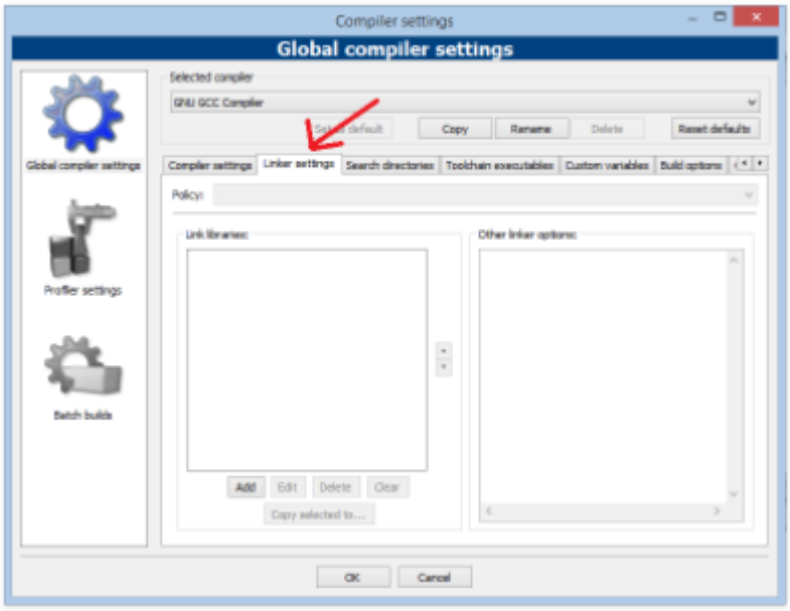
graphics.h in CodeBlocks

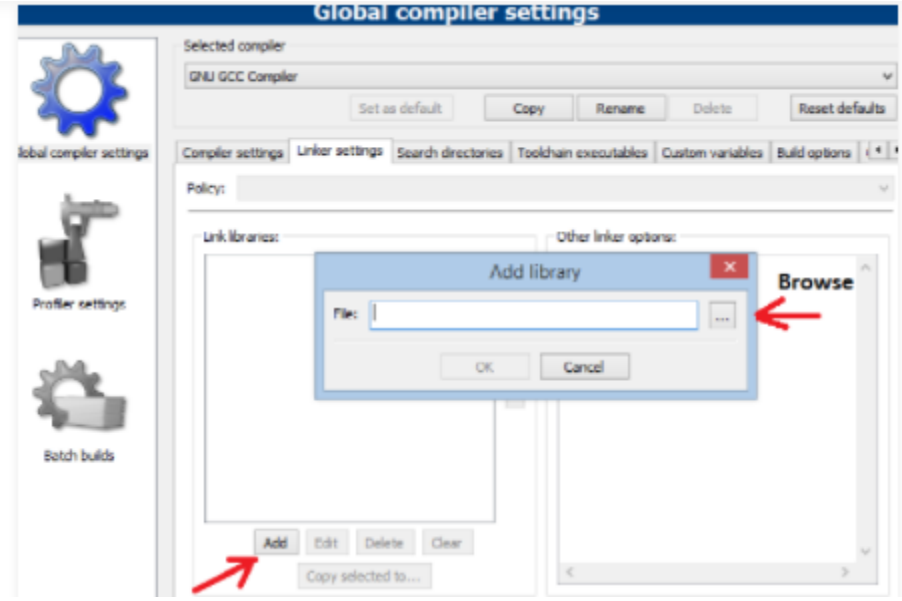
Step 1 : To setup “graphics.h” in CodeBlocks, first set up winBGIm graphics library. Download WinBGIm from http://winbgim.codecutter.org/ or use this link.
Step 2 : Extract the downloaded file. There will be three files:

- graphics.h
- winbgim.h
- libbgi.a

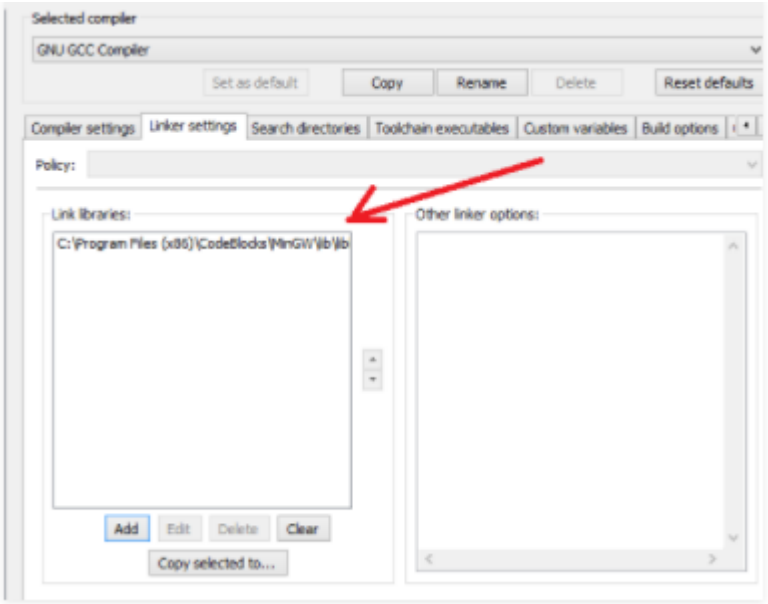
Step 3 : Copy and paste **graphics.h** and **winbgim.h** files into the **include** folder of compiler directory. (If you have Code::Blocks installed in C drive of your computer, go through: Disk C >> Program Files >> CodeBlocks >> MinGW >> include. Paste these two files there.)
Step 4 : Copy and paste **libbgi.a** to the **lib** folder of compiler directory.
Step 5 : Open Code::Blocks. Go to Settings >> Compiler >> Linker settings.



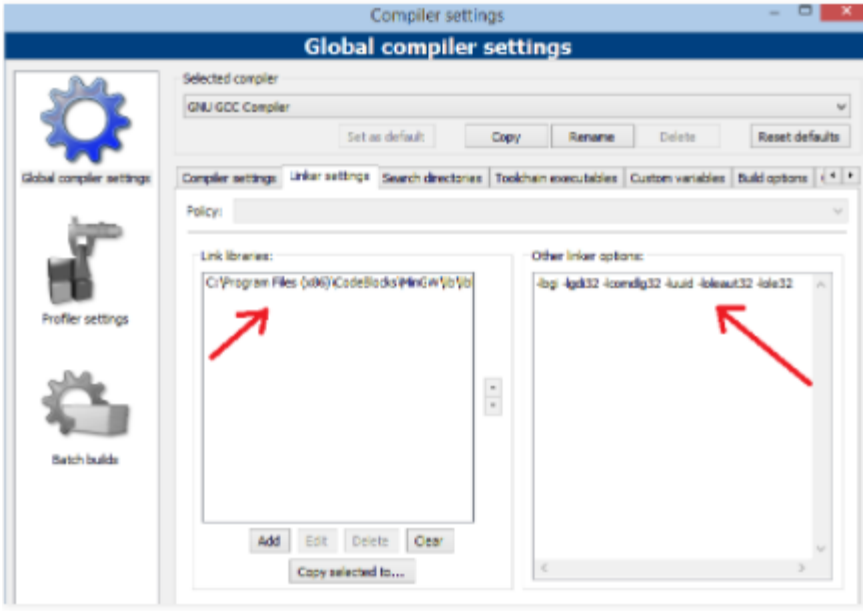
Step 6 : In that window, click the Add button under the “Link libraries” part, and browse.



Select the **libbgi.a** file copied to the lib folder in step 4.



