一、编写mytimeinput.h文件；

```
#ifndef _MY_TIME_INPUT_H
#define _MY_TIME_INPUT_H

#include <glib.h>
#include <glib-object.h>
#include <gtk/gtkentry.h>

G_BEGIN_DECLS

#define MY_TIME_INPUT_TYPE \
  (my_time_input_get_type())
#define MY_TIME_INPUT(obj) \
  (G_TYPE_CHECK_INSTANCE_CAST((obj), MY_TIME_INPUT_TYPE, MyTimeInput))
#define MY_TIME_INPUT_CLASS(klass) \
  (G_TYPE_CHECK_CLASS_CAST((klass), MY_TIME_INPUT_TYPE, MyTimeInputClass))
#define IS_MY_TIME_INPUT(obj) \
  (G_TYPE_CHECK_INSTANCE_TYPE((klass), MY_TIME_INPUT_TYPE))
#define IS_MY_TIME_INPUT_CLASS(klass) \
  (G_TYPE_CHECK_CLASS_TYPE((klass), MY_TIME_INPUT_TYPE))

typedef struct _MyTimeInput MyTimeInput;
typedef struct _MyTimeInputClass MyTimeInputClass;

struct _MyTimeInput
{
 GtkEntry entry;
};

struct _MyTimeInputClass
{
 GtkEntryClass parent_class;
 void (* time_input_changed)(MyTimeInput *my_time_input);
};

GType my_time_input_get_type(void) G_GNUC_CONST;
GtkWidget * my_time_input_new(void);
gchar *my_time_input_get_time(MyTimeInput *my_time_input);
gint my_time_input_set_time_str(MyTimeInput *my_time_input, const gchar *time_str);
void my_time_input_set_time(MyTimeInput *my_time_input, guint time[2]);

G_END_DECLS
#endif
```

二、编写mytimeinput.c文件；

```
#include <gtk/gtk.h>
#include <gdk/gdkkeysyms.h>
```

```
#include <stdlib.h>
#include <math.h>
#include "mytimeinput.h"
#include <stdio.h>

#define MY_TIME_INPUT_GET_PRIVATE(obj) \
  (G_TYPE_INSTANCE_GET_PRIVATE((obj), MY_TIME_INPUT_TYPE, MyTimeInputPrivate))

G_DEFINE_TYPE(MyTimeInput, my_time_input, GTK_TYPE_ENTRY);

typedef struct _MyTimeInputPrivate MyTimeInputPrivate;

struct _MyTimeInputPrivate
{
 guint time[2];
};

enum
{
 CHANGED_SIGNAL,
 LAST_SIGNAL
};

static guint my_time_input_signals[LAST_SIGNAL] = { 0 };

static void my_time_input_render(MyTimeInput *my_time_input)
{
 MyTimeInputPrivate *priv = MY_TIME_INPUT_GET_PRIVATE(my_time_input);
 GString *text;
 guint i;

 text = g_string_new(NULL);
 for(i = 0; i < 2; i++)
 {
  gchar *temp = g_strdup_printf("%02d:", priv->time[i]);
  text = g_string_append(text, temp);
  g_free(temp);
 }

 text = g_string_truncate(text, 5);
 gtk_entry_set_text(GTK_ENTRY(my_time_input), text->str);
 g_string_free(text, TRUE);
}

gchar *my_time_input_get_time (MyTimeInput *my_time_input)
{
 MyTimeInputPrivate *priv = MY_TIME_INPUT_GET_PRIVATE (my_time_input);
 return g_strdup_printf ("%d:%d", priv->time[0], priv->time[1]);
}

void my_time_input_set_time (MyTimeInput *my_time_input, guint time[2])
{
```

```
 MyTimeInputPrivate *priv = MY_TIME_INPUT_GET_PRIVATE (my_time_input);

 if (time[0] >= 0 && time[0] < 24 && time[1] >= 0 && time[1] < 60)
 {
  priv->time[0] = time[0];
  priv->time[1] = time[1];
 }
 my_time_input_render (my_time_input);
 g_signal_emit_by_name ((gpointer) my_time_input, "time-input-changed");
}

gint my_time_input_set_time_str (MyTimeInput *my_time_input, const gchar *time_str)
{
 if ((strcmp("", time_str) == 0) || (time_str == NULL))
 {
  return -1;
 }
 else
 {
  guint time[2];
  sscanf(time_str, "%d:%d", &time[0], &time[1]);
  my_time_input_set_time (my_time_input, time);
  return 0;
 }
}

static void my_time_input_move_cursor(GObject *entry, GParamSpec *spec)
{
 gint cursor = gtk_editable_get_position(GTK_EDITABLE(entry));

 if (cursor <= 2)
  gtk_editable_set_position(GTK_EDITABLE(entry), 2);
 else
  gtk_editable_set_position(GTK_EDITABLE(entry), 5);

