Προγραμματισμός ΙΙ

Δομές Δεδομένων

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Δομές Δεδομένων

Μια *δομή δεδομένων* είναι μια συλλογή δεδομένων με κάποιες ιδιότητες η οποία προσφέρει εύκολη πρόσβαση σε αυτά τα δεδομένα.

Έχουμε δει ως τώρα μια στατική δομή δεδομένων, τον **πίνακα**.

Τι κάνουμε όμως στην περίπτωση που θέλουμε να αποθηκεύουμε ένα συνεχώς μεταβαλλόμενο αριθμό από δεδομένα;

Δυναμικές Δομές Δεδομένων

Είναι δομές δεδομένων που μπορούν να αυξομειώνουν το μέγεθος τους, δηλαδή να αναπτύσσονται και να συρρικνώνονται στον χρόνο εκτέλεσης.

Η C ως γλώσσα προγραμματισμού δεν παρέχει τέτοιες δομές. Μας παρέχει όμως τα απαραίτητα εργαλεία για να φτιάξουμε τέτοιες δομές δεδομένων.

Απαραίτητα Εργαλεία

1. structs - εγγραφές
2. δυναμική καταχώρηση μνήμης - malloc, free
3. δείκτες - pointers
4. μετονομασία τύπων - typedef

Για να φτιάξουμε δυναμικές δομές δεδομένων χρειάζεται να χρησιμοποιήσουμε τα πιο σημαντικά και συχνά πιο δυσκολονόητα κομμάτια της γλώσσας.

Διάφορες Δομές Δεδομένων

Θα δούμε δύο από τις πιο σημαντικές δομές δεδομένων

1. συνδεδεμένες λίστες
2. στοίβες

Ανάλογα με τις ανάγκες της εφαρμογής χρησιμοποιούμε διαφορετική δομή.

Αυτοαναφερόμενες Εγγραφές

Η C μας παρέχει την δυνατότητα να φτιάξουμε εγγραφές (structs) οι οποίες αυτοαναφέρονται:

```
struct node {
    int data;
    struct node *next;
};
```

Η δομή `node` περιέχει έναν ακέραιο και ένα δείκτη τύπου `node`.

Αυτοαναφερόμενες Δομές

Ουσιαστικά η C μας επιτρέπει να δηλώσουμε μια μεταβλητή δείκτη σε τύπο που δεν έχει οριστεί ακόμη. Για παράδειγμα το παρακάτω πρόγραμμα είναι απόλυτα σωστό.

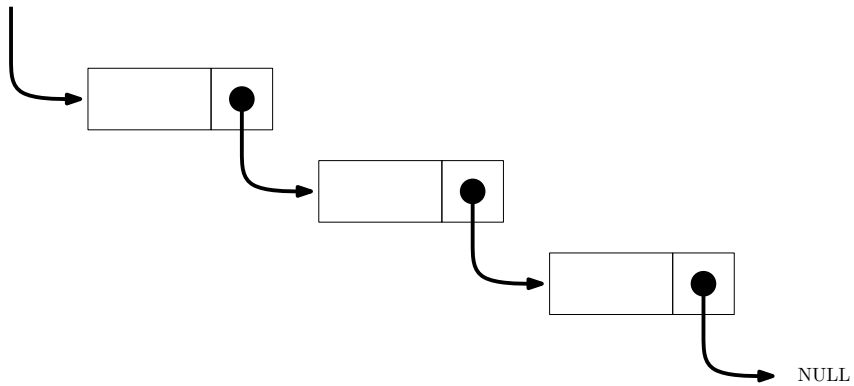
```
#include <stdio.h>

int main()
{
    struct node *ptr;
}
```

Προσέξτε πως στο παραπάνω πρόγραμμα δεν δηλώνουμε ποτέ την εγγραφή **node**.

Συνδεδεμένη Λίστα

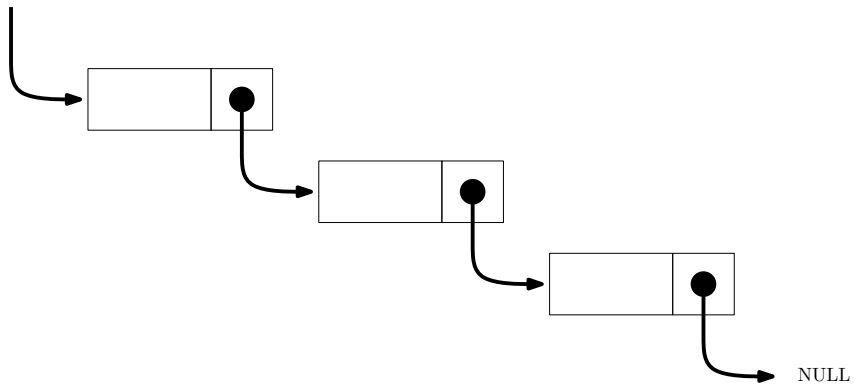
Μια γραμμική συλλογή αυτοαναφερόμενων εγγραφών, που ονομάζονται κόμβοι, και συνδέονται από συνδέσεις δεικτών.



