C

Dynamic data structures

Patrick Stiller

November 24, 2014

Memory allocation

- Definition
- Allocation commandos
- Other Commandos

Dynamic data structures

- Linked lists
- Trees

Definition

Definition: "C dynamic memory allocation refers to performing manual memory management for dynamic memory allocation in the C programming language via a group of functions in the C standard library, namely malloc, realloc, calloc and free."

Source: http://en.wikipedia.org/wiki/C_dynamic_memory_allocation

Malloc

malloc() : Allocates a block of size bytes of memory, returning a pointer to the beginning of the block.

Sizeof

sizeof(): Returns the number of bytes , which are reserved by a datatype.

Programstructure

Use a combination of malloc() and sizeof():

```
1 malloc(sizeof(data type));
2
3 // example : an int array with 5 elements
4
5 int * pointerToInt;
6
7 pointerToInt =malloc(5 *(sizeof(int));
```

free()

"The free() function frees the memory space pointed to by pointer, which must have been returned by a previous call to malloc()."

Source:http://linux.die.net/man/3/malloc
Use following programstructure:

```
1  free(pointer);
2
3  //example
4
5  int * pointerToInt;
6
7  pointerToInt =malloc(5 *(sizeof(int));
8  free(pointerToInt);
```

Other allocation-commandos are calloc() and realloc(). For some informations take a look to the *man* page of malloc().

OR visit :

http://linux.die.net/man/3/ malloc

calloc()

Allocates a block of size bytes of memory, returning a pointer to the beginning of the block.

```
1 calloc(size_t num ,size_t size);
```

Example

```
1 int * array ;
2 array = (int*) calloc(5,sizeof(int));
3 // creates an array with five elements
```

List Element

Definition:

```
1  struct listelement{
2
3  type Entry;
4  ...
5  struct listelement* next; // important points to the
       next element
6
7  };
```

First Element

```
1  struct listElement {
2  int value ;
3  struct listelement* next ;
4  };
5
6  int main(int argc, char *argv[]){
7  struct listElement* firstElement ; //pointer of first
       element ;
8  firstElement = (struct listElement*)malloc
9                 (sizeof(struct listelement));
10
11
12 }
```

(struct listElement*)-transform the return value of malloc into a pointer that exhibits to an list element

Access to the Elements

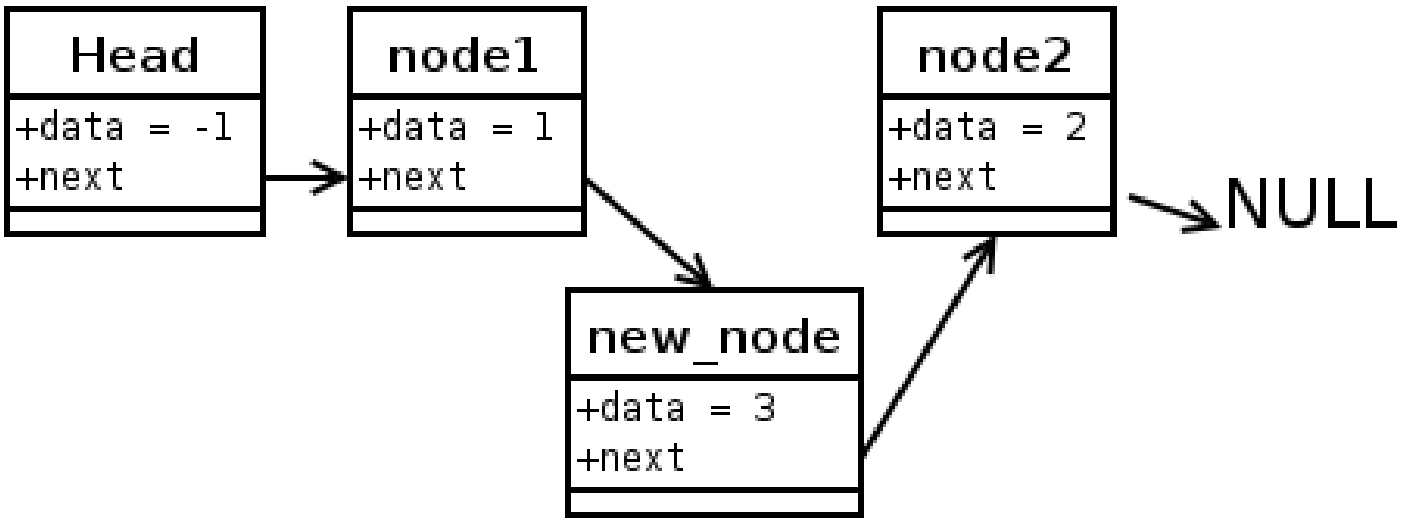
```
1  struct listElement {
2  int value ;
3  struct listelement* next ;
4  };
5
6  int main(int argc, char *argv[]){
7  struct listElement* firstElement;
8   //pointer of first element;
9  firstElement = (struct listElement*)malloc(
10                  sizeof(struct listelement));
11 firstElement ->value = 3;
12 firstElement ->next = NULL;
```

firstElement ->value is equivalent to the dereference-operator(*) and to the statement (*firstelement).value = 3

Add one Element

```
1
2  //create a new Element ;
3
4  newElement = (struct listelement*)malloc(
5  sizeof(struct listelement));
6  newElement->value = 4;
7  newElement->next = NULL;
8
9  //connect two Elements
10
11 firstElement->next = newElement;
```

Illustration



source: http://perlgeek.de/images/dmisc/sll04.png

Definition

You can define you struct arbitaryly. So you are able to define a binary tree.

```
1 struct binaryTreeElement{
2
3 type Entry;
4 ...
5 struct binaryTreeElement* left;
6 struct binaryTreeElement* right;
7 }
```

Illustation