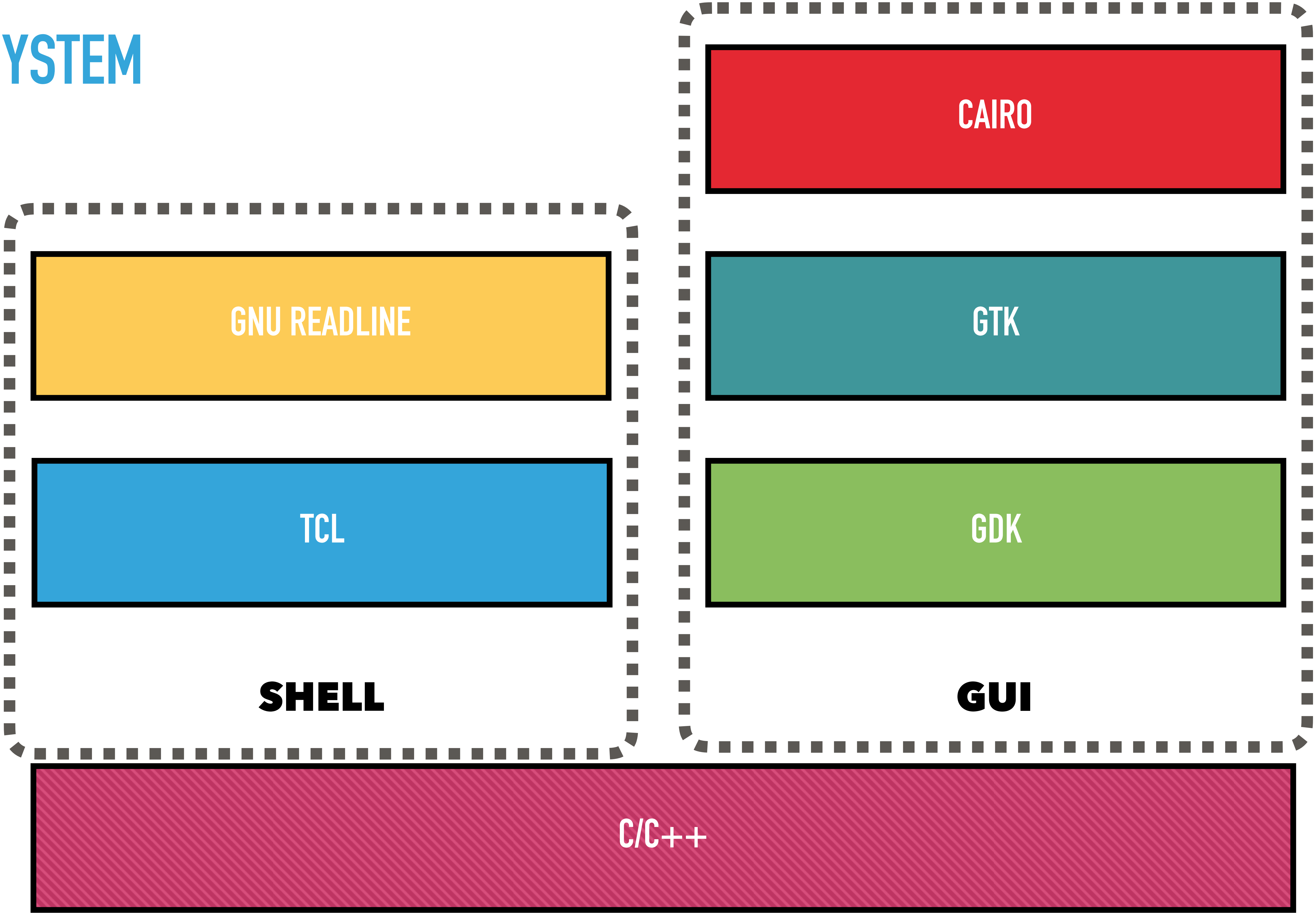
PANAGIOTIS SKRIMPONIS

CHRISTOS SOTIRIOU

UNIVERSITY OF THESSALY

PHYSICAL CAD

SYSTEM



CAIRO

GTK

GDK

GUI

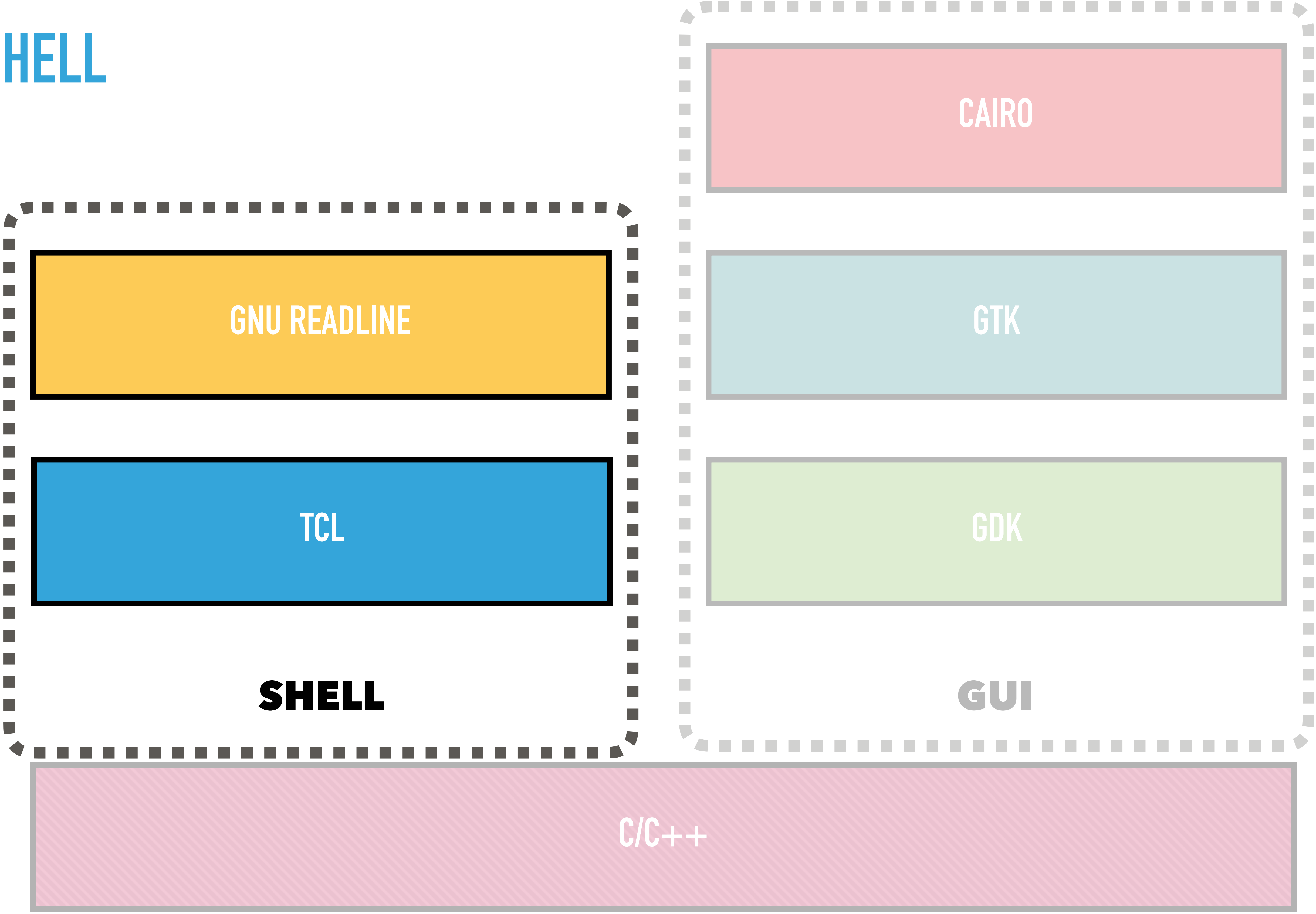
GNU READLINE

TCL

SHELL

C/C++

SHELL



GNU READLINE

TCL

SHELL

CAIRO

GTK

GDK

GUI

C/C++

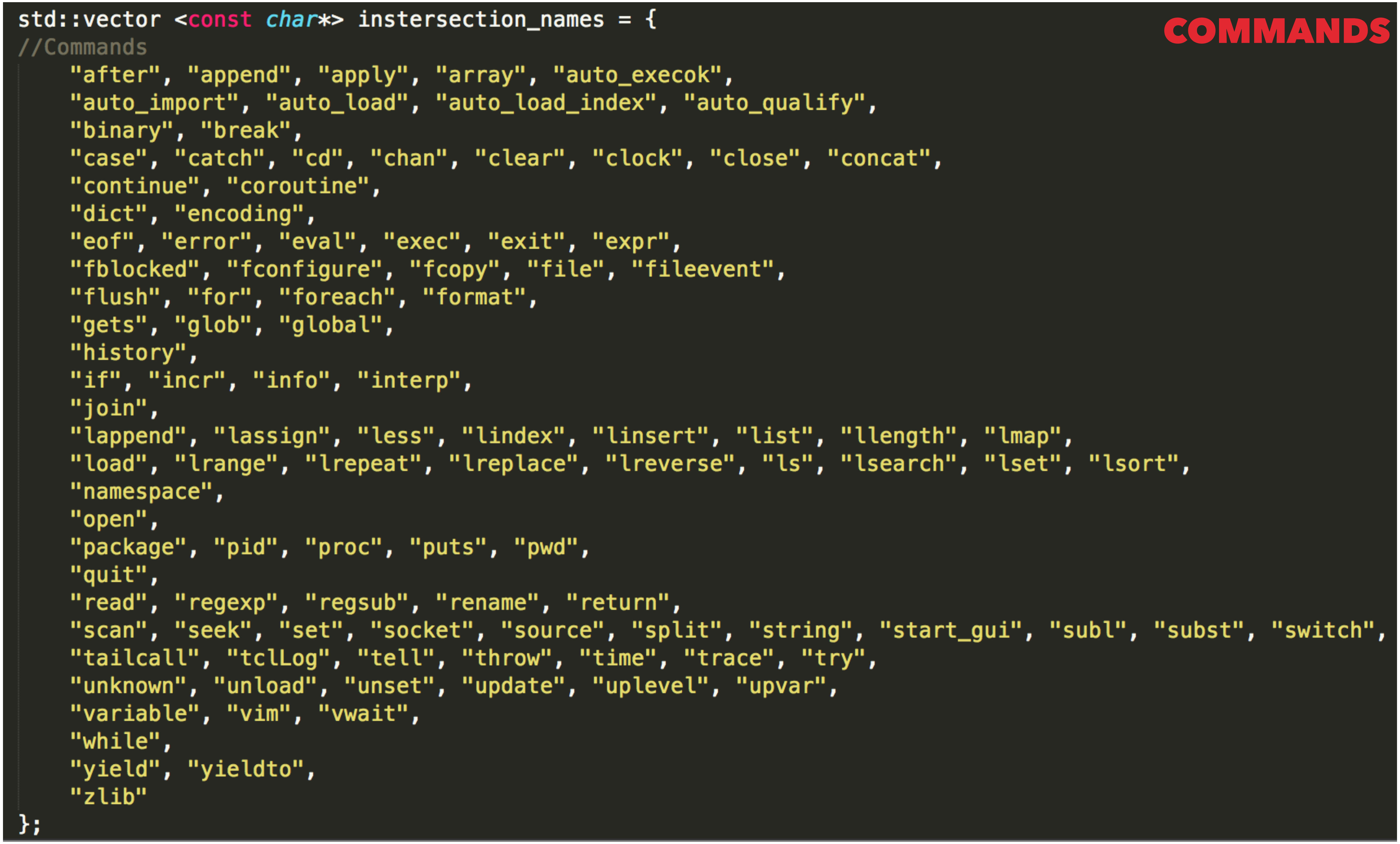
TCL

- TCL (Toolkit Command Language)
  - Used for rapid prototyping, it can easily create collections of values and manipulate them in a variety of ways.
- TCL C API
  - Tcl_FindExecutable();
  - Tcl_CreateInterp();
  - Tcl_Eval();

GNU READLINE

- GNU Readline library
  - char *cmd = readline ("→ ”);
  - custom auto-complete function
- GNU History library
  - Command History Management
  - add_history (cmd);