Step 7 : In right part (ie. other linker options) paste commands
-lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32



Step 8 : Click Ok

Step 9 : Try compiling a graphics.h program in C or C++, still there will be an error. To solve it, open graphics.h file (pasted in include folder in step 3) with Notepad++. Go to **line number 302**, and replace that line with this line : **int left=0, int top=0, int right=INT_MAX, int bottom=INT_MAX,**

```
// Image Functions (drawing.cpp)
unsigned imagesize( int left, int top, int right, int bottom );
void getimage( int left, int top, int right, int bottom, void *bitmap );
void putimage( int left, int top, void *bitmap, int op );
void printimage(
    const char* title=NULL,
    double width_inches=7, double border_left_inches=0.75, double border_top_inches=0.75,
    int left=0, int top=0, int right=INT_MAX, int bottom=INT_MAX,
    bool active=true, HWND hwnd=NULL
    );
```

Step 10 : Save the file. Done !

Note : Now, you can compile any C or C++ program containing graphics.h header file. If you compile C codes, you'll still get an error saying: **"fatal error: sstream : no such file directory".**

For this issue, change your file extension to .cpp if it is .c

OpenGL in GRAPHICS

OpenGL provides a powerful but primitive set of rendering command, and all higher-level drawing must be done in terms of these commands. There are several libraries that allow you to simplify your programming tasks, including the following:

- ❏ **OpenGL Utility Library (GLU)** contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections and rendering surfaces.
- ❏ **OpenGL Utility Toolkit (GLUT)** is a window-system-independent toolkit, written by Mark Kilgard, to hide the complexities of differing window APIs.

We are using the OpenGL Utility Toolkit (GLUT) for our rest of the experiments or laboratory work.

Setting up Compilers

IDE: Codeblocks
Version: Latest 17.12 or above

Now,

- ❏ Download “glut-3.7.6-bin.zip” from google drive.
- ❏ Goto _____/MinGW/lib folder and put glut32.lib,glut.def there.
- ❏ Goto _____/MinGW/ include/GL/ and put glut.h there.
- ❏ Goto _____/MinGW/bin and put glut.dll , glut.def there.

We are done with the configuration.

My First OpenGL Program:

```
#include <windows.h>  // for MS Windows
#include <GL/glut.h>  // GLUT, include glu.h and gl.h

/* Handler for window-repaint event. Call back when the window first
appears and
   whenever the window needs to be re-painted. */

void display() {
}

/* Main function: GLUT runs as a console application starting at main()  */
int main(int argc, char** argv) {

   glutInit(&argc, argv);                 // Initialize GLUT
   glutCreateWindow("Hello OpenGL"); // Create a window with the given
title
   glutDisplayFunc(display); // Register display callback handler for
window re-paint
   glutMainLoop();           // Enter the event-processing loop
   return 0;
}
```

glutInit(&argc, argv); This function initializes the toolkit. Its arguments are the standard ones for passing command line information; we will make no use of them here.

glutCreateWindow("Hello OpenGL"); This function actually opens and displays the screen window, putting the title “my first attempt” in the title bar

Now, we can modify the above program to display the window in different dimension.

```
int main(int argc, char** argv) {

   glutInit(&argc, argv);                 // Initialize GLUT
   glutInitWindowSize(800, 600);   // Set the window's initial width &
height
   glutInitWindowPosition(100, 150); // Position the window's initial top-
left corner
   glutCreateWindow("Hello OpenGL"); // Create a window with the given
title
```

```
   glutDisplayFunc(display); // Register display callback handler for
window re-paint
   glutMainLoop();           // Enter the event-processing loop
   return 0;
}
```

glutInitWindowSize(800,600); This function specifies that the screen window should initially be 800 pixels wide by 480 pixels high. When the program is running the user can resize this window as desired.

glutInitWindowPosition(100, 150); This function specifies that the window's upper left corner should be positioned on the screen 100 pixels from the left edge and 150 pixels down from the top. When the program is running the user can move this window wherever desired.

glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); This function specifies how the display should be initialized. The built-in constants GLUT_SINGLE and GLUT_RGB, which are OR'd together, indicate that a single display buffer should be allocated and that colors are specified using desired amounts of red, green, and blue. (Later we will alter these arguments: for example, we will use double buffering for smooth animation.)