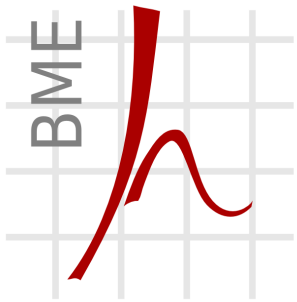
Generic programming – State machines

Basics of Programming 1



Department of Networked Systems and Services
G. Horváth, A.B. Nagy, Z. Zsóka, P. Fiala, A. Vitéz

7 December, 2022

Content

Menu system with function pointers

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```c
void start_game(void) { printf("Game starts...\n"); }
void list_scores(void) { printf("Steve: 10\n"); }
```

```c
typedef struct {
  char command[21];
  void (*func)(void);
} menu_t;
```

```c
menu_t menu[] = {
  {"game",   start_game },
  {"scores", list_scores},
  {"save",   save_game  },
  {"",       NULL/*end*/}
};
```

```c
char command[21];
do {            /* no need to touch it again :) */
  unsigned i;
  printf("Choose: ");
  scanf("%s", command);
  for (i = 0; menu[i].func != NULL; ++i)
    if (strcmp(command, menu[i].command)==0)
      menu[i].func();
} while (strcmp(command, "exit"));
```

link

Chapter 1

Generic algorithm

Motivation

Let us sort 2D points with bubblesort!

```
1  typedef struct { double x, y; } point;
```

```
1  void xchg(point *px, point *py)
2  {
3    point tmp = *px;
4    *px = *py;
5    *py = tmp;
6  }
```

according to coordinate x in an ascending order

```
1  void bubble_point_by_x_asc(point t[], int n)
2  {
3    int iter, i;
4    for (iter = 0; iter < n-1; ++iter)
5      for (i = 0; i < n-iter-1; ++i)
6        if (t[i].x > t[i+1].x)
7          xchg(t+i, t+i+1);
8  }
```

Motivation

There are so many combinations...

```
1 void bubble_point_by_x_asc(point t[], int n);
2 void bubble_point_by_x_desc(point t[], int n);
3 void bubble_point_by_y_asc(point t[], int n);
4 void bubble_point_by_y_desc(point t[], int n);
5 void bubble_point_by_abs_asc(point t[], int n);
6 void bubble_point_by_abs_desc(point t[], int n);
7 void bubble_point_by_angle_asc(point t[], int n);
8 void bubble_point_by_angle_desc(point t[], int n);
```

...and these are only 2D points...

- Let us write a bubble sort algorithm that is independent on the data to sort and the sorting criteria!
- This will be a generic algorithm.

Analysis

What is sorting?

- It is an algorithm consisting of
  - comparisons
  - swaps
- These are the primitives of the algorithm
- The primitives operate on the data, they have to know its type and specifics
- The sorting algorithm itself determines the calling order of the primitives only, independent on the data

Generic algorithm:
Step I.:

- Let us implement the primitives as functions!
  - We have done it with the swapping already (function `xchg`)

Generic sorting

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let us put the comparisons into a separate function!

```
1  int comp_x_asc(point *a, point *b)
2  {
3    return a->x > b->x;
4  }
```

```
1  void bubble_point_by_x_asc(point t[], int n)
2  {
3    int iter, i;
4    for (iter = 0; iter < n-1; ++iter)
5      for (i = 0; i < n-iter-1; ++i)
6        if (comp_x_asc(t+i, t+i+1))
7          xchg(t+i, t+i+1);
8  }
```

These primitives will be called by different sorting functions

Generic sorting

All primitives doing comparison look the same way:

```
1 int comp_by_???(point *a, point *b);
```

Let us define a function pointer pointing to such functions

```
1 typedef int (*comp_fp)(point*, point*);
```

The comparison primitive is a parameter of the sorting function

```
1 void bubble_point(point t[], int n, comp_fp comp)
2 {
3   int iter, i;
4   for (iter = 0; iter < n-1; ++iter)
5     for (i = 0; i < n-iter-1; ++i)
6       if (comp(t+i, t+i+1))
7         xchg(t+i, t+i+1);
8 }
```

Pass the apropriate primitive when calling the sorting function

```
1 bubble_point(points, 8, comp_x_asc);
```

Generic sorting

- We need to create a function for comparing two data for every possible sorting criteria
- The bubble sort algorihm, written only once and forever, receives it as a parameter

- Can the `bubble_point` function sort cats according to their age?
- Not yet, unfortunately
- But it will be possible soon!

Generic sorting

Let us change the parameters of the primitives

```
int comp_by_???(point *array, int i, int j) { ... }
void xchg_point(point *array, int i, int j) { ... }
```

The corresponding function pointer types are:

```
typedef int (*comp_fp)(point*, int, int);
typedef void (*xchg_fp)(point*, int, int);
```

Let us pass the exchange primitives as parameters, too