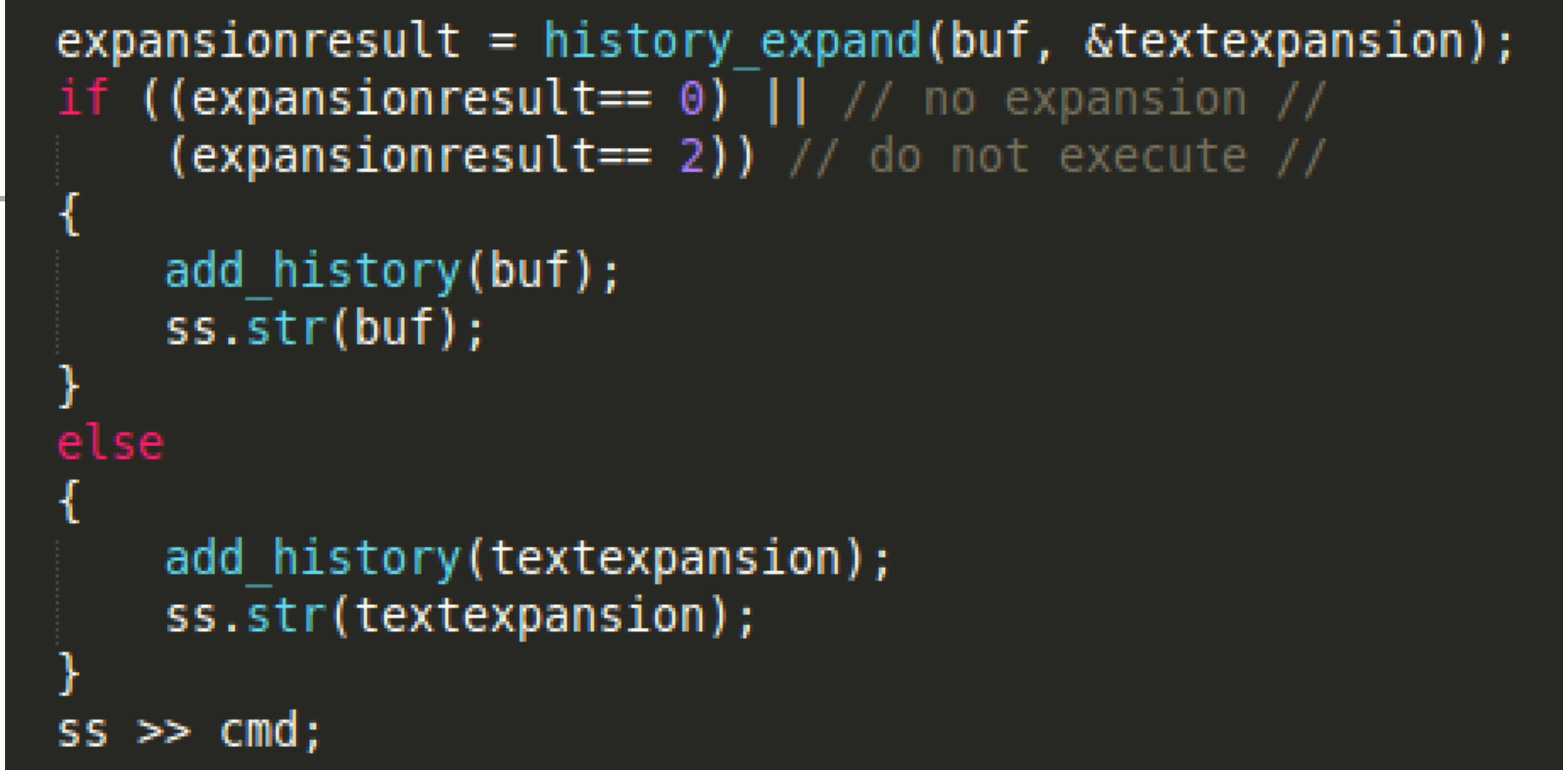
SHELL CODE (1/3)



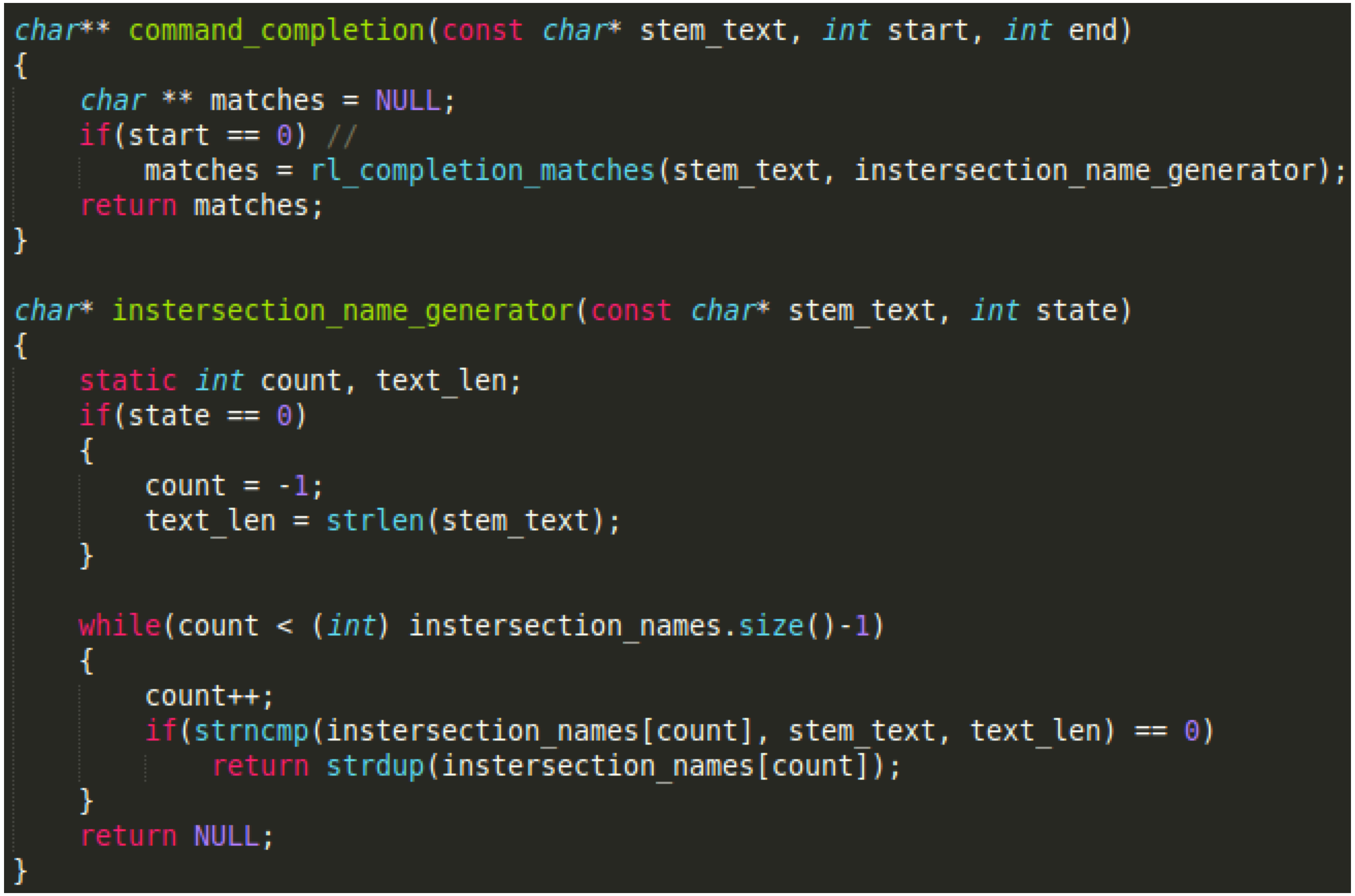
COMMANDS

```
std::vector <const char*> instersection_names = {
//Commands
    "after", "append", "apply", "array", "auto_execok",
    "auto_import", "auto_load", "auto_load_index", "auto_qualify",
    "binary", "break",
    "case", "catch", "cd", "chan", "clear", "clock", "close", "concat",
    "continue", "coroutine",
    "dict", "encoding",
    "eof", "error", "eval", "exec", "exit", "expr",
    "fblocked", "fconfigure", "fcopy", "file", "fileevent",
    "flush", "for", "foreach", "format",
    "gets", "glob", "global",
    "history",
    "if", "incr", "info", "interp",
    "join",
    "lappend", "lassign", "less", "lindex", "linsert", "list", "llength", "lmap",
    "load", "lrange", "lrepeat", "lreplace", "lreverse", "ls", "lsearch", "lset", "lsort",
    "namespace",
    "open",
    "package", "pid", "proc", "puts", "pwd",
    "quit",
    "read", "regexp", "regsub", "rename", "return",
    "scan", "seek", "set", "socket", "source", "split", "string", "start_gui", "subl", "subst", "switch",
    "tailcall", "tclLog", "tell", "throw", "time", "trace", "try",
    "unknown", "unload", "unset", "update", "uplevel", "upvar",
    "variable", "vim", "vwait",
    "while",
    "yield", "yieldto",
    "zlib"
};
```

SHELL CODE (2/3)

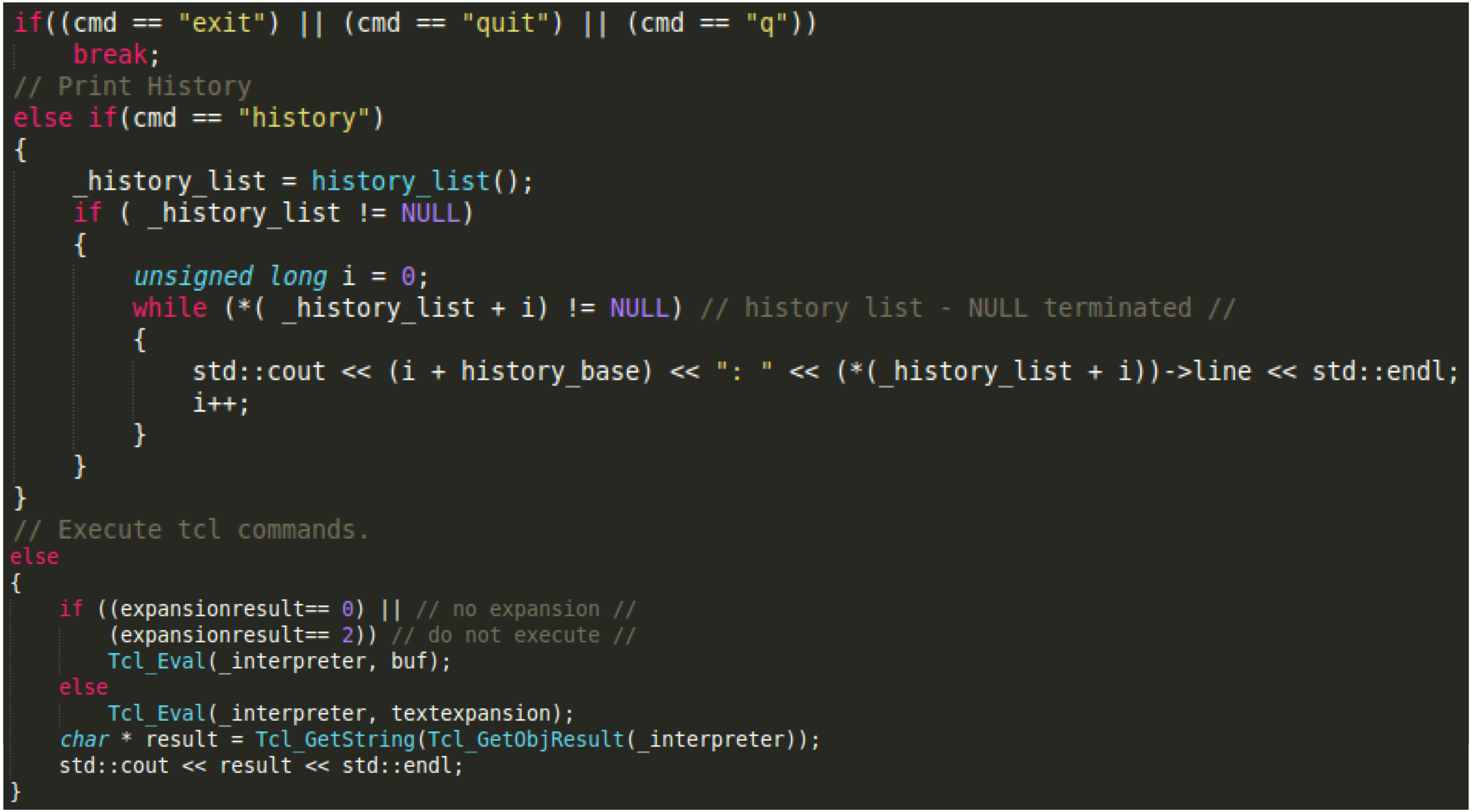


```
expansionresult = history_expand(buf, &textexpansion);
if ((expansionresult== 0) || // no expansion //
    (expansionresult== 2)) // do not execute //
{
    add_history(buf);
    ss.str(buf);
}
else
{
    add_history(textexpansion);
    ss.str(textexpansion);
}
ss >> cmd;
```



