Διάλεξη 10: Στοίβες:Υλοποίηση & Εφαρμογές

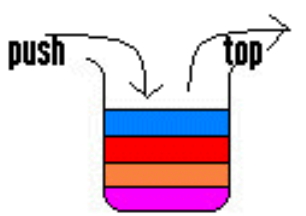
Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Υλοποίηση Στοιβών με Δυναμική Δέσμευση Μνήμης
- Εφαρμογή Στοιβών 1: Αναδρομικές συναρτήσεις
- Εφαρμογή Στοιβών 2: Ισοζυγισμός Παρενθέσεων

Διδάσκων: Παναγιώτης Ανδρέου

ΑΤΔ Στοίβα - Πράξεις

- Θυμηθείτε τον ΑΤΔ στοίβα με τις πράξεις του:



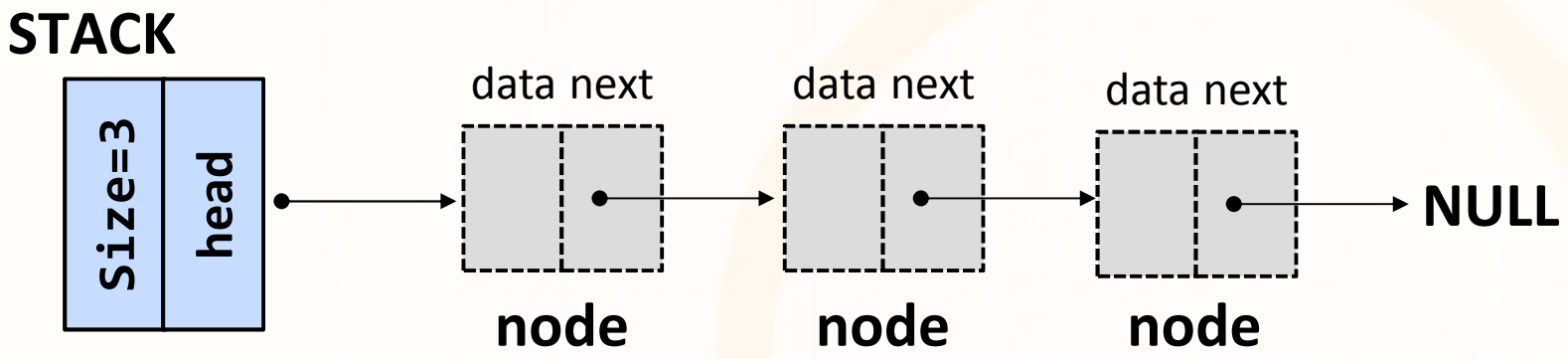
MakeEmptyStack()	δημιούργησε την κενή στοίβα ⟨⟩.
IsEmptyStack(S)	επέστρεψε τη λογική τιμή που εκφράζει το αν η S είναι κενή.
Push(x,S)	εισήγαγε τον κόμβο x στη στοίβα S.
Pop(S)	διέγραψε τον κόμβο κορυφής της S.
Top(S)	δώσε τον κόμβο κορυφής της S.

- Είχαμε πει ότι αυτές οι πράξεις μπορούν να υλοποιηθούν με την στατική δέσμευση μνήμης

```
typedef struct {
   type list[ size ];
   int Length;
} STACK;
```

Στοίβα με Δυναμική Δέσμευση Μνήμης

- Για την παράσταση μιας στοίβας με στοιχεία $\alpha_1, \alpha_2, ..., \alpha_n$ χρησιμοποιούμε μια συνδεδεμένη λίστα από κόμβους.
- Κάθε κόμβος αποτελείται από ένα στοιχείο (στοιχεία της στοίβας) και από ένα δείκτη (προς τον επόμενο κόμβο της στοίβας). Η κορυφή της στοίβας είναι ο πρώτος κόμβος της λίστας,
- Χρησιμοποιούμε μια μεταβλητή για να φυλάγουμε στοιχεία σχετικά με τη στοίβα π.χ. **μέγεθος** (size) και δείκτη προς την **κορυφή της στοίβας (head).**



Στοίβα με Δυναμική Δέσμευση Μνήμης (συν.)

- Υλοποίηση δομής (χρησιμοποιείται `int` αντί για `type`):

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

```
typedef struct stack {
   NODE  *head;
   int   size;
} STACK;
```

- Υλοποίηση πράξεων (χρησιμοποιείται `int` αντί για `type`):

```
void MakeEmptyStack(STACK *S) {
   S->size = 0;
   S->head = NULL;
}
```

```
int IsEmpty(STACK *S) {
   return (S->size == 0);
}
```

```
void Pop(STACK *S) {
NODE *p = NULL;
   if ((S->size) > 0){
      p = S->head;
      S->head = p->next;
      free(p);
      (S->size)--;
   }
}
```

Στοίβα με Δυναμική Δέσμευση Μνήμης (συν.)

```
void Push(STACK *S, int x){
   NODE *p = NULL;
   p = (NODE *)
        malloc(sizeof(NODE));
   p->data = x;
   p->next = S->head;
   S->head = p;
   (S->size) ++;
}
```

```
int Top(STACK *S){
   if ((S->size) > 0){
      return S->head->data;
   }
}
```

ΒΟΗΘΗΤΙΚΗ ΣΥΝΑΡΤΗΣΗ ΓΙΑ ΕΚΤΥΠΩΣΗ

```
void PrintStack(STACK *S){
   NODE *tmp = S->head;
   for(int i=0; i<(S->size); i++){
      printf("%d ", tmp->data);
      tmp = tmp->next;
   }
   printf("\n");
}
```

Στοίβα με Δυναμική Δέσμευση Μνήμης (συν.)

