CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
Switch statements
**switch Statement**

**Syntax**

```
switch (expression)
{
    case const-expr₁: statements₁
    :
    case const-exprₙ: statementsₙ
    default: statementsₙ₊₁
}
```

- `expression` is evaluated and tested for a match with the `const-expr` in each `case` clause.
- The `statements` in the matching `case` clause is executed.
switch Statement

Flow Diagram (default)

START

Const-expr₁ = expression?
  YES → statement₁
  NO →

Const-expr₂ = expression?
  YES → statement₂
  NO →

... (repeated)

Const-exprₙ = expression?
  YES → statementₙ
  NO →

statementₙ₊₁

END

Notice that each statement falls through to the next

This is the default behavior of the switch statement
**switch Statement**

Flow Diagram (modified)

```
START

Const-expr_1 = expression?
  YES
  statement_1
  break;
  NO

Const-expr_2 = expression?
  YES
  statement_2
  break;
  NO

... 

Const-expr_n = expression?
  YES
  statement_n
  break;
  NO

statement_n+1

END
```

Adding a `break` statement to each statement block will eliminate fall through, allowing only one case clause's statement block to be executed.
switch Statement

Simple example

switch Example 1

```c
switch (channel) {
    case 2:  puts("WBBM Chicago"); break;
    case 3:  puts("DVD Player"); break;
    case 4:  puts("WTMJ Milwaukee"); break;
    case 5:  puts("WMAQ Chicago"); break;
    case 6:  puts("WITI Milwaukee"); break;
    case 7:  puts("WLS Chicago"); break;
    case 9:  puts("WGN Chicago"); break;
    case 10: puts("WMVS Milwaukee"); break;
    case 11: puts("WTTW Chicago"); break;
    case 12: puts("WISN Milwaukee"); break;
    default: puts("No Signal Available");
}
```
switch Example 1

```c
switch (channel) {
    case 2:
        puts("WBBM Chicago");
        break;
    case 3:
        puts("DVD Player");
        break;
    case 4:
        puts("WTMJ Milwaukee");
        break;
    ...
}
```
switch Statement
With ASCII

switch Example 2

```c
switch (letter) {
  case 'a':
    puts("Letter 'a' found.");
    break;
  case 'b':
    puts("Letter 'b' found.");
    break;
  case 'c':
    puts("Letter 'c' found.");
    break;
  default:
    puts("Letter not in list."); // FATAL ERROR
}
```
switch Statement

Fall-through

switch Example 3

```c
switch(channel) {
    case 4:
    case 5:
    case 6:
    case 7:
        puts("VHF Station");
        break;
    case 9:
    case 10:
    case 11:
    case 12:
        puts("VHF Station");
        break;
    default:
        puts("No Signal Available");
}
```
switch Statement

Range syntax

switch Example 3

switch (channel) {
    case 4 ... 7:
        puts("VHF Station");
        break;
    case 9 ... 12:
        puts("VHF Station");
        break;
    default:
        puts("No Signal Available");
}
# switch Statement

Real-world example

```c
bool IsHex(char character) {
    switch (character) {
    case 'a' ... 'f' :
    case 'A' ... 'F' :
    case '0' ... '9' :
        return true;
    default:
        return false;
    }
}
```
CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
100

State machines
State machines

- Known as Finite State Machines (FSM)
- Mathematical model of computation where system has a single state
- Triggering conditions can change that state
- FSMs are defined completely by both their states and the transitions between them
State machines

State

- The system only exists in one state at a time
- State persists through time
- Certain conditions can change the state to another state
  - These are specific to the current state

Lock/font
Toaster oven

Days
State machines

Transitions

- Events trigger transitions between states
- A combination of events can be used
- Transitions are all mutually exclusive
- At any given time there must be a valid transition for a state
  - If no transition is explicitly stated, an implied loopback transition exists
State machines

Benefits

• Provides a formal way to reason about a system
  – Allows for testing before writing any code
• Can be easily visualized
• Are language independent
• States are only dependent on current state and current inputs
State machines

When to use

- Can be used whenever there are a finite set of states for the system
  - Car transmission
  - Stoplight
  - Vending machine
  - Toaster oven
  - Video games
State machines

Use in the SeaSlug

- Transmission protocol
  - Mission management
  - Parameter management
- Operating state
  - Handling errors/system faults
- Calibration
  - Rudder
  - Radio controller
State machines

Diagrams

STATE_1

STATE_2

condition2
action2

condition1
action1
State machines

Coding

Example

typedef enum { STATE_1, STATE_2 } SystemState;
static SystemState state;
{
    switch (state) {
    case STATE_1:
    default:
        if (condition1) {
            Action1();
            state = STATE_2;
        }
        break;
    case STATE_2:
        if (condition2) {
            Action2();
            state = STATE_1;
        }
    }
}
State machines

Example

typedef enum { STATE_1, STATE_2 } SystemState;
static SystemState state;
int main (void) {
    // Initialize system

    // Event loop
    while (1) {
        // State machine
        switch (state) {
            ...
        }
    }
}
CMPE-013/L

Linked Lists

Maxwell James Dunne
Linked List
Theory
typedef struct ListItem {
    struct ListItem *previousItem;
    struct ListItem *nextItem;
    char *data;
} ListItem;
Linked List

ListItem *LinkedListNew(char *data);

malloc

\[ \text{data} \]

\[ \text{p} \]

\[ \text{N} \]

\[ \text{D} \]
Linked List

List*ListItem *LinkedListCreateAfter(ListItem *item, char *data);
Linked List

ListItem *LinkedListCreateAfter(ListItem *item, char *data);
Linked List

ListItem *LinkedListCreateAfter(ListItem *item, char *data);
Linked List

char *LinkedListRemove(ListItem *item);

\[\text{item} \rightarrow \text{n} \rightarrow \text{p} = 0;\]

\[\text{item} \rightarrow \text{p} \rightarrow \text{n} = 0\]

\[\text{free}(\text{item})\]
LL Remove

item → p → N = item → N

item → N → p = item → p
Linked List

ListItem *LinkedListGetFirst(ListItem *list);

Loop

list = list->next;
list->p = 0
Linked List

```c
int LinkedListSize(ListItem *list);
int LinkedListPrint(ListItem *list);
```

```
list = Get first

LOOP
  list = list -> n
  list -> n = = 0
```
Selection sort
for length(A) - 2

item \rightarrow N = \text{null}

item \rightarrow N \rightarrow N = \text{null}
Stop overthinking

get size / get first / print

↑

item → PI ≠ NULL

[cat, bird, dog, cat]

Print

P pointer = NULL
test NULL pointer

Heap to zero or very small

/test\*data+9

(strlen) [strcmp] NULL
I → p ∨ n = I → n
I → n ∨ p = I → p
46/111  free(s)
I → data
SS
I \rightarrow N \rightarrow N \neq NULL
M \rightarrow N \neq NULL
L \times I
for(I = LLGS(List); I \rightarrow N \neq NULL; I := I \rightarrow N)