

Data Structures and Algorithms

Exercise 5

Markus Innerebner, Romans Kasperovics
dsa(a)inf.unibz.it

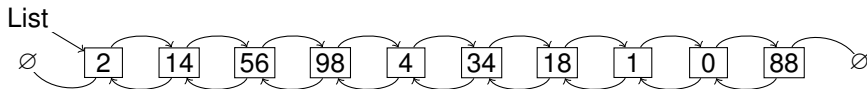
Free University of Bozen - Bolzano, Italy

Thursday, March 26th, 2009

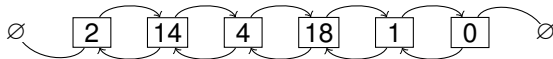
Assignment 05

Task 1

Write a C function `LE* ExtractLess(LE** List, int Max)` that extracts from a double-linked list all elements with key values less than `Max`, and returns a new double-linked list with extracted elements. No additional memory should be allocated. For example, for the list



`ExtractLess(List,30)` returns the list

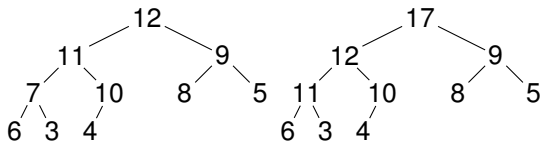


Write a program that initializes a list `List` with the values shown, calls `ExtractLess(List,30)`, and prints the resulting list.

Assignment 05

Task 2

Consider a heap-based implementation of a priority queue. Write a function `void UpdateQueue(int A[], int n, int i, int p)` that for the given array A of size n , index i , and a new priority p , changes the priority of $A[i]$ to p . If after the assignment, the array A does not hold the heap properties, the function reorders A so that the heap properties are restored. Your function should run in $O(\lg n)$ time. For example, the following figure shows the initial queue (left) and the queue after updating 7 to 17 (right).



Use the example above to test your implementation. Print the queue before and after update.

Hints

Single-linked List

```
/* declaring single-linked list element structure */
struct le {
    int key;
    struct le* next;
}; // semicolon here is important!
typedef struct le LE; // now 'LE' is a shorthand for 'struct le'

int main() {
    /* declaring pointer to single-linked lists */
    LE* head = 0; // points to "nowhere"
    LE* tmp; // uninitialized
    ...
    // doing something
    ...
    return 0;
}
```

Hints

Single-linked List

```
/* Creating a list */
int i;
for (i=1; i≤10; i++) {
    // asking for a new piece of memory
    tmp = (LE*) malloc(sizeof(LE));
    // assigning value
    tmp→key = i;
    // attaching to the list
    tmp→next = head;
    head = tmp;
}
/* Printing a list */
tmp = head;
while (tmp ≠0) {
    printf("%d ", tmp→key);
    tmp = tmp→next;
}
printf("\n");
```

Hints

Single-linked List

```
/* Deleting a list */  
while (head  $\neq$  0) {  
    // detaching from the list  
    tmp = head;  
    head = head→next;  
    // returning (freeing) back a piece of memory  
    free(tmp);  
}
```

By Value

Allows us to change parameters without changing original objects.

```
void giveMeTen(int a) {  
    a = 10;  
    return;  
}  
int main() {  
    int a = 15;  
    giveMeTen(a);  
    printf("%i\n", a);  
}  
  
(prints 15)
```

By Reference

Allows us to change original objects from subroutines.

```
void giveMeTen(int* a) {  
    *a = 10;  
    return;  
}  
int main() {  
    int a = 15;  
    giveMeTen(&a);  
    printf("%i\n", a);  
}  
  
(prints 10)
```

- To pass a parameter by reference you pass a pointer to it
- How do we pass pointers by reference?

| by value | by reference |
|-----------------|---------------------|
| int i | int* i |
| char c | char* c |
| LE* head | LE** head |

```
void swap(LE** l1, LE** l2) {  
    LE* tmp = *l1;  
    *l1 = *l2;  
    *l2 = tmp;  
}
```