String Parsing and Stack

String Parsing

- A string:
  - "Hi, today is another training of ACM."
- How to break into words?
  - "Hi", "today", "is", "another", "training", "of", "ACM."

Basic Implementation

- Check every character

```
char input[80] = "Hi, today is another training of ACM.";
char *p = input, *cur = input;

while(*p != '\0') {

    if(*p == ',' || *p == ' ') {

        *p = '\0';

        if(*cur != '\0') {

            printf("%s\n", cur);
        }
        cur = ++p;
    } else {

        p++;
    }
}
printf("%s", cur);
```

Use of `strtok`

- C function in <string.h> library
- Purpose: find the next token in a string
- Syntax:
  - char* strtok (char* *line*, char* *symbols*);
    - *line* - the string you want to parse, or set *line* to NULL to get the next token
    - *symbols* - list of separators (e.g. a space) which delimit each token
    - Returns a pointer to the first character of the found token

Example

- Given:

```
char *ptr, delimiters[5]=" ,",
input[80]="Hi, today is another
training of ACM.";
```

- **`ptr = strtok (input, delimiters);`**
  - **`ptr`** points to the first character of the first token "Hi"
- **`ptr = strtok (NULL, delimiters);`**
  - **`ptr`** points to the first character of the next token "today"

Example Code

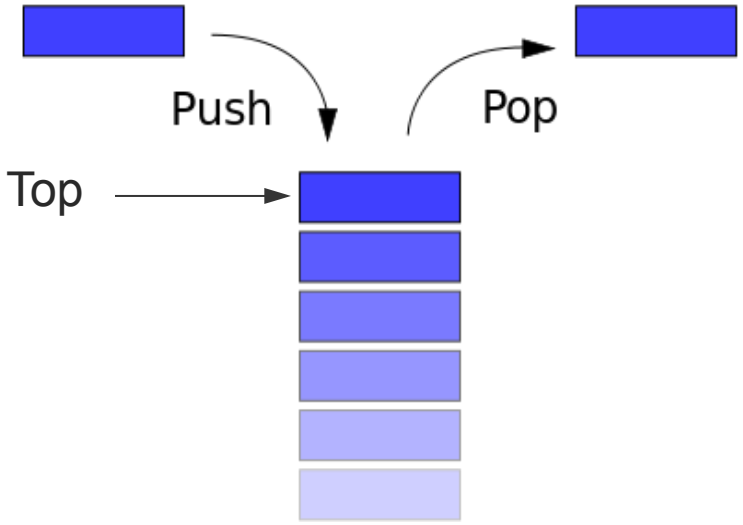
```c
#include <stdio.h>
#include <string.h>

int main(int argc, const char * argv[]) {

    char *ptr;
    char delimiters[5]=" ,";
    char input[80]="Hi, today is another training of ACM.";

    ptr = strtok(input, delimiters);

    while (ptr) {

        printf("%s\n", ptr);

        ptr = strtok(NULL, delimiters);
    }

    return 0;
}
```

Discussion

- Given a string of ‘(’ and ‘)’, check if the brackets are balanced.
  - (( ) ( )) ( ) --- Balanced
  - (( ) ( ))) (( ) --- Not balanced
  - (( )) (( ))) --- Not balanced
- How to check?
- What if more than 1 kind of bracket?
  - { [ ] [ ] ( ){ (( )) } }

