Subject: Re: Some Experiment with Size of Upp Executable
Posted by Lance on Mon, 27 Dec 2021 04:28:01 GMT
View Forum Message <> Reply to Message

Obviously *.t(s) don't contribute too much to the size of the final executables of empty U++ projects.

Question: who contribute(s) most? In a CtrlLib, we expect all *.iml files do a part. What about the plain Core console application?

Anyway, some more relevant/irrelevant tests.
Test 3: How well compilers optimize out unused code.
A common nonsense function used by both scenarios which will potentially increase the sizes of final executables by noticeable amounts.

```
int BigFunction(int i)
{
 static const char * s[]={R"(
#include "CtrlCore.h"

namespace Upp {

#define LTIMING(x)

ImageBuffer::ImageBuffer(ImageDraw& iw)
{
 Image m = iw;
 Set(m);
}

// more omitted. basically I take a file, paste it
// multiple times, encode each in R"( )", and create an
// const char* array[].
)",R"(....)"};

 int sum=0;
 for(int j=0; j<(int)strlen(s[i]); ++j)
  sum+=s[i][j];
 return sum;
}
```

Senario 1

```
#include <Core/Core.h>

using namespace Upp;

int BigFunction(int i);

CONSOLE_APP_MAIN
{
```

```
 RLOG(BigFunction(Upp::Random()%7));
}
```
Senario 2: With reference to BigFunction commented out.
```
#include <Core/Core.h>

using namespace Upp;

int BigFunction(int i);

CONSOLE_APP_MAIN
{
// RLOG(BigFunction(Upp::Random()%7));
}
```
Senario 3
In senario 3, I changed the BigFunction so that the const char * s[]; is move out of the function body, with or without the static modifier. Senario 3 is otherwise same as Senario 1, with invocation of BigFunction commented out.

Test result
```
   NoBigFun  Sen.2   Sen.1   Sen.3
// MSBT22x64 Release  767488  767488  770560  769024
// MSBT22 Release  644608  644608  648192  646144
// CLANGx64 Release 1725952 1727488 1728512 1727488
// CLANG  Release 1849856 1852928 1853440 1852416
```

Conclusion: In this test, CLANG performed very poorly. However, MS Build Tools (almost) did exactly what's expected: simply including the definition of a function without invoking it should have no impact on the sizes of final executables. As long as one compiler can do that, we expect others (eg. CLANG) to catch up in the future.

And It makes sense to encapsulate big objects in functions.