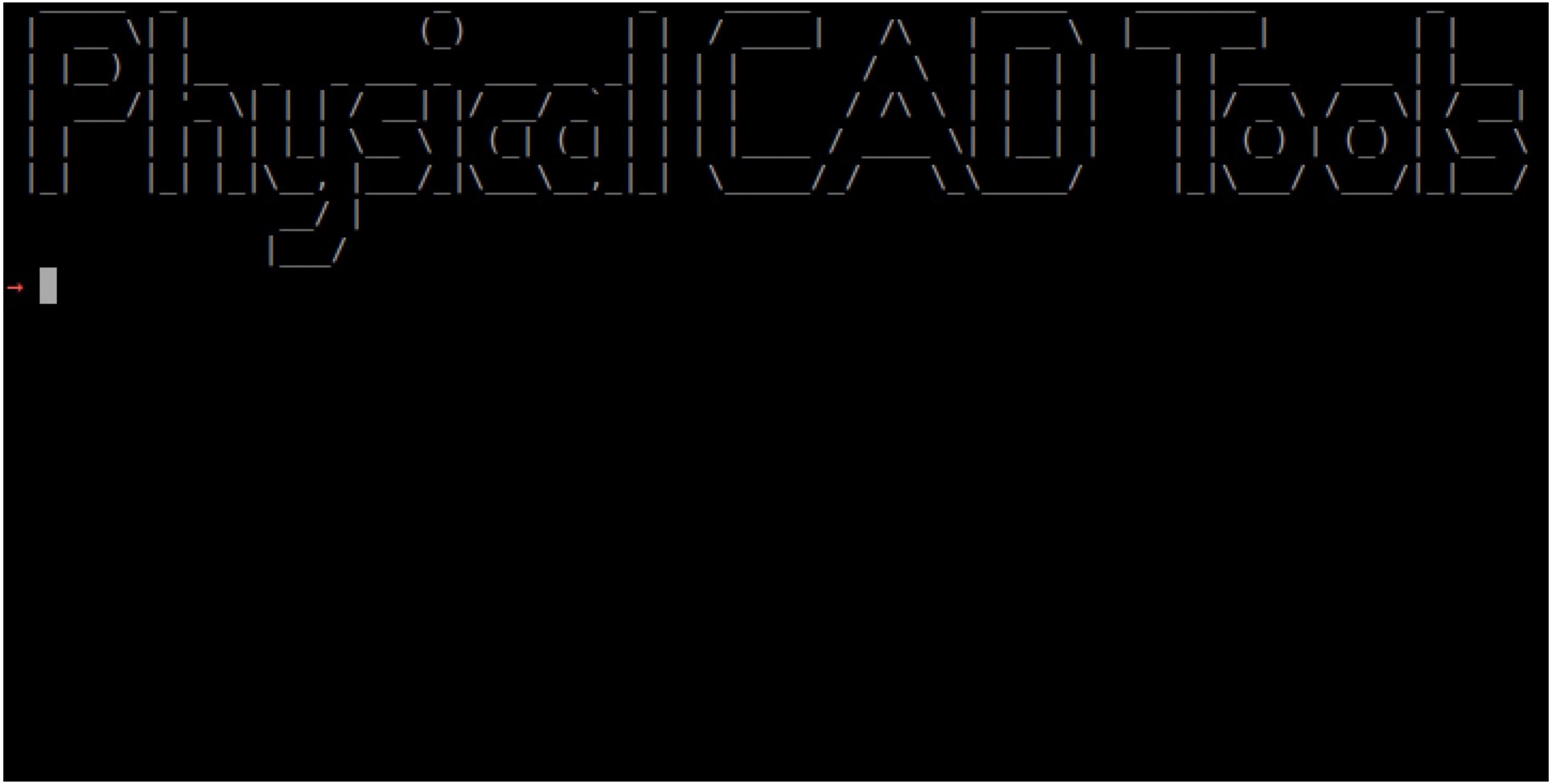
```
char** command_completion(const char* stem_text, int start, int end)
{
    char ** matches = NULL;
    if(start == 0) //
        matches = rl_completion_matches(stem_text, instersection_name_generator);
    return matches;
}

char* instersection_name_generator(const char* stem_text, int state)
{
    static int count, text_len;
    if(state == 0)
    {
        count = -1;
        text_len = strlen(stem_text);
    }

    while(count < (int) instersection_names.size()-1)
    {
        count++;
        if(strncmp(instersection_names[count], stem_text, text_len) == 0)
            return strdup(instersection_names[count]);
    }
    return NULL;
}
```

SHELL CODE (3/3)



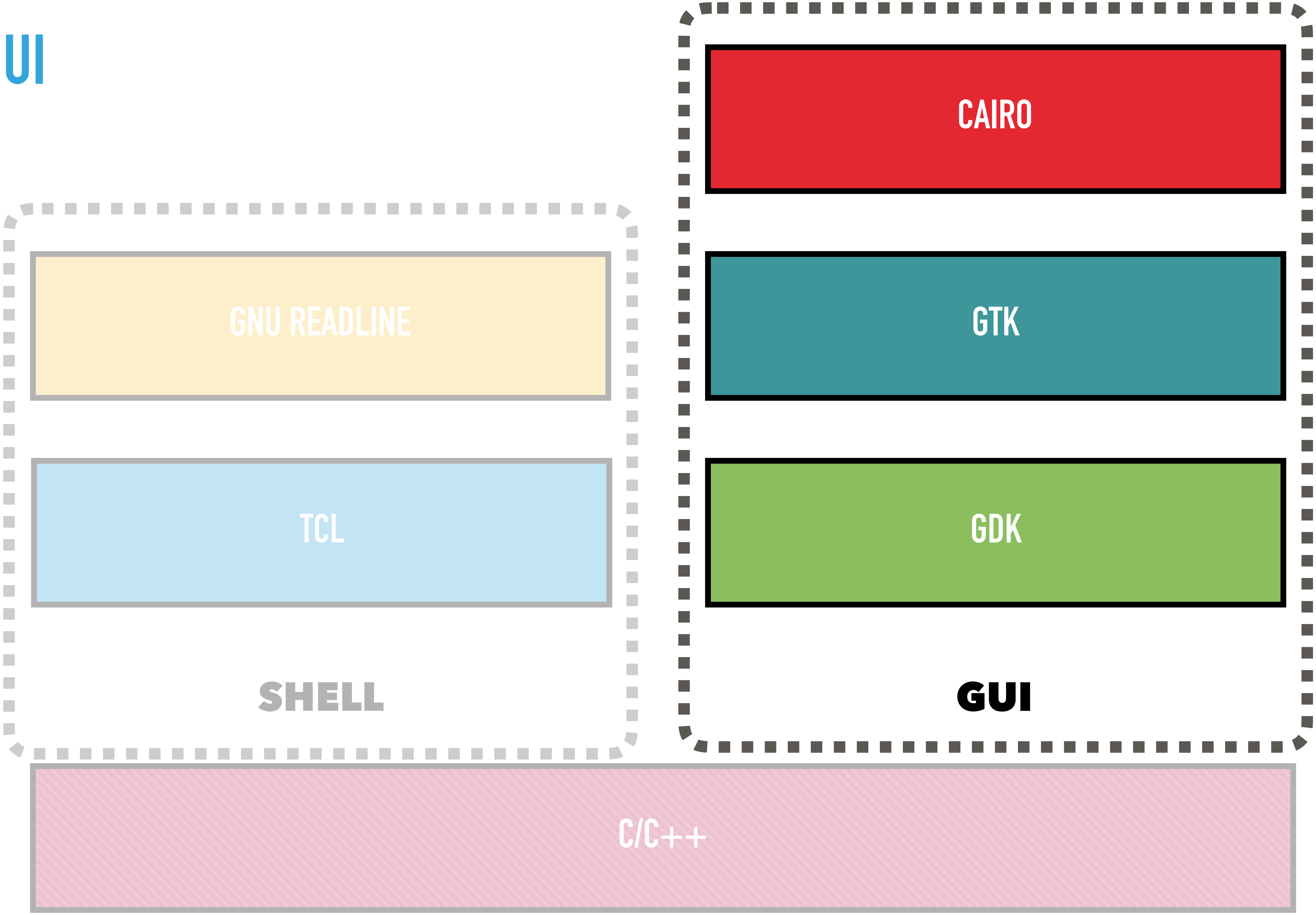
```
if((cmd == "exit") || (cmd == "quit") || (cmd == "q"))
    break;
// Print History
else if(cmd == "history")
{
    _history_list = history_list();
    if ( _history_list != NULL)
    {
        unsigned long i = 0;
        while (*( _history_list + i) != NULL) // history list - NULL terminated //
        {
            std::cout << (i + history_base) << ": " << (*(_history_list + i))->line << std::endl;
            i++;
        }
    }
}
// Execute tcl commands.
else
{
    if ((expansionresult== 0) || // no expansion //
        (expansionresult== 2)) // do not execute //
        Tcl_Eval(_interpreter, buf);
    else
        Tcl_Eval(_interpreter, textexpansion);
    char * result = Tcl_GetString(Tcl_GetObjResult(_interpreter));
    std::cout << result << std::endl;
}
```

SHELL



```
 _____  _               _           _    _____          _____    _______          _
|  __ \| |             (_)         | |  / ____|   /\   |  __ \  |__   __|        | |
| |__) | |__  _   _ ___ _  ___ __ _| | | |       /  \  | |  | |    | | ___   ___ | |___
|  ___/| '_ \| | | / __| |/ __/ _` | | | |      / /\ \ | |  | |    | |/ _ \ / _ \| / __|
| |    | | | | |_| \__ \ | (_| (_| | | | |____ / ____ \| |__| |    | | (_) | (_) | \__ \
|_|    |_| |_|\__, |___/_|\___\__,_|_|  \_____/_/    \_\_____/     |_|\___/ \___/|_|___/
               __/ |
              |___/
→
```