- Παράδειγμα εκτέλεσης:

```
int main(int argc, char* argv[]) {
    STACK stack;
    MakeEmptyStack(&stack);
    // fill stack
    for(int i=0; i<10; i++) {
        Push(&stack, i);
        PrintStack(&stack);
    }
    // print stack
    for(int i=0; i<10; i++) {
        Top(&stack);
        Pop(&stack);
        PrintStack(&stack);
 }
```

Εκτυπώνει

```
0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
6 5 4 3 2 1 0
5 4 3 2 1 0
4 3 2 1 0
3 2 1 0
2 1 0
1 0
0
```

Εφαρμογή Στοίβας 1: Αναδρομικές Διαδικασίες

- Οι στοίβες βρίσκουν μεγάλη χρήση στην πληροφορική για δημιουργία άλλων δομών και σε βασικό λογισμικό.
- Κλασικό παράδειγμα αφορά την **κλήση υποπρογραμμάτων** (function calls) και **αναδρομικών διαδικασιών**.
  - Σε κάθε κλήση οποιασδήποτε συνάρτησης ένα σύνολο από λέξεις (stack frame) φυλάσσεται σε μια στοίβα, από όπου μπορεί να ανασυρθεί.
  - Όταν μια συνάρτηση καλεί μια άλλη συνάρτηση οι **παράμετροι της συνάρτησης**, η **διεύθυνση επιστροφής** και οι **τοπικές μεταβλητές** της καλούσας συνάρτησης φυλάσσονται μέσα στη στοίβα του προγράμματος.
  - Έτσι, όταν η κληθείσα **συνάρτηση τερματίσει**, το περιβάλλον την καλούσας συνάρτησης **ανασύρεται** από τη στοίβα για να συνεχιστεί κανονικά η εκτέλεσή της.

Εφαρμογή Στοίβας 1: Αναδρομικές Διαδικασίες

- Όταν ένα πρόγραμμα φορτώνεται στην μνήμη του υπολογιστή, τότε το η δεσμευμένη μνήμη οργανώνεται σε τρεις περιοχές (segments): *a) text (code) segment*, b) *stack segment*, and c) *heap segment*.
- Το *text segment* περιέχει τον κώδικα υπό εκτέλεση, ενώ το *heap segment* επιτρέπει στον πρόγραμμα να δεσμεύσει μνήμη (με την malloc!). Αυτή η μνήμη παραμένει μέχρι να αποδεσμευτεί ρητά (με την free!) ή να τερματιστεί το πρόγραμμα.
- Το *stack segment* από την άλλη χρησιμεύει για να φυλάγονται όλες οι πληροφορίες σχετικές με την εκτέλεση συναρτήσεων.

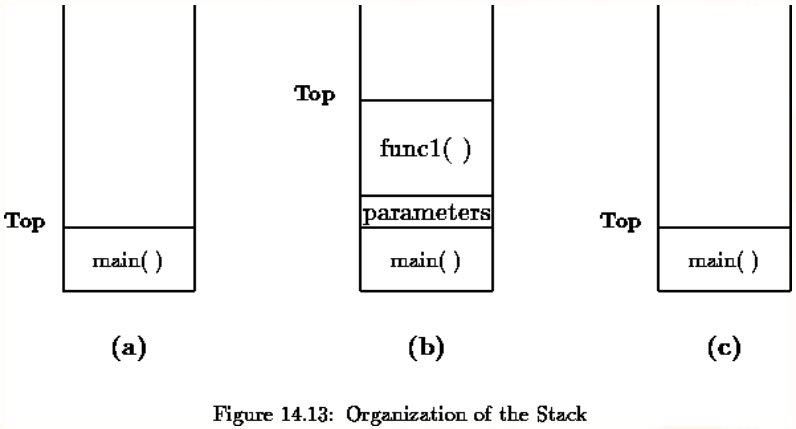


Figure 14.13: Organization of the Stack

Η εντολή ulimit -a στο unix δείχνει τους πόρους που είναι διαθέσιμους σε ένα πρόγραμμα: virtual memory, stack, cpu time, processes, κτλ.

Εφαρμογή Στοίβας 2: Ισοζυγισμός Παρενθέσεων

- Ο έλεγχος σύνταξης (π.χ. ενός προγράμματος) απαιτεί να ταιριάξουμε σύμβολα/λέξεις όπως:

  `begin` με `end`

  `else` με `if`

  παρενθέσεις `{` με `}`

- Ας υποθέσουμε την ύπαρξη του συνόλου χαρακτήρων: { , } , [,] , (,).

