Prep C
Datastructures

Dr. Madhav Rao

Revision Date: July 28, 2016

Basic Idea of Structures

In general, the public interface of a class X is placed in a file named X.h. The public interface consists of the structure that encapsulates the components of the class and the prototypes of the public functions (methods) that (typically) operate upon the structure. The public interface also includes external declarations of any components that are shared between objects of a class. The implementation of the class X is placed in a file named X.c. The implementation consists of the function definitions and variable declarations that were mentioned in the public interface along with definitions and declarations that are private (i.e., not visible outside the class).

Write the following declarations in *node.h* file.

```
typedef struct nodeobject
{
char *type;

int ival;
double rval;    /* the value part of the node */
char *sval;

} node;

extern char *INTEGER;
extern char *REAL;
extern char *STRING;

extern node *newIntegerNode(int value);
extern node *newRealNode(double value);
extern node *newStringNode(char *value);
```

Write the following program in *node.c* file.

```
#include <stdio.h>
#include <stdlib.h>
#include "node.h"

static node *newNode(void);

/*************** public interface *************/
```

```
char *INTEGER = "integer";
char *REAL = "real";
char *STRING = "string";

node *
newIntegerNode(int v)
{
node *p = newNode();
p->type = INTEGER;
p->ival = v;
return p;
}

node *
newRealNode(double v)
{
node *p = newNode();
p->type = REAL;
p->rval = v;
return p;
}

node *
newStringNode(char *v)
{
node *p = newNode();
p->type = STRING;
p->sval = v;
return p;
}

/*************** private methods *************/

static node *
newNode()
{
node *n = (node *) malloc(sizeof(node));
if (n == 0) { fprintf(stderr,"out of memory"); exit(-1); }
return n;
}
```

Write your main fuction in *public.c* file and test the public interface functions: *newIntegerNode*, *newRealNode*, and *newStringNode*. Remember that you will need to compile the code using *makefile*. Also note that you have created a node using *malloc* function. The memory allocation needs to freed once the purpose of the program is completed. Hence define and test a *deleteNode* function which has to be a public-interface function.

Function pointer in the structure

Add a function pointer *display* as a member to your *node* class. Initialize the display member to a static private function *print*. You are supposed to develop a single private function *print* which takes one formal parameter of *struct nodeobject* and will print the respective values of the class *nodeobject*.

Similarly add *destructor* function pointer as a member to your *node* class which will free the memory allocated

to nodeobject. Note that you will have to modify the declaration of *deleteNode* function. Test your destructor member and operations related to this.

Similarly add a *child* member of node class as shown below:

```
typedef struct nodeobject
{
char *type;
int ival;
.........
.........
.........
.........
struct nodeobject *child;
} node;
```

Note that child has inherited similar properties to that of parent *node* class. Properties in this case, means same number of members and methods. However the child pointer remains an unallocated memory pointer. Test whether you can have child of different types "STRING" to that of parent type "REAL". Modify your display function to display all inherited values.

Linked list

Add a *next* member of node class as shown below:

```
typedef struct nodeobject
{
char *type;
int ival;
.........
.........
.........
.........
struct nodeobject *child;
struct nodeobject *next;
} node;
```

Make a linklist using the *next* member variable. Change the newIntegerNode(), newRealNode() and newStringNode() such that the next member is intialized to previous struct node variable or to 0. Make sure that you modify your *display* function to consider *child* and *next* members.