Μια συνδεδεμένη λίστα προσπελαύνεται μέσω ενός δείκτη στον πρώτο κόμβο της λίστας. Κατά σύμβαση, ο δείκτης σύνδεσης στον τελευταίο κόμβο της λίστας τίθεται σε `NULL` για να σηματοδοτήσει το τέλος της λίστας.

Συνδεδεμένη Λίστα

Η πληροφορία σε κάθε κόμβο μπορεί να είναι οποιαδήποτε τύπου, ακόμη και `struct`.



Συνδεδεμένη Λίστα

Κόμβος

```
typedef
struct {
    int AM;
    char name[50];
}
student;

struct node {
    student  data;
    struct node  *next;
};
```

Ένας κόμβος λίστας που περιέχει πληροφορίες για έναν φοιτητή.

Υλοποίηση Λίστας

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <assert.h>
4
5  typedef struct _node* node;
6
7  struct _node {
8      int data;
9      node  next;
10 };
```

Υλοποίηση Λίστας

```
11
12  typedef struct _list* list;
13
14  struct _list {
15      node head;
16      int size;
17  };
18
19  /* create list */
20  list  list_create() {
21      list l = (list) malloc(sizeof(struct _list));
22      assert(l);
23      l->head = NULL;
24      l->size = 0;
25      return l;
26  }
```

Υλοποίηση Λίστας

```
27  /* create node */
28  node list_newnode(int data) {
29      node n = (node) malloc(sizeof(struct _node));
30      assert(n);
31      n->data = data;
32      n->next = NULL;
33      return n;
34  }
35
36  /* free node memory */
37  void list_freenode(node n) {
38      assert(n);
39      free(n);
40  }
```

Υλοποίηση Λίστας

```
41  /* get first list node */
42  node list_first(list l) {
43      assert( l );
44      return l->head;
45  }
46
47  /* get next list node */
48  node list_next(node n) {
49      assert(n);
50      return n->next;
51  }
52
53  /* get node data */
54  int list_data(node n) {
55      assert(n);
56      return n->data;
57  }
```

Υλοποίηση Λίστας

```
58  /* check if list is empty */
59  int list_empty(list l) {
60      assert(l);
61      return l->head == NULL;
62  }
63
64  /* get list size */
65  int list_size(list l) {
66      assert(l);
67      return l->size;
68  }
```

Υλοποίηση Λίστας

```
69  /* insert as first node in list */
70  void list_insertfirst(list l, node n) {
71      assert(l && n);
72      n->next = l->head;
73      l->head = n;
74      l->size++;
75  }
76
77  /* delete first node in list */
78  node list_deletefirst(list l) {
79      assert(l && l->head);
80      l->head = l->head->next;
81      l->size--;
82      ret->next = NULL;
83      return ret;
84  }
```

Υλοποίηση Λίστας