GUI



GNU READLINE

TCL

SHELL

CAIRO

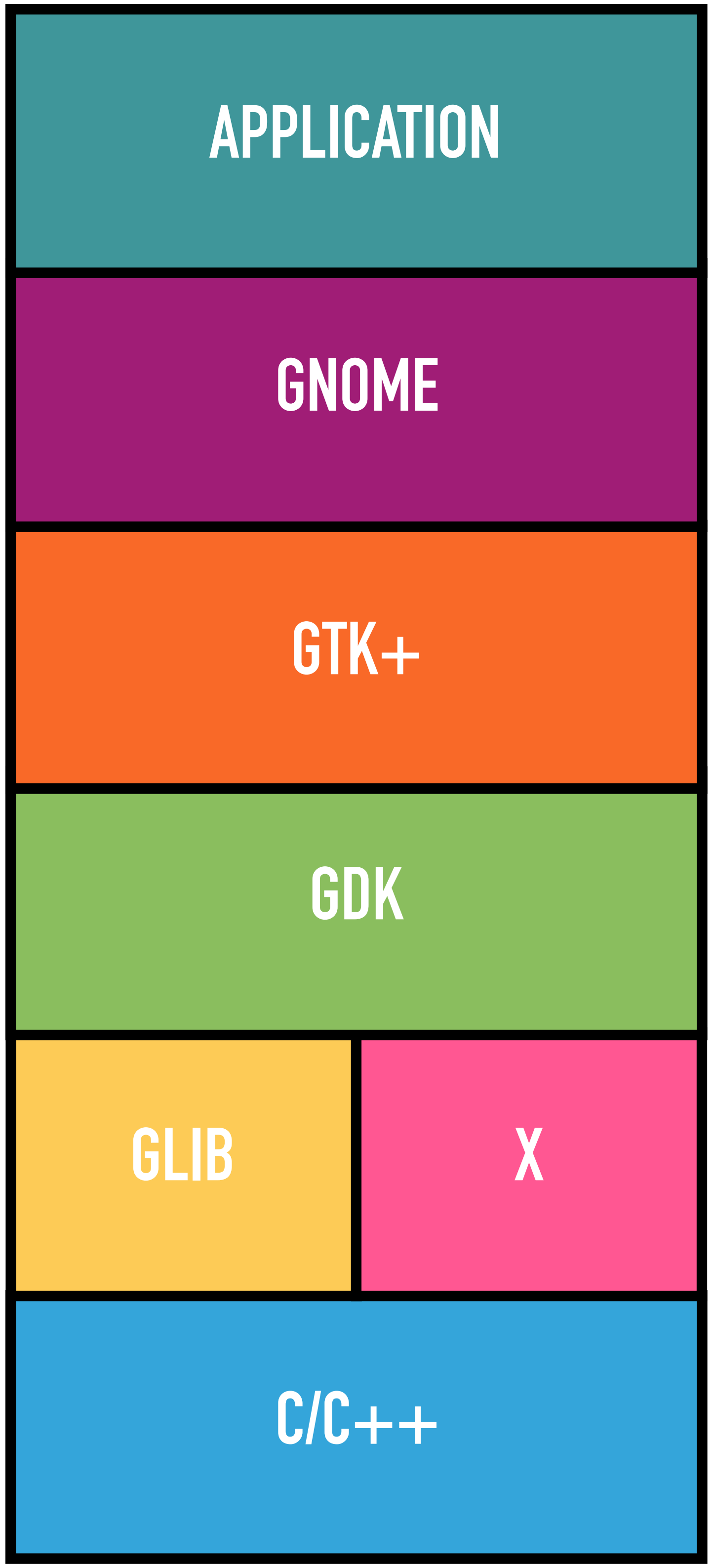
GTK

GDK

GUI

C/C++

GTK+2.0

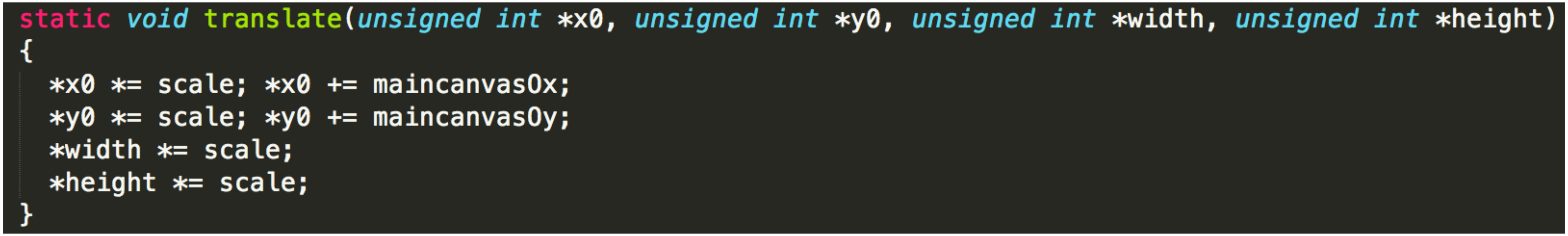


- X11 Graphics Library
  - low-level functions to control the display
- GLIB
  - library of C functions, macros and structs used by GDK, GTK+ and GNOME
- C Standard Libraries
- Linux System Calls

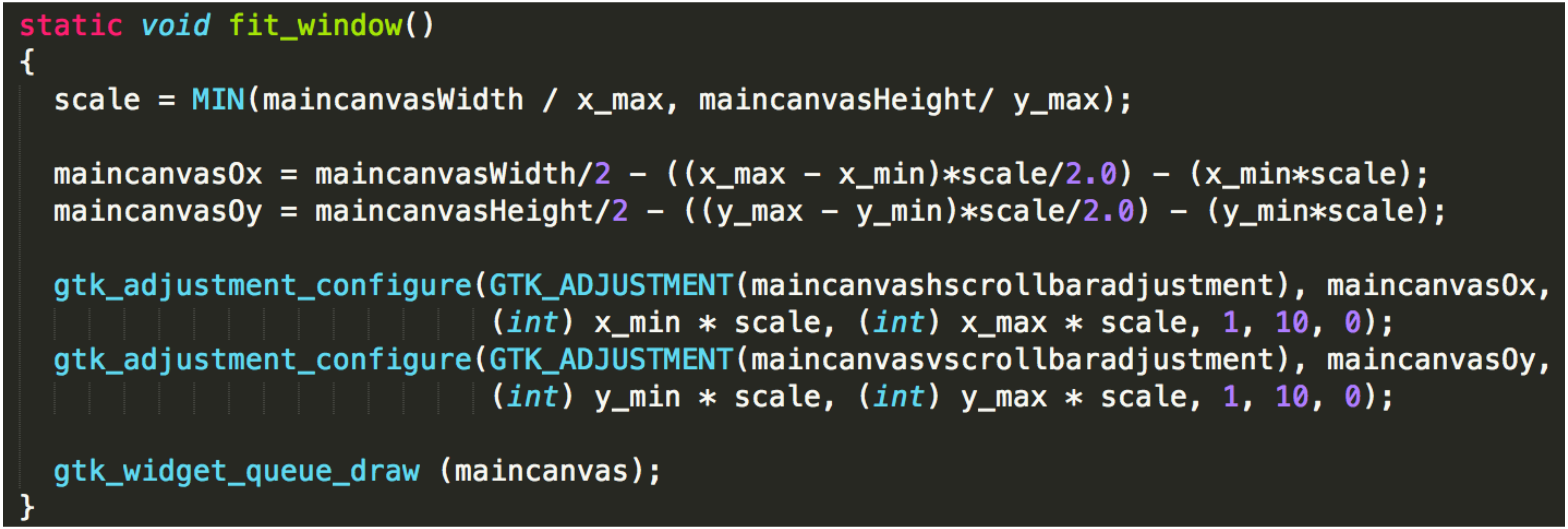
CAIRO

- CAIRO is a 2D graphics library with support for multiple output devices
- CAIRO API
  - CAIRO is implemented as a library written in C programming language, but provides bindings for multiple other programming languages.
- CAIRO is a free software under GNU Licence

GUI CODE (1/2)

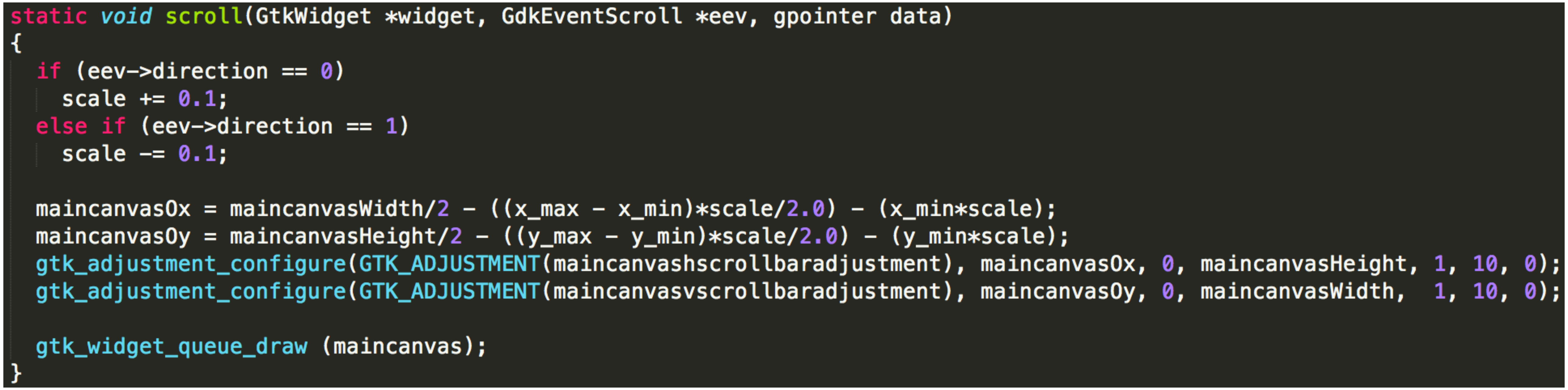


```c
static void translate(unsigned int *x0, unsigned int *y0, unsigned int *width, unsigned int *height)
{
  *x0 *= scale; *x0 += maincanvasOx;
  *y0 *= scale; *y0 += maincanvasOy;
  *width *= scale;
  *height *= scale;
}
```

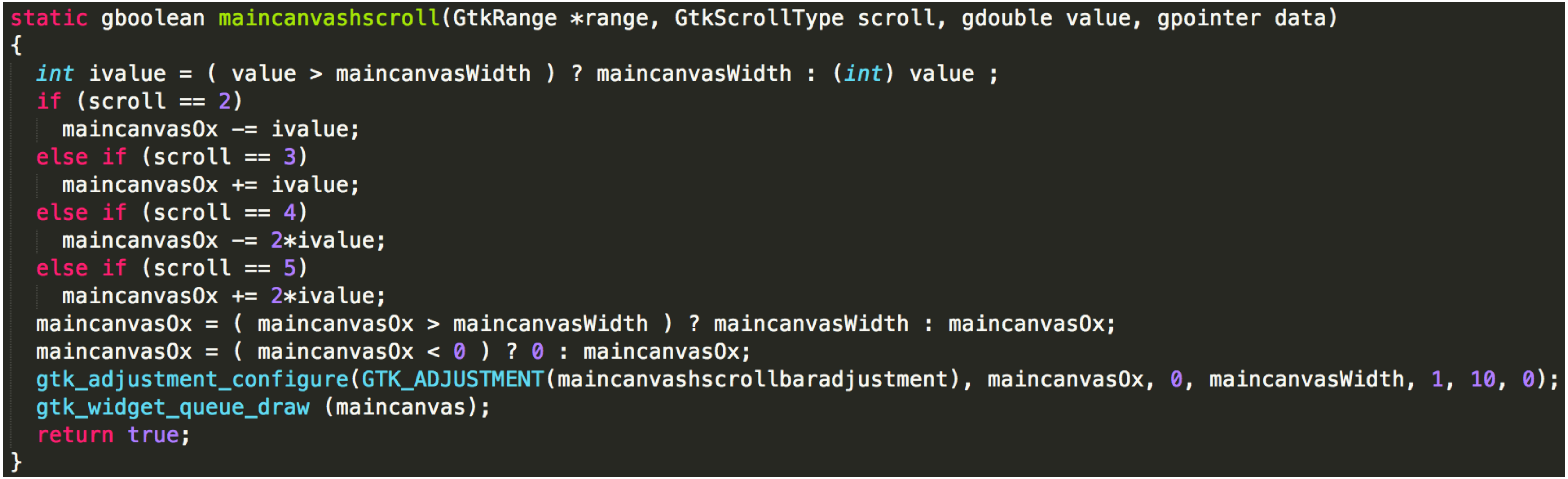


```c
static void fit_window()
{
  scale = MIN(maincanvasWidth / x_max, maincanvasHeight/ y_max);

  maincanvasOx = maincanvasWidth/2 - ((x_max - x_min)*scale/2.0) - (x_min*scale);
  maincanvasOy = maincanvasHeight/2 - ((y_max - y_min)*scale/2.0) - (y_min*scale);

  gtk_adjustment_configure(GTK_ADJUSTMENT(maincanvashscrollbaradjustment), maincanvasOx,
                           (int) x_min * scale, (int) x_max * scale, 1, 10, 0);
  gtk_adjustment_configure(GTK_ADJUSTMENT(maincanvasvscrollbaradjustment), maincanvasOy,
                           (int) y_min * scale, (int) y_max * scale, 1, 10, 0);

  gtk_widget_queue_draw (maincanvas);
}
```

GUI CODE (2/2)

