Computer Labs: C Function Pointers

2º MIEIC

Pedro F. Souto (pfs@fe.up.pt)

March 14, 2024

Function Pointers

- C supports pointers to functions, which can be:
  - assigned;
  - placed in arrays;
  - members of structs or unions;
  - passed to functions;
  - returned by functions
- `int (*fp)(int);` declares `fp` as a pointer to a function that takes an integer as argument and returns an integer
- Let `int foo(int);` be such a function
- Then:

  `fp = foo;`

  initializes `fp` to point to `foo()`
- And:

  `n = (*fp)(i);`

  invokes the function pointed to by `fp`, `foo`, with argument `i` and assigns the return value to variable `n`

Function Pointers Application: Event Dispatching

- One simple implementation of event dispatching is:
  - to use a `switch` instruction on the event type;
  - to call, in each `case` clause, the corresponding event handler

```
switch(ev) { // identify event
case EV0:
    ev0_handler(); // call handler
    break;
...
```

- An alternative implementation is similar to vectored interrupts:
  - use a table (array) of (pointers to) event handlers (functions);
  - index that table to jump to the handler;

```
void (*eht[])(void) = {ev0_handler, ev1_handler, ...};
...
(*eht[ev])();  // index into table and call handler
```

- Of course, the event handlers may take arguments

Function Pointers Application: Event Dispatching in State Machines

- We can use the state as an argument to the event handler:

```
typedef enum {ST0, ST1, ST2, ..} state_t;
void (*eht[])(state_t) = {ev0_handler, ev1_handler, ...};
state_t st;
...
(*eht[ev])(st);  // index into table and call handler
```

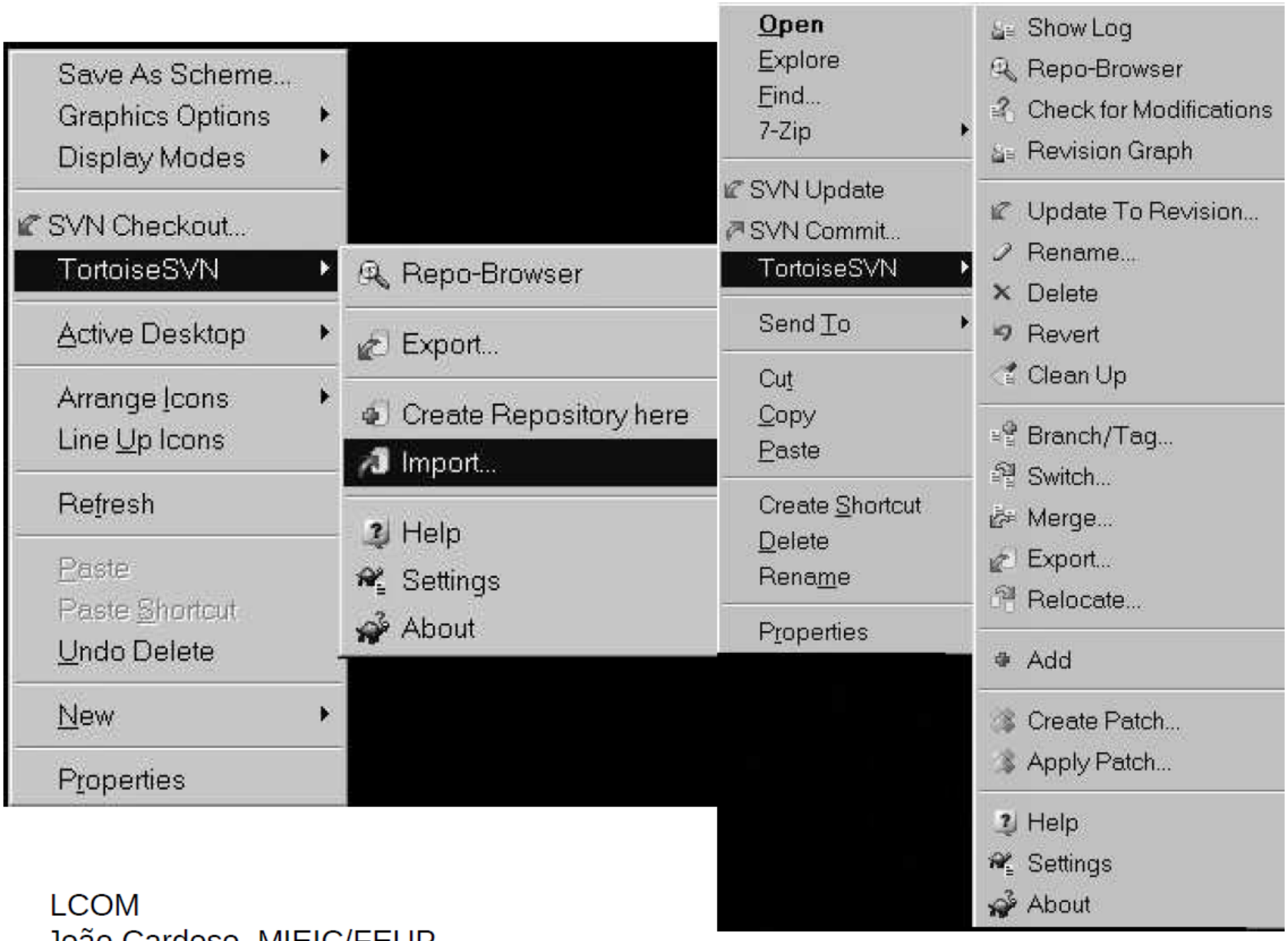
- Alternatively, we can use as a table a two-dimensional array, which is indexed not only by the event but also by the state

```
void (*eht[][])(void) = {{s0e0_handler, ...},
                         {s1e0_handler, ...},...};
...
(*eht[st][ev])();  // index into table and call handler
```