```
85  /* insert after node in list */
86  void list_insertafter(list l, node after, node n) {
87      assert(l && after && n);
88      n->next = after->next;
89      after->next = n;
90      l->size++;
91  }
92
93  /* delete a node after a node */
94  node list_deleteafter(list l, node prev) {
95      assert(l && prev && prev->next);
96      node ret = prev->next;
97      prev->next = ret->next;
98      ret->next = NULL;
99      return ret;
100 }
```

Υλοποίηση Λίστας

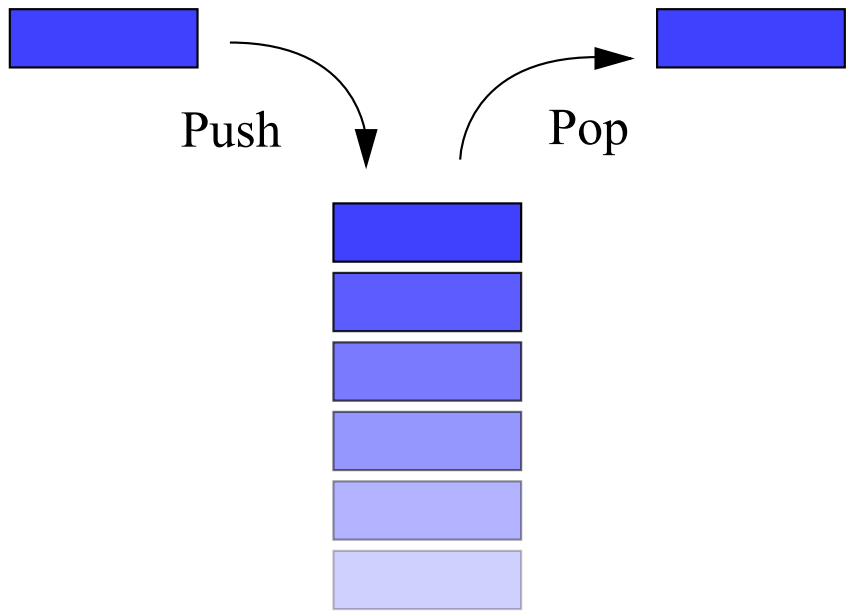
```
101  /* destroy list and free memory */
102  void list_destroy(list l) {
103      while(!list_empty(l))
104          list_freenode(list_deletefirst(l));
105
106      free(l);
107  }
```

Υλοποίηση Λίστας

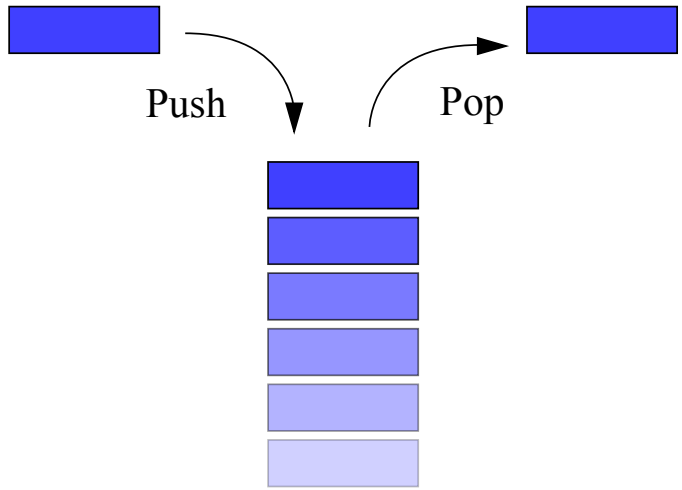
```
108 int main()
109 {
110     int am;
111     node n;
112     list l = list_create();
113 
114     do {
115         printf("Give a student AM (-1 to quit): ");
116         scanf("%d", &am);
117 
118         if (am >= 0) {
119             list_insertfirst(l, list_newnode(am));
120         }
121     } while(am >= 0);
122 
123     for(n = list_first(l); n != NULL; n = list_next(n))
124         printf("%d ", list_data(n));
125     printf("\n");
126 
127     list_destroy(l);
128     return 0;
129 }
```

Στοίβα

Η στοίβα είναι μια δυναμική δομή δεδομένων όπου μπορούμε να έχουμε πρόσβαση μόνο στο τελευταίο στοιχείο που εισήχθει (την λεγόμενη κορυφή της στοίβας).



Λειτουργίες Στοίβας



Μια στοίβα παρέχει τις λειτουργίες

- `PUSH( S, x )`
  - Προσθέτει ένα στοιχείο `x` στην στοίβα `S`.
- `POP( S )`
  - Διαγράφει το νεότερο στοιχείο που βρίσκεται στην στοίβα `S`.
- `TOP( S )`
  - Επιστρέφει το νεότερο στοιχείο που βρίσκεται στην στοίβα `S`.

Υλοποίηση Στοίβας

Μπορούμε να υλοποιήσουμε μια στοίβα με την ίδια τεχνική που χρησιμοποιήσαμε για την λίστα.

Υλοποίηση Στοίβας

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <assert.h>
4
5  typedef struct _node* node;
6
7  struct _node {
8      char data;
9      node  next;
10 };
```

Υλοποίηση Στοίβας

```
11
12  /* stack definition */
13  typedef struct _stack* stack;
14
15  struct _stack {
16      node head;
17      int size;
18  };
19
20  /* create stack */
21  stack stack_create() {
22      stack s = (stack) malloc(sizeof(struct _stack));
23      assert(s);
24      s->head = NULL;
25      s->size = 0;
26      return s;
27  }
```

Υλοποίηση Στοίβας

```
28  /* check if stack is empty */
29  int stack_empty(stack s) {
30      assert(s);
31      return s->head == NULL;
32  }
33
34  /* get stack size */
35  int stack_size(stack s) {
36      assert(s);
37      return s->size;
38  }
39
40  /* get newest stack element */
41  char stack_top(stack s) {
42      assert(s && s->head);
43      return s->head->data;
44  }
```

Υλοποίηση Στοίβας

```
45  void stack_push(stack s, char data) {
46      assert(s);
47      node n = (node) malloc(sizeof(struct _node));
48      assert(n);
49      n->data = data;
50      n->next = s->head;
51      s->head = n;
52      s->size++;
53  }
54
55  void stack_pop(stack s) {
56      assert(s && s->head);
57
58      node ret = s->head;
59      s->head = s->head->next;
60      s->size--;
61
62      free(ret);
63  }
```

Υλοποίηση Στοίβας

```
64  /* destroy stack and free memory */
65  void stack_destroy(stack s) {
66      while(!stack_empty(s))
67          stack_pop(s);
68
69      free(s);
70  }
```

Υλοποίηση Στοίβας

```
71  int main()
72  {
73      int am;
74      char c;
75      node n;
76      stack s = stack_create();
77
78      while((c=getchar()) != EOF)
79          stack_push(s, c);
80
81      while(!stack_empty(s)) {
82          printf("%c", stack_top(s));
83          stack_pop(s);
84      }
85      stack_destroy(s);
86      return 0;
87  }
```

Άλλες Δομές Δεδομένων

Στα επόμενα έτη θα συναντήσετε μια πληθώρα δομών δεδομένων όπως

- δέντρα
- γραφήματα
- λεξικά
- κ.τ.λ

Κάθε δομή δεδομένων έχει τα πλεονεκτήματα και μειονεκτήματα της.