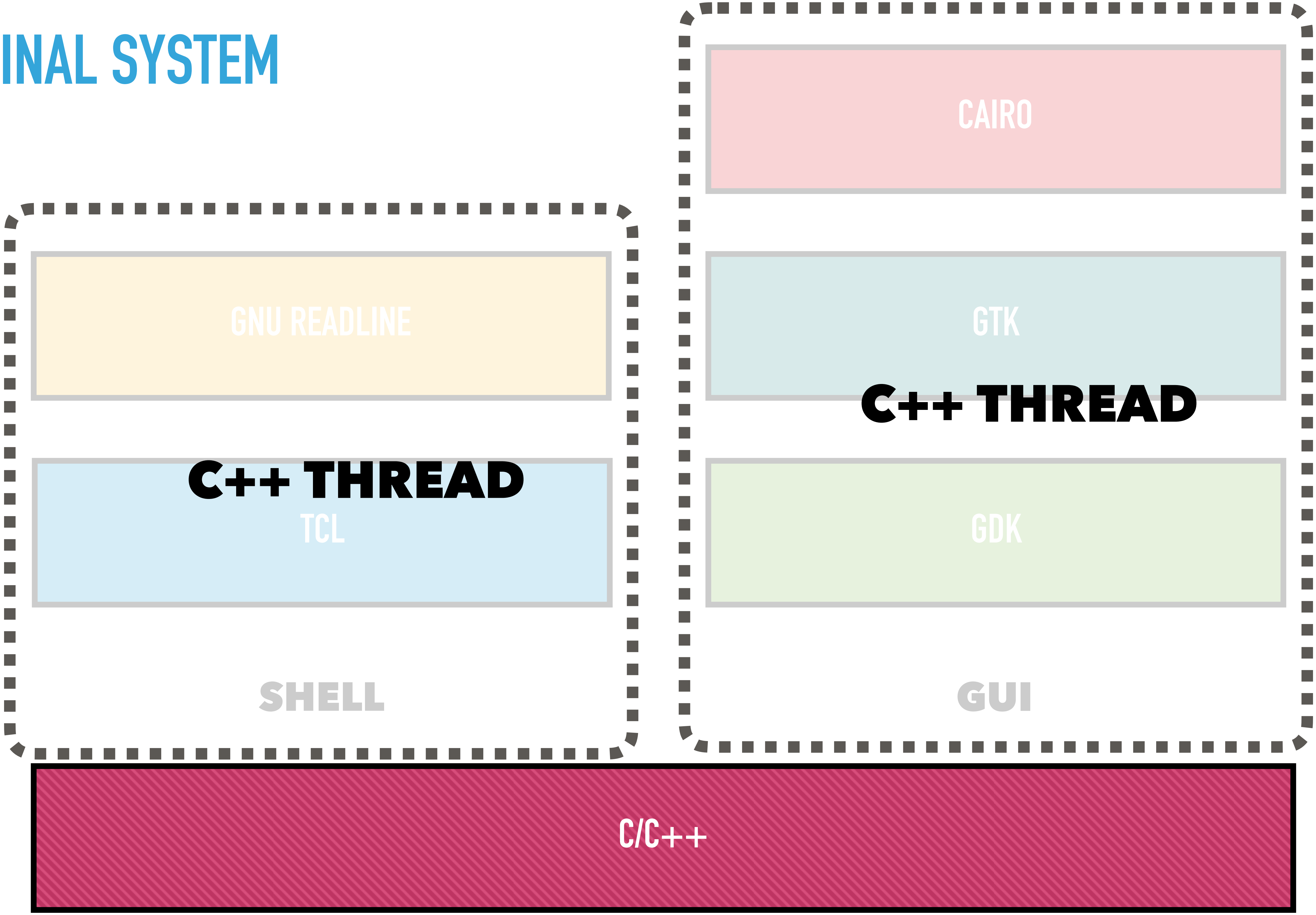


```
static void scroll(GtkWidget *widget, GdkEventScroll *eev, gpointer data)
{
  if (eev->direction == 0)
    scale += 0.1;
  else if (eev->direction == 1)
    scale -= 0.1;

  maincanvasOx = maincanvasWidth/2 - ((x_max - x_min)*scale/2.0) - (x_min*scale);
  maincanvasOy = maincanvasHeight/2 - ((y_max - y_min)*scale/2.0) - (y_min*scale);
  gtk_adjustment_configure(GTK_ADJUSTMENT(maincanvashscrollbaradjustment), maincanvasOx, 0, maincanvasHeight, 1, 10, 0);
  gtk_adjustment_configure(GTK_ADJUSTMENT(maincanvasvscrollbaradjustment), maincanvasOy, 0, maincanvasWidth,  1, 10, 0);

  gtk_widget_queue_draw (maincanvas);
}
```

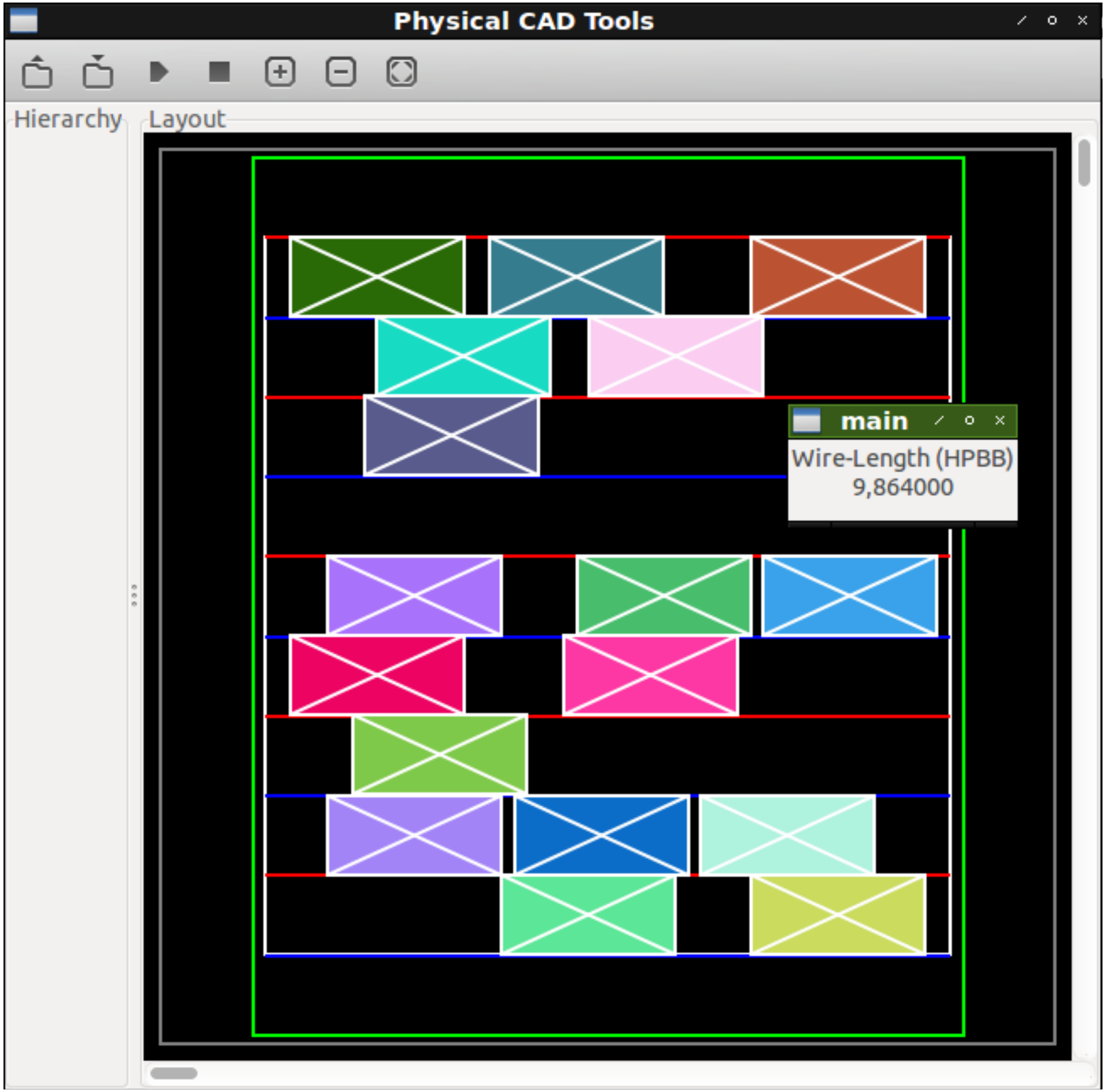


```
static gboolean maincanvashscroll(GtkRange *range, GtkScrollType scroll, gdouble value, gpointer data)
{
  int ivalue = ( value > maincanvasWidth ) ? maincanvasWidth : (int) value ;
  if (scroll == 2)
    maincanvasOx -= ivalue;
  else if (scroll == 3)
    maincanvasOx += ivalue;
  else if (scroll == 4)
    maincanvasOx -= 2*ivalue;
  else if (scroll == 5)
    maincanvasOx += 2*ivalue;
  maincanvasOx = ( maincanvasOx > maincanvasWidth ) ? maincanvasWidth : maincanvasOx;
  maincanvasOx = ( maincanvasOx < 0 ) ? 0 : maincanvasOx;
  gtk_adjustment_configure(GTK_ADJUSTMENT(maincanvashscrollbaradjustment), maincanvasOx, 0, maincanvasWidth, 1, 10, 0);
  gtk_widget_queue_draw (maincanvas);
  return true;
}
```

