eric data structures in C

ember 30, 2010 (http://andreinc.net/2010/09/30/generic-data-structures-in-c/) Andrei Ciobanu ndreinc.net/author/admin/) C (http://andreinc.net/category/programming-languages/c/), Data

my code . If you see any problems let me know .

to prove that generic programming (http://en.wikipedia.org/wiki/Generic_programming) (the st puter programming in which algorithms are written in terms of *to-be-specified-later* types that are then*instantiated* when needed for specific types pro parameters) can be achieved in **C**, let's write the implementation of a generic **Stack (ht en.wikipedia.org/wiki/Stack_(data_structure))** data structure (htt en.wikipedia.org/wiki/Stack_(data_structure)) . We w l follow two possible approaches:

- *acking* with the #preprocessor;
- sing the flexibility of the void pointer (void*);

You ca always try both of the approaches and see which one is more suitable for your particular case. Also note th

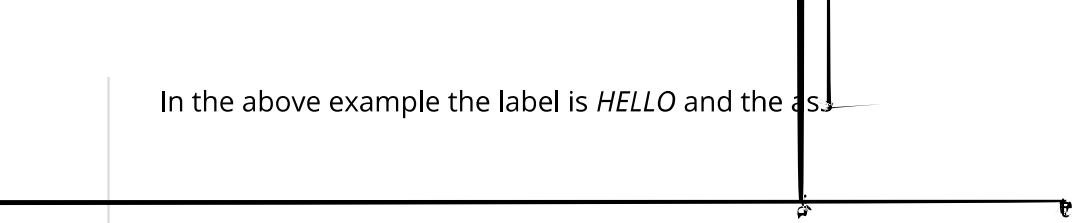
s approach you will need to be familiar with **C macros** ncepts: *function-like macros*; *macro a*

ros.html). If you already know your stuff, you can skip the h your memory / find errors and correct me :P).

counters it,

```
1 #include <stdio.h>
2 #define HELLO "Hello World Macro!"
3 int main(){
4     printf(HELLO);
5     return 0;
6 }
```

In the above example the label is *HELLO* and the a s.



```
ck_##type##_s *next;
 stack_##type ;void stack_##type##_push(stack_##type **stack, type data);

ype stack_##type##_pop(stack_##type **stack);
```

```
/* Expansion if int is supplied as the macro argument : STACK_DECLARE(int) */
ypedef struct stack_int_s {
nt data;
truct stack_int_s *next;
 stack_int;
oid stack_int_push(stack_int **stack, int data);
nt stack_int_pop(stack_int **stack);/* Expansion if double is supplied as the macro ar
ypedef struct stack_double_s {
ouble data;
truct stack_double_s *next;
}
```

on:

g generic functions into macros

```
#define STACK_TYPE(type)
stack_##type#define STACK_DATA(stack)
(stack)->data

#define STACK_PUSH(type, stack, data)
stack_##type##_push(stack, data)

#define STACK_POP(type, stack)
stack_##type##_pop(stack)
```

Step 4 – Putting all togheter

Now let's build an example *against* the newly built generic data structure. We will use two different stacks: one that holds doubles, and another that holds only integers. Then we will push / pop the numbers from [1..100] :

```
#include <stdio.h>
#include <stdlib.h>#define STACK_DECLARE(type)
typedef struct stack_##type##_s {
type data;
struct stack_##type##_s *next;
} stack_##type ;

void stack_##type##_push(stack_##type **stack, type data);

type stack_##type##_pop(stack_##type **stack);

#define STACK_DEFINE(type)
void stack_##type##_push(stack_##type **stack, type data) {
stack_##type * new_node = malloc(sizeof(*new_node));
if (NULL == new_node) {
fputs("Couldn't allocate memoryn", stderr);
abort();
}
new_node->data = data;
new_node->next = *stack;
*stack = new_node;
}

type stack_##type##_pop(stack_##
if (NULL == stack || NULL == *
fputs("Stack underflow.n",
abort();
}
stack_##type *top = *s
type value = top->data
```

```
/* If you w t to work with a stack that holds integers you should
* use those macros. They will expand and the associated functions will be
* generated .
*/
STACK_DECLARE(int)
STACK_DEFINE(int)
STA  DECLARE(double)
STA  _DEFINE(double)

int main(int argc, char** argv) {
int i;

/* New stack . Alaways assign NULL */
STACK_TYPE(int)


                100; ++i) {
printf("PUSH: %dn", i);
STACK_PUSH(int, &amp;amp;st, i);
STACK_PUSH(double, &amp;amp;st2, i);
}

while (i-- > 0) {
printf("POP: %d %2.2fn", STACK_POP(int, &amp;amp;st),
STACK_POP(double, &amp;amp;st2));
}
return (0);
}
```

```
typedef struct stack_s {
void *data; /* Can hold any type of pointer */
struct stack_s *next; /* The stack is built as a linked list */
} stack;
```

```
        printf("%d ", *tmp);
    free(tmp);
    }

    return (0);
    }
```

```c
#include <stdio.h>
#include <stdlib.h>typedef struct stack_s {
void *data; /* Can hold any type of pointer */
struct stack_s *next; /* The stack is built as a linked list */
} stack;

void stack_push(stack **head, void *data);
void *stack_pop(stack **head);

void stack_push(stack **head, void *data) {
stack *new_node = malloc(sizeof (*new_node));
if (NULL == new_node) {
fputs("Couldn't allocate memoryn", stderr);
abort();
}
new_node->data = data;
new_node->next = *head;
*head = new_node;
}

void *stack_pop(stack **head) {
stack *top;
void *value;
if (NULL == head || NULL == *head) {
fputs("Stack underflow.n", stderr);
abort();
}
top = *head;
value = top->data;
*head = top->next;
free(top);
return value;
}

int main() {
stack *s = NULL;
int i, *tmp;

/* Add values from [1..100] into the stack */
printf("Pushing: n");
for (i = 0; i < 100; ++i) {
tmp =_malloc(sizeof (*tmp))
```

🙂

Legend

And I love it .

Related Posts via Categories

- Implementing a generic Priority Queue in C (using heaps (http://andreinc.net/2011/06/01/implementing-a-generic
- Sieve of Eratosthenes (finding all prime numbers up to a (http://andreinc.net/2010/12/12/sieve-of-eratosthenes-fi specific-integer/)
- Binary GCD (Stein's Algorithm) in C (http://andreinc.net/2 algorithm-in-c/)
- Euclid's Algorithm: Reducing fraction to lowest terms (ht algorithm-reducing-fraction-to-lowest-terms/)
- Euclid's Algorithm (http://andreinc.net/2010/12/11/euclid
- Credit Card Validation (Python 3.x) (http://andreinc.net/2
- RPN Calculator (using Scala, python and Java) (http://and using-python-scala-and-java/)
- Insertion Sort Algorithm (http://andreinc.net/2010/12/26
- Bottom-up Merge Sort (non-recursive) (http://andreinc.n non-recursive/)
- The merge sort algorithm (implementation in Java) (http sort-algorithm-implementation-in-java/)

advanced (http://andreinc.net/tag/advanced/) advanced c (http://and

c (http://andreinc.net/tag/c-2/) c