- Πρόβλημα: να διαπιστώσετε αν μια συμβολοσειρά που περιέχει τους πιο πάνω χαρακτήρες είναι ισοζυγισμένη, δηλαδή όλες οι παρενθέσεις ταιριάζουν.

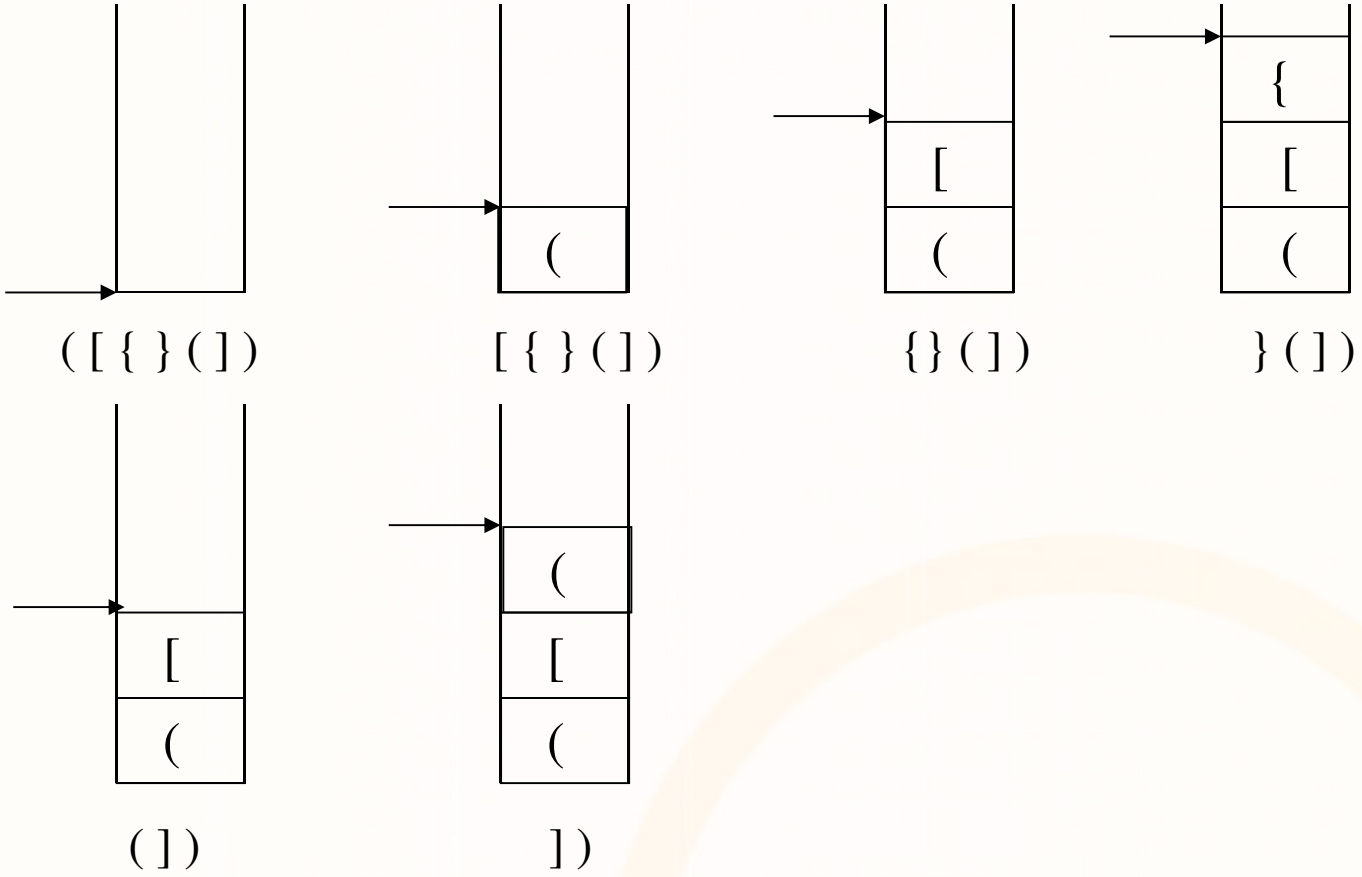
- π.χ. `{[]}`

  `([{}{}[]))`

  `([]{()})`

Εφαρμογή Στοίβας 2: Ισοζυγισμός Παρενθέσεων

- Ανά πάσα στιγμή, η στοίβα περιέχει όλες τις `αριστερές΄ παρενθέσεις που δεν έχουν ακόμη `ταιριαστεί΄.



Εφαρμογή Στοίβας 2: Ισοζυγισμός Παρενθέσεων

- Λύση (ψευδοκώδικας) βασισμένη σε στοίβες

```
MakeEmpty (S);
while (c = nextcharacter() and (c is not EOF))
    if c != (,[,{, } , ], ) continue;

    if c = (,[,{
          Push(c,S);
    else // pop item from stack
       if IsEmpty(S) report error;
        else
           d = Top(S); Pop(S);
           if c does not match d
           report error
 }
 if IsEmpty(S) report success
 else report error
```