FINAL SYSTEM

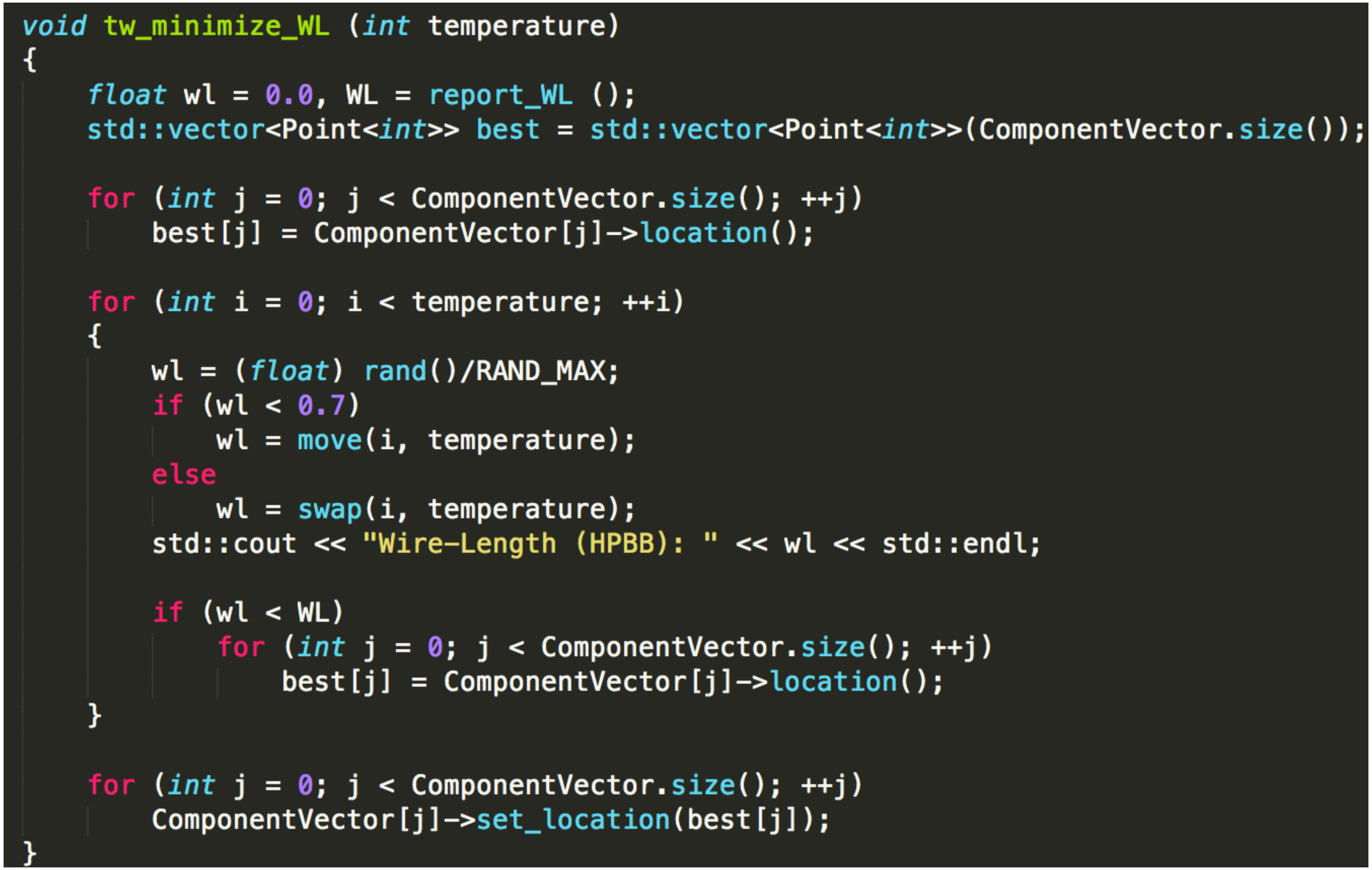


GUI



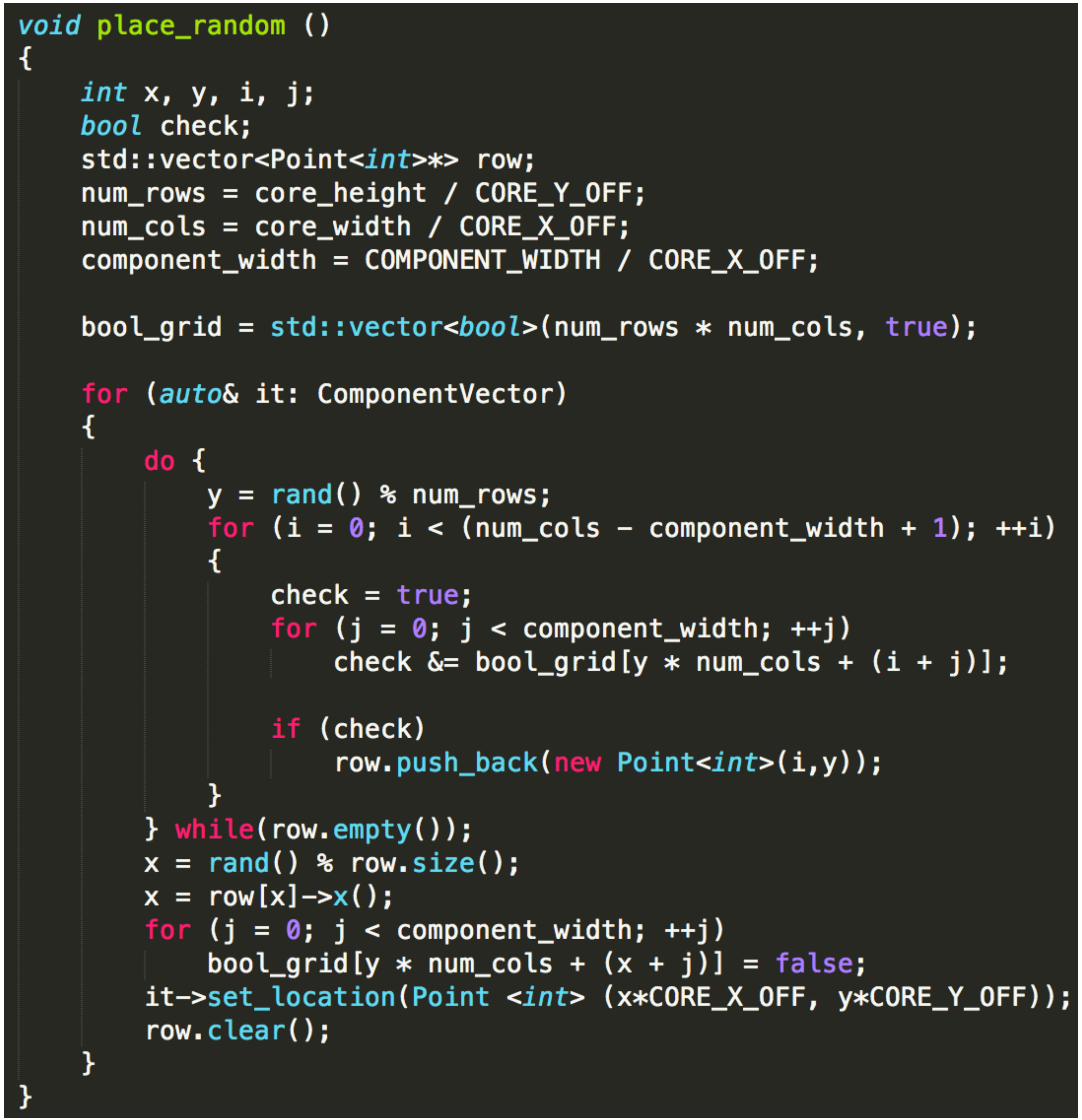
Wire-Length (HPBB)
9,864000

TIMBER WOLF



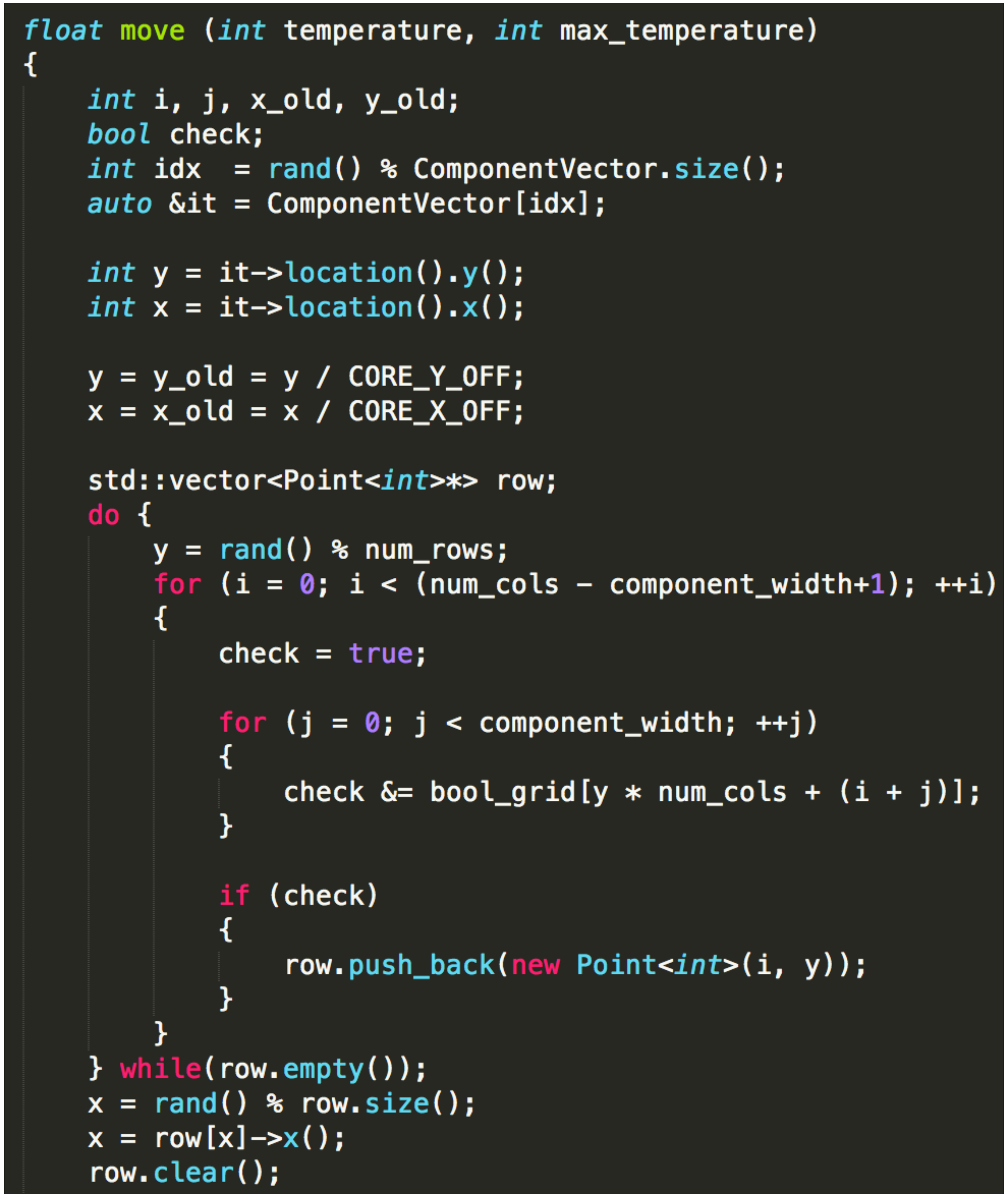
```
void tw_minimize_WL (int temperature)
{
    float wl = 0.0, WL = report_WL ();
    std::vector<Point<int>> best = std::vector<Point<int>>(ComponentVector.size());

    for (int j = 0; j < ComponentVector.size(); ++j)
        best[j] = ComponentVector[j]->location();

    for (int i = 0; i < temperature; ++i)
    {
        wl = (float) rand()/RAND_MAX;
        if (wl < 0.7)
            wl = move(i, temperature);
        else
            wl = swap(i, temperature);
        std::cout << "Wire-Length (HPBB): " << wl << std::endl;

        if (wl < WL)
            for (int j = 0; j < ComponentVector.size(); ++j)
                best[j] = ComponentVector[j]->location();
    }

    for (int j = 0; j < ComponentVector.size(); ++j)
        ComponentVector[j]->set_location(best[j]);
}
```

TIMBER WOLF: RANDOM PLACEMENT

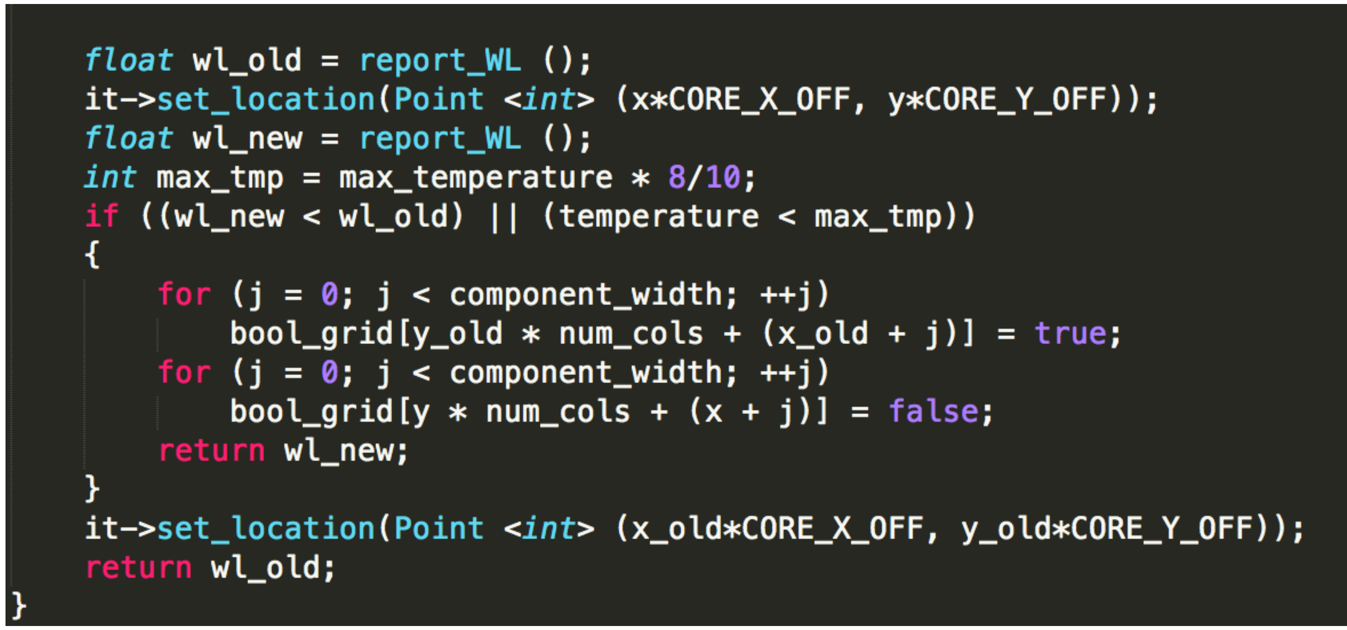


```
void place_random ()
{
    int x, y, i, j;
    bool check;
    std::vector<Point<int>*> row;
    num_rows = core_height / CORE_Y_OFF;
    num_cols = core_width / CORE_X_OFF;
    component_width = COMPONENT_WIDTH / CORE_X_OFF;

    bool_grid = std::vector<bool>(num_rows * num_cols, true);

    for (auto& it: ComponentVector)
    {
        do {
            y = rand() % num_rows;
            for (i = 0; i < (num_cols - component_width + 1); ++i)
            {
                check = true;
                for (j = 0; j < component_width; ++j)
                    check &= bool_grid[y * num_cols + (i + j)];

                if (check)
                    row.push_back(new Point<int>(i,y));
            }
        } while(row.empty());
        x = rand() % row.size();
        x = row[x]->x();
        for (j = 0; j < component_width; ++j)
            bool_grid[y * num_cols + (x + j)] = false;
        it->set_location(Point <int> (x*CORE_X_OFF, y*CORE_Y_OFF));
        row.clear();
    }
}
```