Stack

- A vertical box with a single I/O
- Special feature: Last-In-First-Out



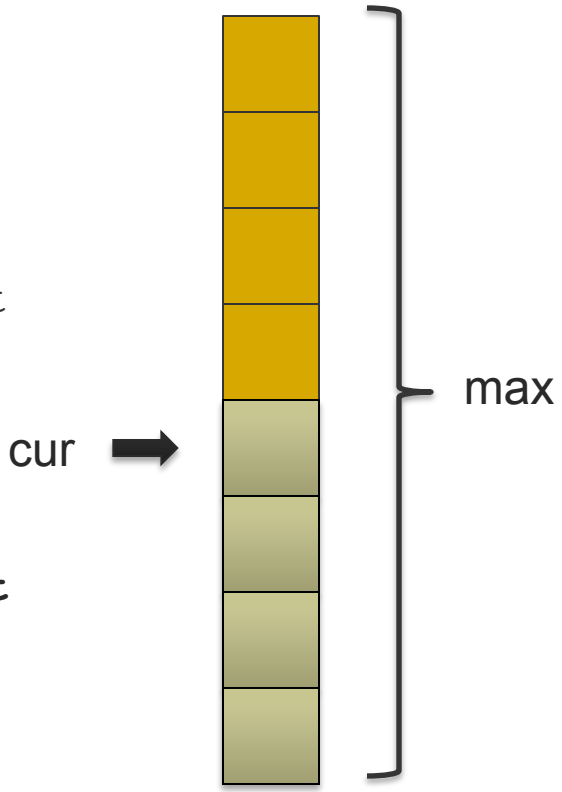
Stack Operations

- Push an item into the stack
- Pop the top item out of the stack
- What's the Top item?
- Is the stack full / empty?

Implementation

- Use an Array to realize

```
typedef struct st_stack
{
    char *data; // store the data
    int l;      // size of each bucket
    int max;    // maximum size
    int cur;    // current pointer
}stack;

int stack_init(stack *p, int size, int
   len);
int stack_pop(stack *p);
int stack_push(stack *p, char *data);
int stack_empty(stack *p);
int stack_full(stack *p);
```



What if the Stack is Full?

- Use a bigger space
  - Simple but not efficient.
- Apply a new stack and connect them
  - Complicated.
- Use linked list
  - High efficiency but needs more space to store the pointer.

Check if the brackets are balanced?

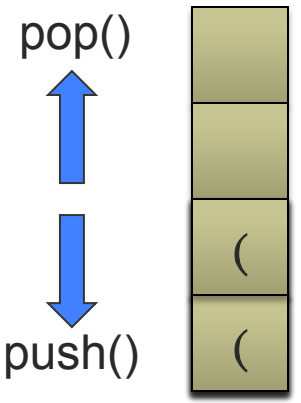
```
while(c = getchar()){
      if(c is '(')
            push(c);
      else
            if(stack_empty())
                  unbalanced;
            else
                  pop();
}
if(stack_empty())
      balanced;
else
      unbalanced;
```

Balanced

(( ) ( )) ( )

Unbalanced

(( )) (( )))



Empty! Cannot Pop()

Infix, Prefix and Postfix Notation

- Expression: (a + b) * c
  - In infix notation: (a + b) * c
  - In prefix notation: * + a b c
  - In postfix notation: a b + c *
- Other examples:

Infix	Prefix	Postfix
(a + b) / (c + d)		
(a - b / c + d) * (a + b)		

- No brackets for prefix and postfix

Postfix Evaluation

- Evaluate a given postfix expression
- Solution - stack:
  - Scan the expression from left to right, token by token
  - If current token is an operand → Push operand
  - If current token is an operator → Pop 2 operands, apply operator and push result

Postfix Evaluation Example

- (1+2)*4 = 12
- 12+4* = ?

Operator Priority

- The first thing is to define the operator priority:

(	)	+	-	*	/
1	1	2	2	3	3

Translate Infix to Postfix

- Scan from left to right, token by token
- If current token is an operand → display token
- (✸) If current token is an operator
  - case 1: stack is empty / new operator has higher priority → push the new operator
  - case 2: the priority is same or lower → pop old operator and display, and check current token again (✸)
- If current token is '(' → Push
- If current token is ')' → Pop until '(' and display all operators
- When expression finishes, pop and display remaining operators

Example: Infix to Postfix

- (a+b)*c → ab+c*
- Left to right

Translate Infix to Prefix

- Scan from **right** to **left**, token by token
- If current token is an operand → output token (use a buffer to store the output)
- (✸) If current token is an operator
  - case 1: stack is empty / new operator has **same or higher** priority → push the new operator
  - case 2: the priority is **lower** → pop and output old operator and check current token again (✸)
- If current token is ')' → Push
- If current token is '(' → Pop until ')' and output all operators
- When expression finishes, pop and output remaining operators
- Display the buffer from right to left. (Reverse the output buffer)

Example: Infix to Prefix

- (a+b)*c → *+abc
- Right to left