}

static gboolean my_time_input_key_pressed(GtkEntry *entry, GdkEventKey *event)
{
 MyTimeInputPrivate *priv = MY_TIME_INPUT_GET_PRIVATE(entry);
 guint k = event->keyval;
 gint cursor, value;

 if ((k >= GDK_0 && k <= GDK_9) || (k >= GDK_KP_0 && k <= GDK_KP_9))
 {
  cursor = floor (gtk_editable_get_position (GTK_EDITABLE (entry)) / 3);
  value = g_ascii_digit_value (event->string[0]);

  if (cursor == 0)
  {
   if ((priv->time[0] == 2) && (value > 3))
    return TRUE;
```

```
  if (priv->time[cursor] < 3)
  {
   priv->time[cursor] *= 10;
   priv->time[cursor] += value;
   my_time_input_render (MY_TIME_INPUT (entry));
   gtk_editable_set_position (GTK_EDITABLE (entry), (3 * cursor) + 2);
   g_signal_emit_by_name ((gpointer) entry, "time-input-changed");
  }
 }
 else if (cursor == 1)
 {
  if ((priv->time[1] == 5) && (value > 9))
   return TRUE;
  if (priv->time[cursor] < 6)
  {
   priv->time[cursor] *= 10;
   priv->time[cursor] += value;
   my_time_input_render (MY_TIME_INPUT (entry));
   gtk_editable_set_position (GTK_EDITABLE (entry), (3 * cursor) + 2);
   g_signal_emit_by_name ((gpointer) entry, "time-input-changed");
  }
 }
}
else if (k == GDK_Tab)
{
 cursor = (floor (gtk_editable_get_position (GTK_EDITABLE (entry)) / 3) + 1);
 gtk_editable_set_position (GTK_EDITABLE (entry), (3 * (cursor % 2)) + 2);
}
else if (k == GDK_BackSpace)
{
 cursor = floor (gtk_editable_get_position (GTK_EDITABLE (entry)) / 3);
 priv->time[cursor] /= 10;
 my_time_input_render (MY_TIME_INPUT (entry));
 gtk_editable_set_position (GTK_EDITABLE (entry), (3 * cursor) + 2);
 g_signal_emit_by_name ((gpointer) entry, "time-input-changed");
}
else if (k == GDK_Left)
{
 cursor = floor (gtk_editable_get_position (GTK_EDITABLE (entry)) / 3);
 if (cursor == 1)
  gtk_editable_set_position (GTK_EDITABLE (entry), 2);
}
else if (k == GDK_Right)
{
 cursor = floor (gtk_editable_get_position (GTK_EDITABLE (entry)) / 3);
 if (cursor == 0)
  gtk_editable_set_position (GTK_EDITABLE (entry), 5);
}
else if ((k == GDK_Return) || (k == GDK_KP_Enter))
 gtk_widget_activate (GTK_WIDGET (entry));
return TRUE;
}
```

```
static void my_time_input_class_init(MyTimeInputClass *klass)
{
 GObjectClass *gobject_class = G_OBJECT_CLASS(klass);

 g_type_class_add_private(klass, sizeof(MyTimeInputPrivate));

 my_time_input_signals[CHANGED_SIGNAL] =
  g_signal_new("time-input-changed",
    G_TYPE_FROM_CLASS(klass),
    G_SIGNAL_RUN_FIRST | G_SIGNAL_ACTION,
    G_STRUCT_OFFSET(MyTimeInputClass, time_input_changed),
    NULL,
    NULL,
    g_cclosure_marshal_VOID__VOID,
    G_TYPE_NONE,
    0);
}

static void my_time_input_init(MyTimeInput *my_time_input)
{
 MyTimeInputPrivate *priv = MY_TIME_INPUT_GET_PRIVATE(my_time_input);
 PangoFontDescription *fd;
 guint i;

 for (i = 0; i < 2; i++)
 {
  priv->time[i] = 0;
 }

 fd = pango_font_description_from_string("Monospace");
 gtk_widget_modify_font(GTK_WIDGET(my_time_input), fd);
 my_time_input_render(my_time_input);
 pango_font_description_free(fd);

 g_signal_connect(G_OBJECT(my_time_input), "key-press-event",
   G_CALLBACK(my_time_input_key_pressed), NULL);
 g_signal_connect(G_OBJECT(my_time_input), "notify::cursor-position",
   G_CALLBACK(my_time_input_move_cursor), NULL);
}

GtkWidget *my_time_input_new ()
{
 return GTK_WIDGET (g_object_new (my_time_input_get_type (), NULL));
}
```