

Component-Based Development

DIKU — Software Development

`<oleks@oleks.info>`

March 29, 2016

Takeaway 1/2

Do one thing well

Takeaway 2/2

Use a universal interface

DEMO

Compile-time parametrization.

stack.h: The Interface

```
1  #ifndef STACK_H
2  #define STACK_H
3
4  #define STACK_OVERFLOW (-1)
5  #define STACK_UNDERFLOW (-2)
6
7  // @return 0 on success; STACK_OVERFLOW if stack is full.
8  int
9  stack_push(int value);
10
11 // @param value Assumed to point to memory we can write to.
12 // @return 0 on success; STACK_UNDERFLOW if stack is empty.
13 int
14 stack_pop(int *value);
15
16 #endif // STACK_H
```

`stack.c`: A Statically-Allocated Stack

A statically-allocated variable is allocated at program start-up and sticks around until the program terminates.

stack.c: A Statically-Allocated Stack

A statically-allocated variable is allocated at program start-up and sticks around until the program terminates.

```
3  #include <stddef.h> // size_t
4
5  struct stack {
6      int values[STACK_SIZE];
7      size_t count;
8  } STACK;
```

stack.c: Push

```
10  int
11  stack_push(int value) {
12      if (STACK.count == STACK_SIZE) {
13          return STACK_OVERFLOW;
14      }
15
16      STACK.values[STACK.count] = value;
17      STACK.count += 1;
18      return 0;
19  }
```


stack.c: Pop

```
21  int
22  stack_pop(int *value) {
23      if (STACK.count == 0) {
24          return STACK_UNDERFLOW;
25      }
26
27      STACK.count -= 1;
28      *value = STACK.values[STACK.count];
29      return 0;
30  }
```

Makefile: A Bit Makefile Magic...

```
1 CC=gcc
2 CFLAGS=-Werror -Wextra -Wall -pedantic -std=c11
3
4 all: stack10.o stack100.o stack1000.o # ...
5
6 stack%.o: stack.h stack.c
7     $(CC) $(CFLAGS) \
8         -DSTACK_SIZE=$* \
9         -o stack$*.o \
10        -c stack.c
11
12 clean:
13     rm -f *.o
14
15 .PHONY: all clean
```

Distribution

- ▶ Distribute pre-compiled “object code” i.e., `stack.h` and `stack10.o`, `stack100.o`, *or* `stack1000.o`, etc.
 - ▶ Ask for higher price for bigger stacks — profit.
 - ▶ Customize by supplying stacks for custom types.
- ▶ Distribute source code with documentation and build instructions i.e., `Makefile`, `stack.h`, and `stack.c`.
 - ▶ Feel altruistic/philanthropic, gather street-cred.
 - ▶ Users deal in details of your implementation.

SOMEONE STARRED

MY PUBLIC GITHUB REPO

Limitations

Each object file implements a (1) **fixed-size** and (2) **fixed-type** data structure.

DEMO

Shared Libraries

DEMO

F# Language Binding

DEMO

User-allocated stack

A User-Allocated Stack

A user-allocated variable is allocated by the user of a library; the library relies on pointers to well-allocated space.

A User-Allocated Stack

A user-allocated variable is allocated by the user of a library;
the library relies on pointers to well-allocated space.

```
4  #include <stddef.h> // size_t
5
6  struct stack {
7      int *values;
8      size_t count;
9      size_t size;
10 };
```

Initialization

User should allocate space for, but not initialize the stack.

```
3 void
4 stack_init(struct stack *stack, int *values, size_t size) {
5     stack->values = values;
6     stack->count = 0;
7     stack->size = size;
8 }
```

Push

```
10  int
11  stack_push(struct stack *stack, int value) {
12      if (stack->count == stack->size) {
13          return STACK_OVERFLOW;
14      }
15
16      stack->values[stack->count] = value;
17      stack->count += 1;
18      return 0;
19  }
```

Pop

```
21  int
22  stack_pop(struct stack *stack, int *value) {
23      if (stack->count == 0) {
24          return STACK_UNDERFLOW;
25      }
26
27      stack->count -= 1;
28      *value = stack->values[stack->count];
29      return 0;
30  }
```

A Bit Less Makefile Magic...

```
1 CC=gcc
2 CFLAGS=-Werror -Wextra -Wall -pedantic -std=c11
3
4 all: stack.o
5
6 stack.o: stack.h stack.c
7     $(CC) $(CFLAGS) \
8         -o stack.o \
9         -c stack.c
10
11 clean:
12     rm -f *.o
13
14 .PHONY: all clean
```

Distribution

- ▶ Distribute object code i.e., `stack.h` and `stack.o`.
 - ▶ Lower price, better product — profit.
- ▶ Distribute source code with documentation and build instructions i.e., `Makefile`, `stack.h`, and `stack.c`.
 - ▶ Feel altruistic/philanthropic, gather street-cred.

DEMO

C++, object-oriented stack.

DEMO

C++, object-oriented, template stack.

Reading Material

- ▶ Charles W. Krueger. *Software reuse*. ACM Comput. Surv. 24(2), pp. 131–183. ACM, 1992.
- ▶ John Hughes. *Why functional programming matters*. The computer journal 32(2), pp. 98–107. Oxford University Press, 1989.

Light reading:

- ▶ oleks & br0ns. *Unix-Like Data Processing Utilities*. 2015.
<http://atu15.onlineta.org/unix-like-data-processing.pdf>

Video Material

- ▶ *UNIX: Making Computers Easier To Use* — AT&T Archives film from 1982, Bell Laboratories. <https://youtu.be/XvDZLjaCJuw>.
- ▶ Erlang Factroy SF 2016 — Keynote — John Hughes — *Why Functional Programming Matters*. <https://youtu.be/Z35Tt87pIpg>.

Summer Reading

C/C++:

- ▶ Andrew Koenig and Barbara E. Moo. *Accelerated C++: Practical Programming by Example*. Addison-Wesley, 2000.
- ▶ Jens Gustedt. *Modern C*. Unpublished, 2015. Latest version:
<http://icube-icps.unistra.fr/index.php/File:ModernC.pdf>.

C++ Templates:

- ▶ David Vandevoorde and Nicolai M. Josuttis. *C++ Templates*. Addison-Wesley, 2002.
- ▶ Alexander Stepanov and Paul McJones. *Elements of Programming*, Addison-Wesley, 2009.