```
void bubble_point(point *t, int n,
                  comp_fp comp, xchg_fp xch) {
  int iter, i;
  for (iter = 0; iter < n-1; ++iter)
    for (i = 0; i < n-iter-1; ++i)
      if (comp(t,i,i+1))
        xch(t, i, i+1);
}
```

Generic sorting

The pointer arithmetic has been moved from the `bubble_point` function to the primitives

It does not have to know the size of the array elements, only the address of the array

The array address is passed as `void *`

```
1  void bubble(void *t,int n, comp_fp comp,xchg_fp xch) {
2    int iter, i;
3    for (iter = 0; iter < n-1; ++iter)
4      for (i = 0; i < n-iter-1; ++i)
5        if (comp(t, i, i+1))
6          xch(t, i, i+1);
7  }
```

The `bubble` does not know whether it sorts 2D points or cats. This implies that the primitives have to get the array as `void *`, too.

The appropriate function pointer types are:

```
1  typedef int (*comp_fp)(void*, int, int);
2  typedef void (*xchg_fp)(void*, int, int);
```

Generic sorting

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

The primitives know exactly the data they are working with

The `void *` pointer is converted by explicit casting appropriately

```
1  int comp_cat_by_age_asc(void *t, int i, int j)
2  {
3    cat *c = (cat *)t; /* pointer conversion */
4    return c[i].age > c[j].age;
5  }
```

```
1  void xchg_cat(void *t, int i, int j)
2  {
3    cat *c = (cat *)t; /* pointer conversion */
4    cat tmp = c[i];
5    c[i] = c[j];
6    c[j] = tmp;
7  }
```

link

The function call is now fully general

```
1  bubble(cats, 8, comp_cat_by_age_asc, xchg_cat);
2  bubble(dogs, 24, comp_dog_by_name_desc, xchg_dog);
```

Summary

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Generic vector algorithms

- The algorithm receives the input array as `void *`
- The generic algorithm does not use indexing, does not do pointer arithmetics, it just plays with the indexes
- The specialized primitives receive the array as `void *`, and they work with it after explicit type casting

Further simplifications

- The exchange primitive exchanges the data bit-by-bit, we dont even have to implement it for every data type, it is enough to pass the data size only
- Quick sort algorithm in `<stdlib.h>` along the same concept

```
1 void qsort(void *t, size_t n, size_t elem_size,
2            int (*comp)(void*, void*));
```

Remarks

- The pointer conversion involving `void *` is almost hard „hacking”
- This will compile and run without warnings, too:

```
1 Dalmatian doggies[101]; /* 101 dalmatians */
2 bubble(doggies, 101, comp_train_by_length,
3                       xchg_city);
```

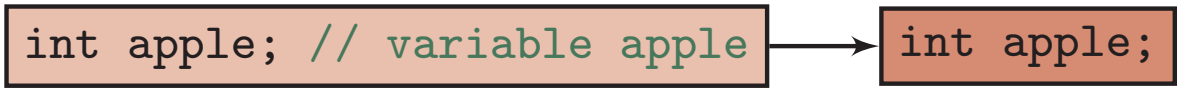
- Watch out, what you are doing!
- We will study a much more elegant approach next semester (using a different language)

Chapter 2

State machines

Motivation

Let us read a text and leave out all C++ comments



- the input is processed character-by-character till the end
- when '/' is detected, we have to wait for one more
- if we are inside a comment, disable the output

Considering the actions to take, our program can be in the following four states:

base	we are not in a comment
slash	one '/' has been detected, we wait for the next one
comm	we are in a comment till the end of the line
end	end of the input text

State machine

We have to specify how to react to the next input character in the different state

- which will be the next state
- the action to take

	'/'		'\n'		EOF		other(ch)	
base	slash	-	base	'\n'	end	-	base	ch
slash	comm	-	base	'/', '\n'	end	'/'	base	'/', ch
comm	comm	-	base	'\n'	end	-	comm	-

Simple implementation in C

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
5  enum state {
6    base, slash, comm
7  } st = base;
8  char ch;
9
10 while(scanf("%c", &ch)==1)
11 {
12   switch(st) {
13   case base:
14     if(ch == '/')
15       st = slash;
16     else
17       printf("%c", ch);
18     break;
19   case slash:
20     if(ch == '/')
21       st = comm;
22     else {
23       printf("/%c", ch);
24       st = base;
25     }
26     break;
27   case comm:
28     if(ch == '\n') {
29       printf("\n");
30       st = base;
31     }
32     break;
33   }
34 }
35 if(st == slash)
36   printf("/");
```

link

Definition of state machines

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

State machine

Event driven programing model based on state variables

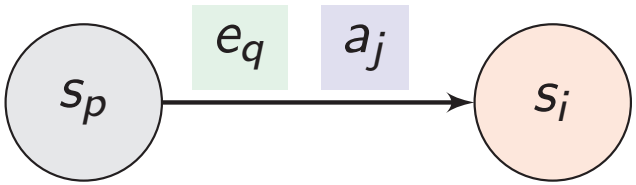
- The program is a finite automaton that changes its state according to its current state and the input
- It executes an action at state transitions