Further Reading

- Máquinas de Estado em C

Function Pointers Application: Menus



Class Menu: A class for building menus

```
#include "menu.h"
#include <stdio.h>

void e1() {printf("-e1-\n");        void e2() {printf("-e2-\n");
void se1() {printf("-se1-\n");      void se2() {printf("-se2-\n");
void sse1() {printf("-sse1-\n");    void sse2() {printf("-sse2-\n");

int main() {
    Menu *ssm1 = newMenu("Sub Sub Menu 1");
    menuAddFunction(ssm1, "Sub Sub Entry 1", sse1);
    menuAddFunction(ssm1, "Sub Sub Entry 2", sse2);

    Menu *sm1 = newMenu("Sub Menu 1");
    menuAddFunction(sm1, "Sub Entry 1", se1);
    menuAddFunction(sm1, "Sub Entry 2", se2);
    menuAddMenu(sm1, "Sub Sub Menu 1", ssm1);

    Menu *m1 = newMenu("Main Menu");
    menuAddFunction(m1, "Entry 1", e1);
    menuAddFunction(m1, "Entry 2", e2);
    menuAddMenu(m1, "Sub Menu 1", sm1);

    menuPost(m1);
    menuDelete(m1); menuDelete(sm1); menuDelete(ssm1);
    return 0;
```

Class Menu: menu.h

```
struct menu;
struct menu_entry;
// handy typedefs
typedef struct menu Menu;
typedef struct menu_entry MenuEntry;
struct menu {
    char *title;           // menu title
    MenuEntry **entries;   // pointer to array of menu entries
    int num                // number of menu entries
    int size;              // array capacity
};
struct menu_entry {
    char *desc;            // menu entry descriptive text
    Menu *subMenu;         // non-NULL if entry is submenu
    void (*func)();        // non-NULL if entry selection calls a
};
Menu * newMenu(char *title); // the "constructor"
void menuDelete(Menu *m);    // destructor
// Other "methods"
void menuAddFunction(Menu *m, char *desc, void (*f)(void));
void menuAddMenu(Menu *m, char *desc, Menu *sm);
void menuPost(Menu *m);   // activate the menu
```

Class Menu: `menu.c` (1/2)

```
Menu *newMenu(char *title) {
    menu *m = malloc(sizeof(Menu)); // missing error checking
    m->title;
    m->num = m->size = 0;
    menuAdjust(m); // if needed increase entries[] size
    return m;
}
void menuAddFunction(Menu *m, char desc, void (*func)()) {
    MenuEntry *me = malloc(sizeof(MenuEntry));
    me->desc = desc;
    me->func = func; me->subMenu = NULL;
    m->entries[m->num++] = me;
    menuAdjust(m);
}
void menuAddMenu(Menu *m, char *desc, Menu *sm) {
    MenuEntry *me = malloc(sizeof(MenuEntry));
    me->desc = desc;
    me->subMenu = sm ; me->func = NULL;
    m->entries[m->num++] = me;
    menuAdjust(m);
}
```

Class Menu: `menu.c` (2/2)

```
void menuPost(Menu *m) {
    int choice;
    char *su = saveUnder(m); // save area under new menu
    while(1) {
        // draw menu and accept user choice
        choice = selectEntry(m);
        if( choice == 0 ) {
            restoreUnder(m, su);
            return;
        }
        if( m->entries[choice-1]->fun != NULL )
            (*(m->entries[choice-1]->fun))(); // call handler
        else // if( m->entries[choice-1]->subMenu != NULL )
            menuPost(m->entries[choice-1]->subMenu); // activate
    }
}
//draw menu, accept user choice
// return index of selected entry (
static int selectEntry(Menu *m) {
    ...
}
```

Thanks to:

I.e. shamelessly translated material on menus by:

- João Cardoso (jcard@fe.up.pt)