TIMBER WOLF: MOVE

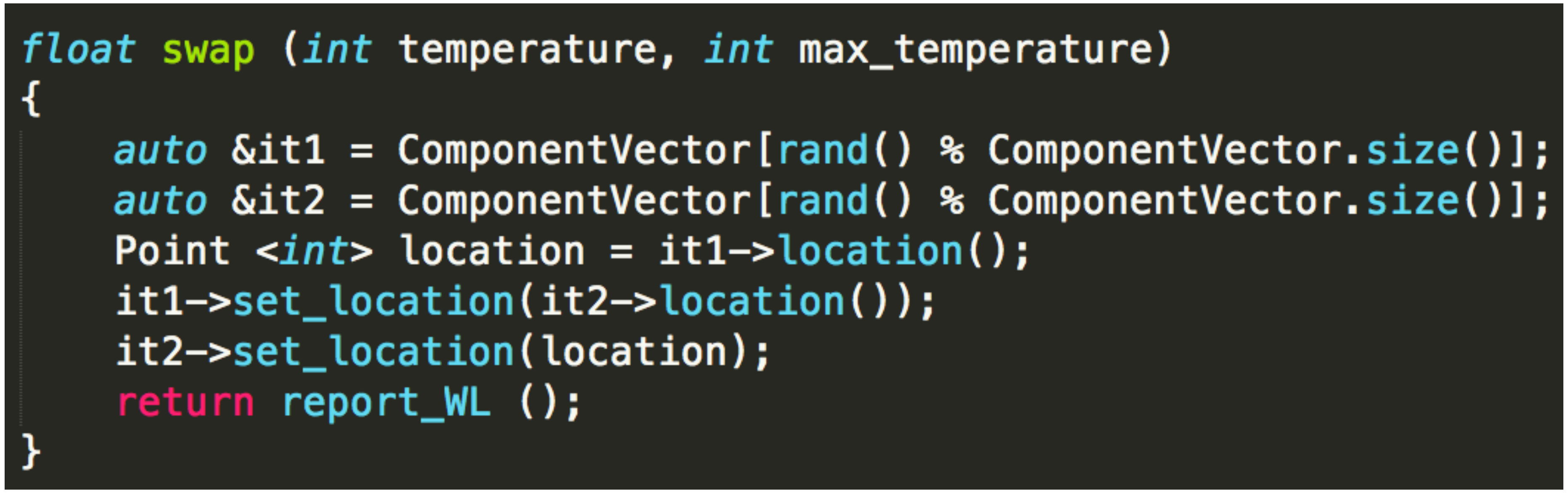


```
float move (int temperature, int max_temperature)
{
    int i, j, x_old, y_old;
    bool check;
    int idx  = rand() % ComponentVector.size();
    auto &it = ComponentVector[idx];

    int y = it->location().y();
    int x = it->location().x();

    y = y_old = y / CORE_Y_OFF;
    x = x_old = x / CORE_X_OFF;

    std::vector<Point<int>*> row;
    do {
        y = rand() % num_rows;
        for (i = 0; i < (num_cols - component_width+1); ++i)
        {
            check = true;

            for (j = 0; j < component_width; ++j)
            {
                check &= bool_grid[y * num_cols + (i + j)];
            }

            if (check)
            {
                row.push_back(new Point<int>(i, y));
            }
        }
    } while(row.empty());
    x = rand() % row.size();
    x = row[x]->x();
    row.clear();
```



```
    float wl_old = report_WL ();
    it->set_location(Point <int> (x*CORE_X_OFF, y*CORE_Y_OFF));
    float wl_new = report_WL ();
    int max_tmp = max_temperature * 8/10;
    if ((wl_new < wl_old) || (temperature < max_tmp))
    {
        for (j = 0; j < component_width; ++j)
            bool_grid[y_old * num_cols + (x_old + j)] = true;
        for (j = 0; j < component_width; ++j)
            bool_grid[y * num_cols + (x + j)] = false;
        return wl_new;
    }
    it->set_location(Point <int> (x_old*CORE_X_OFF, y_old*CORE_Y_OFF));
    return wl_old;
}
```

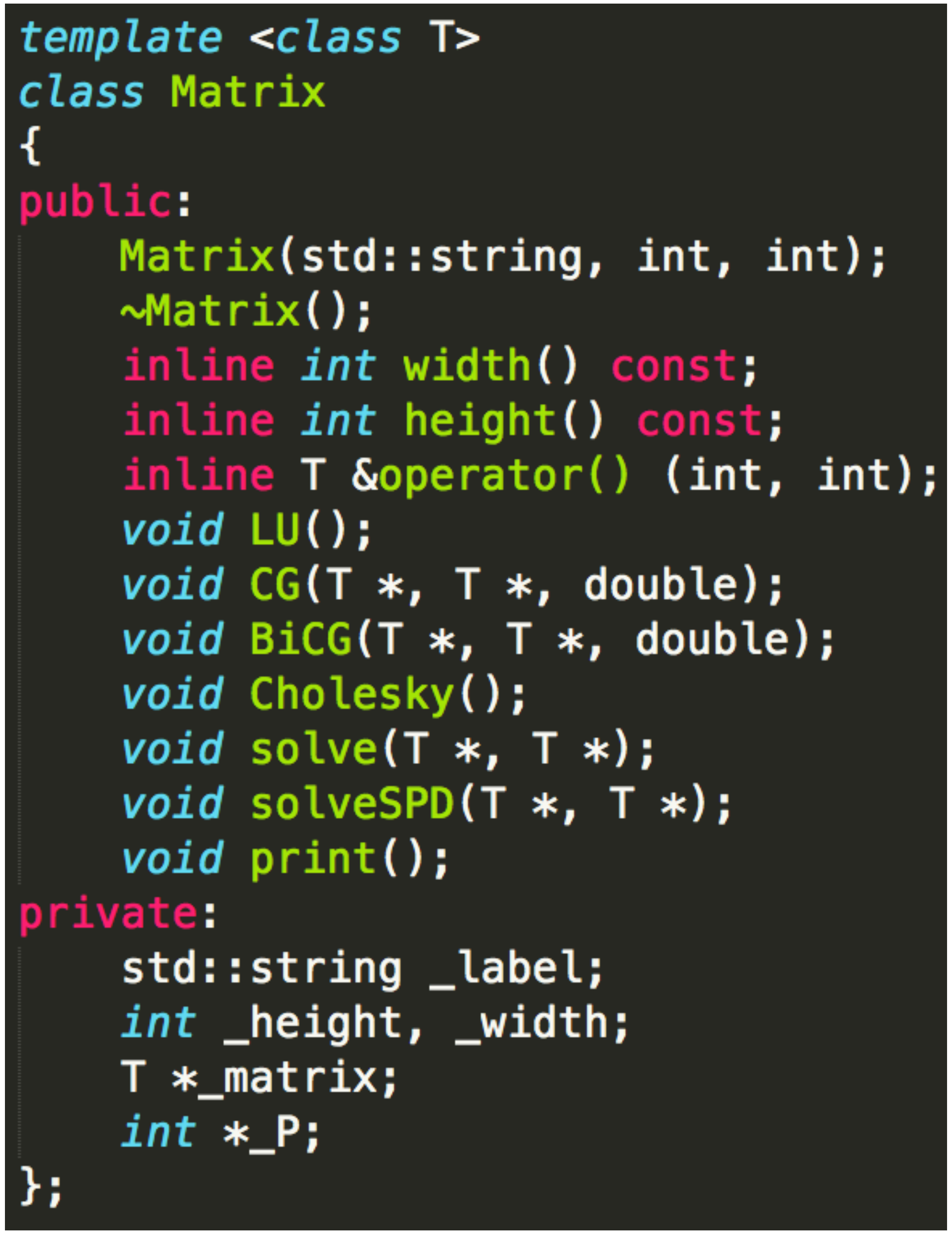
TIMBER WOLF: SWAP



```
float swap (int temperature, int max_temperature)
{
    auto &it1 = ComponentVector[rand() % ComponentVector.size()];
    auto &it2 = ComponentVector[rand() % ComponentVector.size()];
    Point <int> location = it1->location();
    it1->set_location(it2->location());
    it2->set_location(location);
    return report_WL ();
}
```

MATRIX

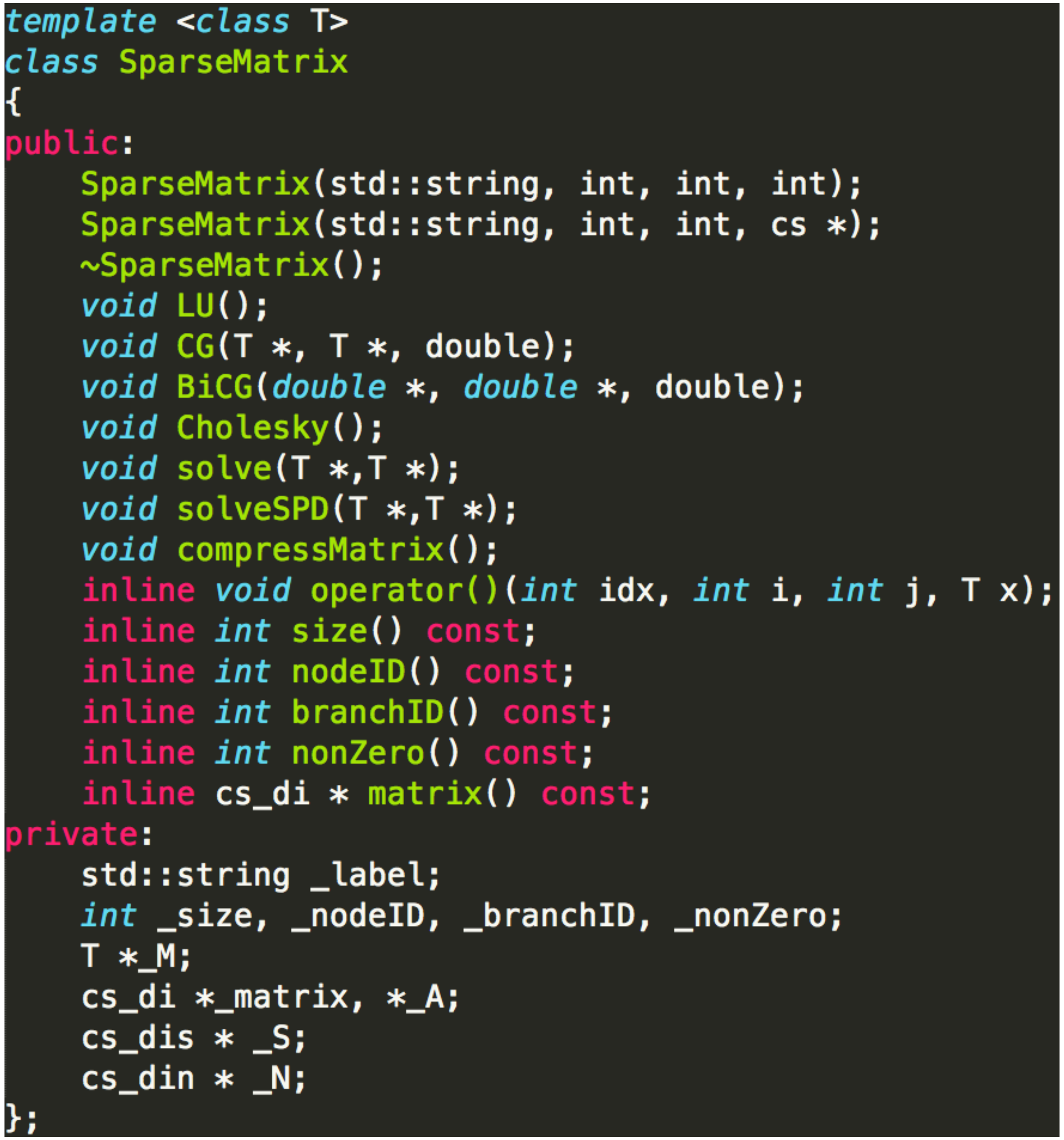
- Template Matrix Class
  - LU/ LUSolver
  - Cholesky/SPD Solver
  - CG
  - BiCG



```
template <class T>
class Matrix
{
public:
    Matrix(std::string, int, int);
    ~Matrix();
    inline int width() const;
    inline int height() const;
    inline T &operator() (int, int);
    void LU();
    void CG(T *, T *, double);
    void BiCG(T *, T *, double);
    void Cholesky();
    void solve(T *, T *);
    void solveSPD(T *, T *);
    void print();
private:
    std::string _label;
    int _height, _width;
    T *_matrix;
    int *_P;
};
```