- The elements of state machines:
  - set of states S
  - set of events E
  - set of actions A
- Defining a state machine
  - based on the state transition graph
  - based on the state transition table

Properties of state machines

Requirements:

- fully specified: for all $(s_p, e_q) : s_p \in S, e_q \in E$ pairs the next state $s_i \in S$ and the action $a_j \in A$ has to be specified
- deterministic: s_i and a_j must be uniqely determined for (s_p, e_q)



	...		e_q	...
...	...	...	...	...
s_p	...	s_i	a_j	...
...	...	...	...	...

s_p: current state

e_q: current event

s_i: next state

a_j: action to take

Implementing state machines

- the states can be represented by an enumerated type
- the events can be represented by (an other) enumerated type
- for every action there should be a separate function receiveing a single character
- the content of the table cells are represented by structures (state, data, function pointer)

```
3  enum state {base, slash, comm, end}; /*states*/
4  enum event {slashc, newline, eof, other}; /*events*/
5
6  typedef void (*act)(char);  /* actions */
7
8  typedef struct{      /* one cell of the table */
9    enum state next_state;
10   act action;
11 } cell;
```

Implementing state machines

- the main loop is very simple

```
35  while(st != end)
36  {
37    char ch;
38    /* identify event */
39    enum event ev = next_event(&ch);
40    /* execute action */
41    table[st][ev].action(ch);
42    /* state transition */
43    st = table[st][ev].next_state;
44  }
```

link

- this loop is the same for every state machine
  - the states, events, actions and the table are of course different
  - we assume that the end state is reached eventually

Auxiliarly functions

- identifying the event based on the character read
- the actions are simple functions

```
13 enum event next_event(char *chp)
14 {
15   if (scanf("%c", chp) !=1 ) return eof;
16   if (*chp == '/') return slashc;
17   if (*chp == '\n') return newline;
18   return other;
19 }
20
21 void print(char c)      { printf("%c", c); }
22 void slashch(char c)    { printf("/%c", c);}
23 void slashout(char c)   { printf("/");}
24 void no_output(char c) {}
```

Specifying state machines

	'/'	'\n'	EOF	other(ch)
base	slash -	base '\n'	end -	base ch
slash	comm -	base '/', '\n'	end '/'	base '/', ch
comm	comm -	base '\n'	end -	comm -

- table is mapped to a 2D array in `main()`
- the program starts from state base

```
28  cell table[end+1][other+1] = {  /* the table */
29  {{slash,no_output}, {base,print}, {end,no_output},{base,pri
30  {{comm,no_output},{base,slashch},{end,slashout},{base,slash
31  {{comm,no_output},{base,print}, {end,no_output},{comm,no_ou
32  };
33    enum state st = base;
```

Further examples

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Task: programming the light in the garage
  - The light can be controlled with the manual switch at the entrance
  - There is a switch next to each parking place
  - Lights are switched off automatically after a certain delay

Light control in the garage

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

States

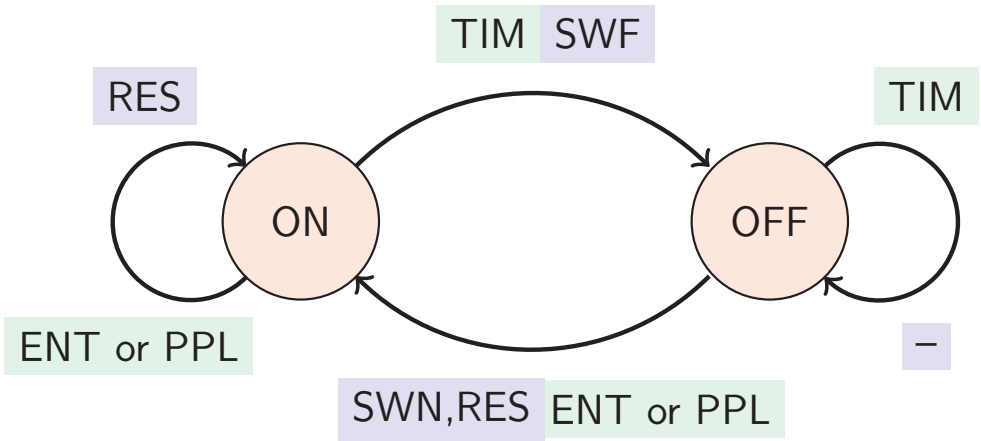
- `ON` switched on
- `OFF` switched off

ACTIONS

- `SWN` switch on light
- `SWF` switch off light
- `RES` reset switch-off timer

Events

- `ENT` button at entrance pressed
- `PPL` button next to a parking place pressed
- `TIM` switch-off timer expired



	ENT		PPL		TIM	
ON	ON	RES	ON	RES	OFF	SWF
OFF	ON	SWN,RES	ON	SWN,RES	OFF	-

Thank you for your attention.