DYNAMIC STACK STRUCTURE

Implementing a stack with an array allocated on the heap

Delroy A. Brinkerhoff

STACK IMPLEMENTATIONS

AUTOMATIC ARRAY

- Easy to use
- The array size is fixed at creation
  - Memory is wasted if the array is much larger than needed
  - The program may fail if the array is too small

DYNAMIC ARRAY

- Flexible
  - The user specifies the array size at creation
- Error-prone
  - Easy to forget to delete the array, causing a memory leak

DYNAMIC STACK HEADER FILE

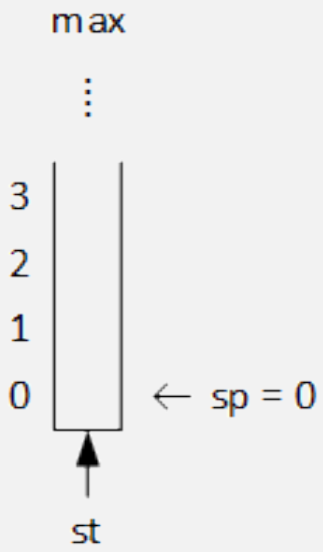
Stack Structure

Stack array allocated on the heap with new

```
struct dstack
{
        char*  st;
        int    sp;
        int    max;
};

dstack make_stack(int size);
void   init_stack(dstack* s, int size);
void   push(dstack* s, char data);
char   pop(dstack* s);
int    size(dstack* s);
char   peek(dstack* s);
void   cleanup(dstack* s);
```

INSTANTIATING AND INITIALIZING




```
dstack s;
init_stack(&s, 10);
```

```
void init_stack(dstack* s, int size)
{
    s->st = new char[size];
    s->sp = 0;
    s->max = size;
}
```

REQUIRED OPERATIONS

```
void push(dstack* s, char data)
{
    if (s->sp < s->max)
        s->st[s->sp++] = data;
    else
        throw "Stack Overflow";
}
```

```
char pop(dstack* s)
{
    if (s->sp > 0)
        return s->st[--(s->sp)];
    else
        throw "Stack Underflow";
}
```

OPTIONAL OPERATIONS

```
int size(stack* s)
{
    return s->sp;
}
```

```
char peek(stack* s)
{
    return s->st[s->sp - 1];
}
```

DESTROYING / DEALLOCATING MEMORY

- Memory allocated on the heap with new remains allocated until deleted
- Memory cannot be reused until it's deallocated
- If a program "loses" the address of memory, it can't use or deallocate it
- Unused heap memory is called "garbage"

```
void cleanup(dstack* s)
{
    delete[] s->st;
}
```

MAKING AND USING A DYNAMIC STACK

A simple client

```
int main()
{
    dstack s;
    init_stack(&s, 10);

    push(&s, 'x');
    push(&s, 'y');
    push(&s, 'z');

    char c = peek(&s);
    cout << c << endl;

    c = pop(&s);
    cout << c << endl;

    while (size(&s) > 0)
        cout << pop(&s) << endl;

    cleanup(&s);

    return 0;
}
```