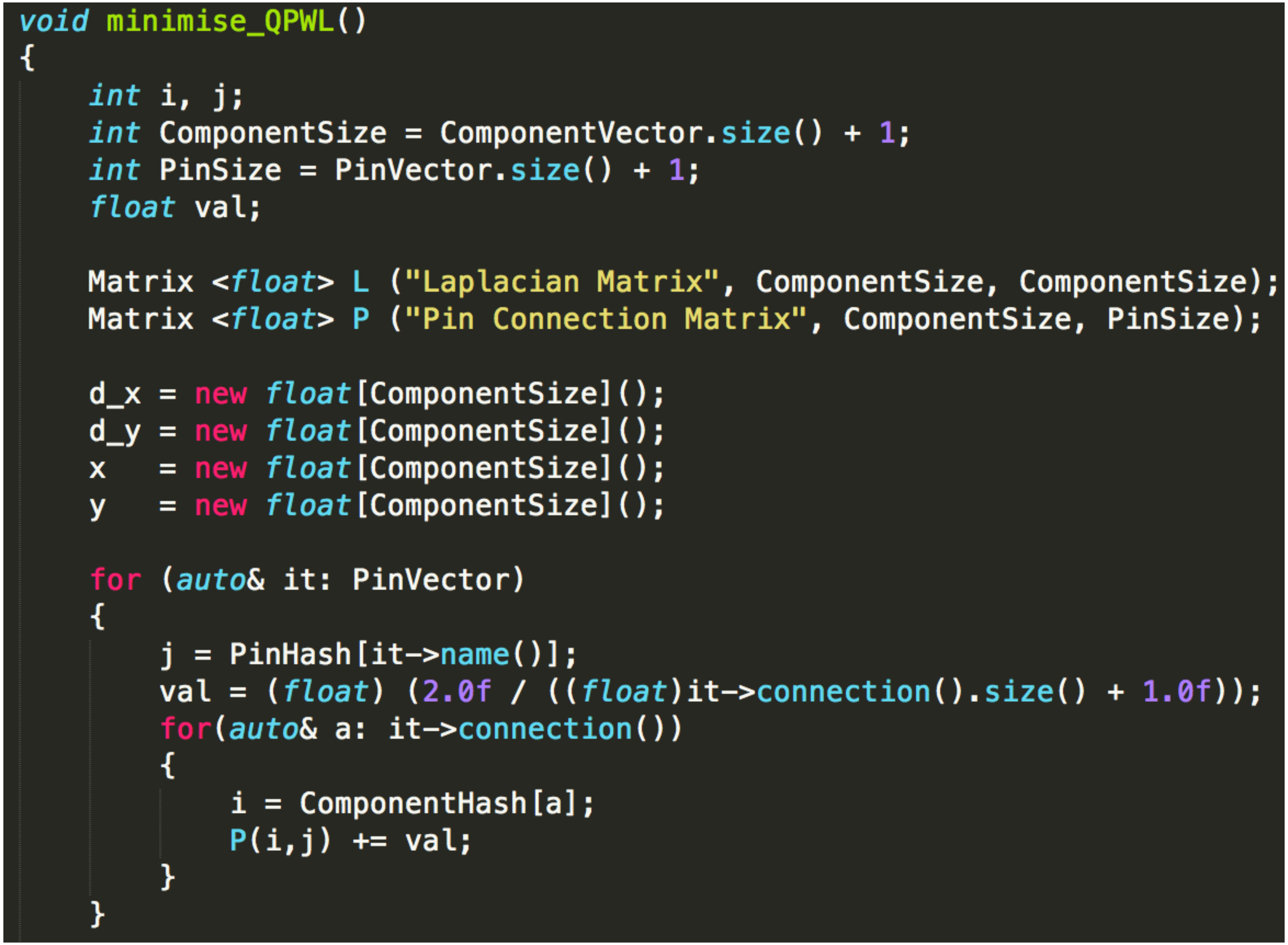
SPARSE-MATRIX

- Template Sparse-Matrix Class
  - LU/ LUSolver
  - Cholesky/SPD Solver
  - CG
  - BiCG
- Based on Sparse Suite.



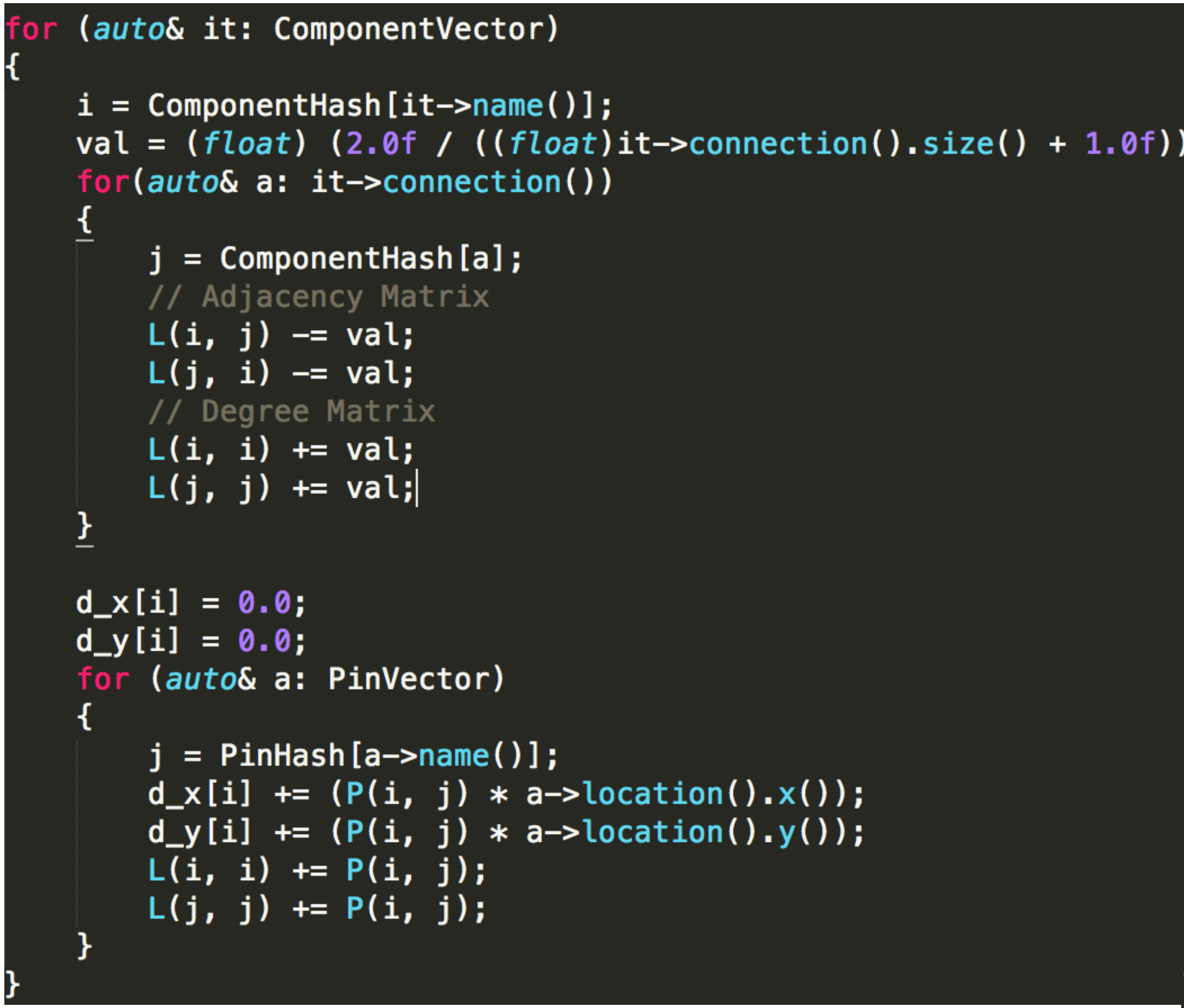
```cpp
template <class T>
class SparseMatrix
{
public:
    SparseMatrix(std::string, int, int, int);
    SparseMatrix(std::string, int, int, cs *);
    ~SparseMatrix();
    void LU();
    void CG(T *, T *, double);
    void BiCG(double *, double *, double);
    void Cholesky();
    void solve(T *,T *);
    void solveSPD(T *,T *);
    void compressMatrix();
    inline void operator()(int idx, int i, int j, T x);
    inline int size() const;
    inline int nodeID() const;
    inline int branchID() const;
    inline int nonZero() const;
    inline cs_di * matrix() const;
private:
    std::string _label;
    int _size, _nodeID, _branchID, _nonZero;
    T *_M;
    cs_di *_matrix, *_A;
    cs_dis * _S;
    cs_din * _N;
};
```

QP: MINIMISE WIRE LENGTH (1/2)

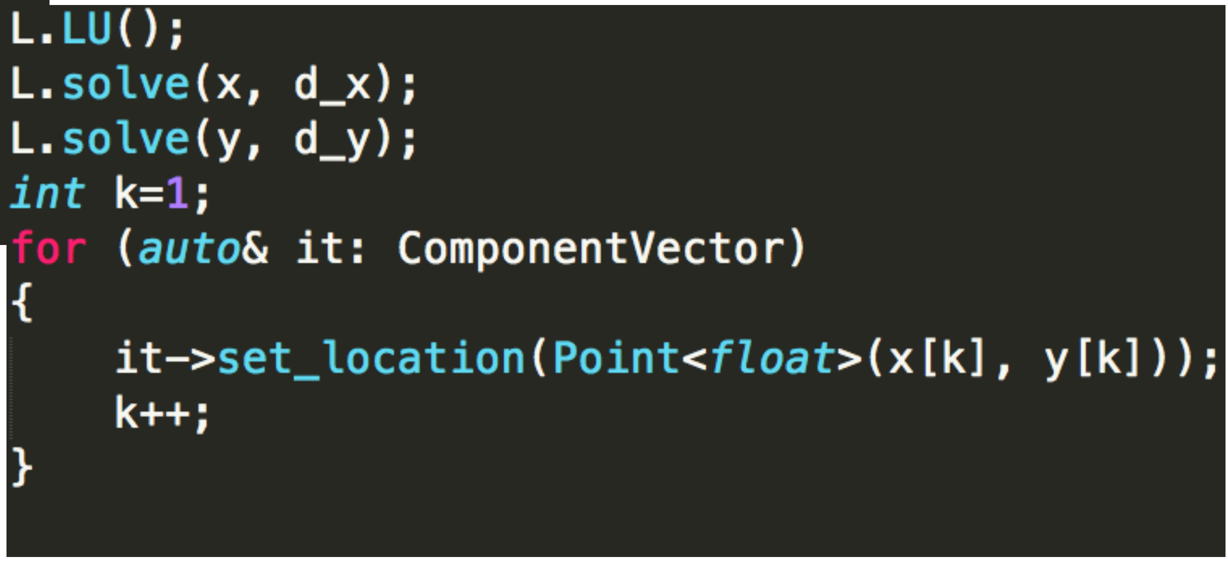


```
void minimise_QPWL()
{
    int i, j;
    int ComponentSize = ComponentVector.size() + 1;
    int PinSize = PinVector.size() + 1;
    float val;

    Matrix <float> L ("Laplacian Matrix", ComponentSize, ComponentSize);
    Matrix <float> P ("Pin Connection Matrix", ComponentSize, PinSize);

    d_x = new float[ComponentSize]();
    d_y = new float[ComponentSize]();
    x   = new float[ComponentSize]();
    y   = new float[ComponentSize]();

    for (auto& it: PinVector)
    {
        j = PinHash[it->name()];
        val = (float) (2.0f / ((float)it->connection().size() + 1.0f));
        for(auto& a: it->connection())
        {
            i = ComponentHash[a];
            P(i,j) += val;
        }
    }
```

QP: MINIMISE WIRE LENGTH (2/2)



```
for (auto& it: ComponentVector)
{
    i = ComponentHash[it->name()];
    val = (float) (2.0f / ((float)it->connection().size() + 1.0f));
    for(auto& a: it->connection())
    {
        j = ComponentHash[a];
        // Adjacency Matrix
        L(i, j) -= val;
        L(j, i) -= val;
        // Degree Matrix
        L(i, i) += val;
        L(j, j) += val;
    }

    d_x[i] = 0.0;
    d_y[i] = 0.0;
    for (auto& a: PinVector)
    {
        j = PinHash[a->name()];
        d_x[i] += (P(i, j) * a->location().x());
        d_y[i] += (P(i, j) * a->location().y());
        L(i, i) += P(i, j);
        L(j, j) += P(i, j);
    }
}
```



```
L.LU();
L.solve(x, d_x);
L.solve(y, d_y);
int k=1;
for (auto& it: ComponentVector)
{
    it->set_location(Point<float>(x[k], y[k]));
    k++;
}
```

THANKS!

Panagiotis Skrimponis