Out of random search of U++ code, I noticed that mirek actually did very well on this. Some other code could potentially benefit from this kind of audit:
```
#include "SDL2GL.h"

namespace Upp {

const static VectorMap<dword, dword> SDL_key_map = {
// { SDLK_BACKSPACE, K_BACK        },
 { SDLK_BACKSPACE, K_BACKSPACE   },
 { SDLK_TAB,       K_TAB         },
 { SDLK_SPACE,     K_SPACE       },
 { SDLK_RETURN,    K_RETURN      },

 { SDLK_LSHIFT,   K_SHIFT_KEY    },
 { SDLK_LCTRL,    K_CTRL_KEY     },
```

```
{ SDLK_LALT,     K_ALT_KEY      },
{ SDLK_CAPSLOCK, K_CAPSLOCK     },
{ SDLK_ESCAPE,   K_ESCAPE       },
{ SDLK_PAGEUP,   K_PAGEUP       },
{ SDLK_PAGEDOWN, K_PAGEDOWN     },
{ SDLK_END,      K_END          },
{ SDLK_HOME,     K_HOME         },
{ SDLK_LEFT,     K_LEFT         },
{ SDLK_UP,       K_UP           },
{ SDLK_RIGHT,    K_RIGHT        },
{ SDLK_DOWN,     K_DOWN         },
{ SDLK_INSERT,   K_INSERT       },
{ SDLK_DELETE,   K_DELETE       },

{ SDLK_KP_0, K_NUMPAD0 },
{ SDLK_KP_1, K_NUMPAD1 },
{ SDLK_KP_2, K_NUMPAD2 },
{ SDLK_KP_3, K_NUMPAD3 },
{ SDLK_KP_4, K_NUMPAD4 },
{ SDLK_KP_5, K_NUMPAD5 },
{ SDLK_KP_6, K_NUMPAD6 },
{ SDLK_KP_7, K_NUMPAD7 },
{ SDLK_KP_8, K_NUMPAD8 },
{ SDLK_KP_9, K_NUMPAD9 },
{ SDLK_KP_MULTIPLY, K_MULTIPLY  },
{ SDLK_KP_PLUS,     K_ADD       },
{ SDLK_KP_PERIOD,   K_SEPARATOR },
{ SDLK_KP_MINUS,    K_SUBTRACT  },
{ SDLK_KP_PERIOD,   K_DECIMAL   },
{ SDLK_KP_DIVIDE,   K_DIVIDE    },
{ SDLK_SCROLLLOCK,  K_SCROLL    },
{ SDLK_KP_ENTER,    K_ENTER     },

{ SDLK_F1,  K_F1  },
{ SDLK_F2,  K_F2  },
{ SDLK_F3,  K_F3  },
{ SDLK_F4,  K_F4  },
{ SDLK_F5,  K_F5  },
{ SDLK_F6,  K_F6  },
{ SDLK_F7,  K_F7  },
{ SDLK_F8,  K_F8  },
{ SDLK_F9,  K_F9  },
{ SDLK_F10, K_F10 },
{ SDLK_F11, K_F11 },
{ SDLK_F12, K_F12 },

{ SDLK_a, K_A },
{ SDLK_b, K_B },
```

```
{ SDLK_c, K_C },
{ SDLK_d, K_D },
{ SDLK_e, K_E },
{ SDLK_f, K_F },
{ SDLK_g, K_G },
{ SDLK_h, K_H },
{ SDLK_i, K_I },
{ SDLK_j, K_J },
{ SDLK_k, K_K },
{ SDLK_l, K_L },
{ SDLK_m, K_M },
{ SDLK_n, K_N },
{ SDLK_o, K_O },
{ SDLK_p, K_P },
{ SDLK_q, K_Q },
{ SDLK_r, K_R },
{ SDLK_s, K_S },
{ SDLK_t, K_T },
{ SDLK_u, K_U },
{ SDLK_v, K_V },
{ SDLK_w, K_W },
{ SDLK_x, K_X },
{ SDLK_y, K_Y },
{ SDLK_z, K_Z },
{ SDLK_0, K_0 },
{ SDLK_1, K_1 },
{ SDLK_2, K_2 },
{ SDLK_3, K_3 },
{ SDLK_4, K_4 },
{ SDLK_5, K_5 },
{ SDLK_6, K_6 },
{ SDLK_7, K_7 },
{ SDLK_8, K_8 },
{ SDLK_9, K_9 },

{ K_CTRL|219,  K_CTRL_LBRACKET   },
{ K_CTRL|221,  K_CTRL_RBRACKET   },
{ K_CTRL|0xbd, K_CTRL_MINUS      },
{ K_CTRL|0xc0, K_CTRL_GRAVE      },
{ K_CTRL|0xbf, K_CTRL_SLASH      },
{ K_CTRL|0xdc, K_CTRL_BACKSLASH  },
{ K_CTRL|0xbc, K_CTRL_COMMA      },
{ K_CTRL|0xbe, K_CTRL_PERIOD     },
{ K_CTRL|0xbe, K_CTRL_SEMICOLON  },
{ K_CTRL|0xbb, K_CTRL_EQUAL      },
{ K_CTRL|0xde, K_CTRL_APOSTROPHE },

{ SDLK_PAUSE, K_BREAK }, // Is it really?
```

```
 { SDLK_PLUS,      K_PLUS      },
 { SDLK_MINUS,     K_MINUS     },
 { SDLK_COMMA,     K_COMMA     },
 { SDLK_PERIOD,    K_PERIOD    },
 { SDLK_SEMICOLON, K_SEMICOLON },

 { SDLK_SLASH,       K_SLASH     },
 { SDLK_CARET,       K_GRAVE     },
 { SDLK_LEFTBRACKET,  K_LBRACKET  },
 { SDLK_BACKSLASH,    K_BACKSLASH },
 { SDLK_RIGHTBRACKET, K_RBRACKET  },
 { SDLK_QUOTEDBL,     K_QUOTEDBL  }
};
```

Majority of the occurrences seem to be from plugin(wrapping of c source libraries). These are quite difficult to fix without actually touch the imported source codes. Ideally a parser should be created to automate the job if it's decided to be of significant savings.
