program 07-05] (Unpublished part is unchanged)

1. ic2-CommonHeaders.h (Diff from 07-04)
2. 07-05-Callback.c (Diff from 07-04)
3. 07-05-Rendering.c (Diff from) 07-03)
4. 07-05-MainFunction.c (Diff from 07-03)

**Exercise**

**07-05-ex1**: Logo color change

Make a program that changes the logo color (any part is acceptable) by 'c' and back by 'C' key.

**07-05-ex2**: Logo shape change

Make a program that geometrically transforms the logo shape (any part is acceptable) by 'd’/ 'D'. You transform it with 'd' and restore it to its original shape with 'D'.

**07-05-ex3**: Logo shape change (toggle)

Based on the program of ex2 above, modify it to a program that transforms only with 'd' (transforms for the first time and restores the original shape for the second time).

**07-05-ex4**: Confirm key event

Confirm that the ic2\_NormalKeyInput() function is called by the input event from the keyboard.