Leibniz Universität Hannover

Programmieren 1

Auditorium Exercise 12

Human-Computer Interaction Group

Jan Feuchter, Tim Dünte
programmieren1@hci.uni-hannover.de

Weiterer Verlauf Übungen

	Ausgabe	Abgabe	Besprechung
Ü10	16.12.	22.12.	09.01.-12.01.
Ü11	22.12.	12.01.	13.01.-19.01.
Ü12	13.01.	19.01.	23.01.-26.01.
Ü13 (optional, Klausurvorbereitung)	20.01.	-	-
letzte Vorlesung 27.01.			

Heute und nächste Woche letzte Präsenzübung

Keine Präsenzübung mehr am 27.01.

Klausur

- Zeitraum: 14.03. – 16.03.
- Im Rechnerraum: Hauptgebäude F411
  - Hier finden auch die Präsenztutorien statt
- Zeitslots über die 3 Tage verteilt
  - Zuteilung erfolgt demnächst per StudIP

Leibniz Universität Hannover

Assignment 12

TEXT STATISTIC

Text Statistic: Typedef for Function Pointer

- `typedef bool (*CharacterTestFunction)(char);`
  - Return a Boolean
  - Expect a single Character as the sole parameter

- Not strictly necessary
- One could always explicitly state the type for every pointer
  - E.g. `bool (*fn)(char);`
- But the alternative is much nicer to read
  - E.g. `CharacterTestFunction fn;`

Text Statistic: Generic Function to generate a statistic

```
int calculate_statistic(const char * text,
                        CharacterTestFunction predicate) {
    int sum = 0;
    for(char c = *text; c != '\0'; c = *++text) {
        if(predicate(c)) sum++;
    }
    return sum;
}
```

- Iterate over string `text`
- Apply function pointer (`predicate`) for every character
- Count instances in which the predicate returns `true`

Text Statistic: Predicate Functions (examples)

```
bool is_letter(char c) {
    return (c >= 'a' && c <= ‘z’) ||
           (c >= 'A' && c <= 'Z’);
}

bool is_digit(char c) {
    return c >= '0' && c <= '9’;
}

bool is_newline(char c) {
    return c == '\n’;
}
```

Text Statistic: Using Predicate Functions

```
TextStatistic make_text_statistic(String text) {

    return (TextStatistic) {
        .chars = calculate_statistic(text, is_valid_char),
        .letters = calculate_statistic(text, is_letter),
        .digits = calculate_statistic(text, is_digit),
        .linebreaks = calculate_statistic(text, is_newline),
        .sentences = calculate_statistic(text, is_sentence_end),
    };

}
```

Leibniz Universität Hannover

Assignment 12

POINTER LIST

Pointer List: Copying and Freeing Items

- These functions need to be generic (i.e. expect a void pointer)
- Duplicating a list needs to duplicate the contents as well
  - Thus: `copy_item`
- Freeing a list needs to free its contents
  - Thus: `free_item`

```
void * copy_item(void * x) {
    Item * item = (Item *)x;
    return new_item(
        item->name,
        item->cat,
        item->price
    );
}

void free_item(void * x) {
    if(x != NULL) {
        Item * item = (Item *)x;
        s_free(item->name);
        free(item);
    }
}
```

Pointer List: Finding an Element via a Predicate Function

```
bool is_electronics(void* element, int i, void* x ) {
    return ((Item *)element)->cat == C_ELECTRONICS;
}

Item * found_item = find_list(list, is_electronics, NULL);
```

- `is_electronics` is called for every item in the list
  - `void* element` points to each individual item
  - `int i` is a counter which is incremented for each item
  - `void* x` points to an optional parameter which can be passed to the predicate (here: NULL)

Pointer List: Mapping a list

```
void * reduce_price(void* element, int i, void* x) {
    Item * item = copy_item(element);
    double factor = *(double *) x;
    item->price = ((1. - factor) * item->price);
    return item;
}

double factor = 0.13;
Node * list2 = map_list(list, reduce_price, &factor);
```

- Map: Create a copy of a list, transform each item using `reduce_price`
- Here, an additional value (`factor`) is passed to the mapping function

Pointer List: Reducing a List

```
void add_prices(void* state, void* element, int index) {
    int * sum = (int *)state;
    Item * i = (Item *)element;
    *sum += i->price;
}

int total_price = 0;
reduce_list(list, add_prices, &total_price);
```

- Iterate over list
- Call `add_prices` function for each item
- Goal: `add_prices` modifies value of state on each call

Pointer List: Custom Compare Function

```
int cmp_item_price(void* x, void* y) {
    Item* a = (Item*) x;
    Item* b = (Item*) y;
    if (a == b) return 0;
    if (a == NULL) return -1;
    if (b == NULL) return 1;
    return a->price - b->price;
}

Node * sorted_list = NULL;
for (Node * n = list; n != NULL; n = n->next) {
    sorted_list = insert_ordered(sorted_list, n->value, cmp_item_price);
}
```

Questions?

Assignment 13

- For self-study purposes
  - No submission possible

- Already available on StudIP