

Chapter 13

Dynamic memory

13.1 Problems of local variables

- Data structures are usually of a fixed size
 - During runtime usually unknown how much data might be stored.
- Values returned by procedures must be copied into the stackframe of caller
 - Impossible for procedures that return large datastructures (eg read an image)

13.2 Heap and Stack

- Local variable stored in stackframe
- Pool for global variables: Heap
- Values can be stored in heap instead of stackframe
 - Independent of procedure call
 - Must be allocated explicitly
 - Must also be released to reuse occupied memory segment

13.3 Allocate memory on heap

- `malloc(size)` allocates `size` bytes on heap
 - Returns pointer to allocated memory segment
 - In case of an error (eg not enough memory available) returns `NULL`.
 - Use `sizeof` to determine size in bytes of allocated datastructure.

```
int *onheap = (int *) malloc(sizeof(int));  
// depending on the system, sizeof(int) will  
// most likely be 4.
```

- Once allocated, it is not possible to ask how much memory was allocated!

13.4 Free memory

- Once memory was allocated on the heap, it is kept until it is explicitly freed
 - If pointer to allocated memory is lost, memory cannot be released anymore
 - Such errors eat up memory and cause crashes (*memory leak*)
- Free memory using `free(address)`.

```
int *onheap = (int *) malloc(sizeof(int));
/*...*/
free(onheap);
```

- Programming languages like Java automatically free unused memory.

13.5 Possible mistakes

- Memory leaks are complex errors
 - Very hard to avoid and detect.
- Further mistakes:
 - Double free (freeing an already freed memory segment)
- Special software to detect such mistakes
 - `valgrind` in Linux/MacOSX
 - DrMemory for Windows

13.6 Memory allocation functions

- In `stdlib.h`
- Allocate memory
 - `malloc(size)` allocates memory
 - `calloc(size)` allocates memory and initializes with 0.
- Changing an allocation
 - `realloc(old_pointer, size)` changes the size of an allocation to size. If necessary, a new memory segment is allocated, data is copied and the old allocation is freed.
- In case of an error (eg out of memory) return 0

13.7 Exercises

- Implement a procedure that takes a string as an argument and returns the string in reversed order.

13.8 Solution

```
char *reverse(char *s) {
    int len = strlen(s);
    char *ret = (char *)malloc((len + 1) * sizeof(char));
    if(ret != NULL) {
        for(int i = 0; i < len; ++i) {
            ret[i] = s[len - i - 1];
        }
        ret[len] = '\0';
    }
    return ret;
}
```

- Caller must free the returned data structure!