CS 136
.RKT IN C

PROFESSOR M. PETRICK • WINTER 2014 • UNIVERSITY OF WATERLOO

Last Revision: January 20, 2015

Table of Contents

Future Modifications

1 Introduction

Lab: MC 3027, 10am-6

Add intro

2 Modularization

Definition 2.1. A **module** is a collection of functions that share a common aspect or purpose. **Modularization** is dividing programs into modules.

- Reusability
- Maintainability
- Abstraction

Definition 2.2. **provide** is used in a module to specify the identifiers availabie in the module.

fun.rkt

```
1 (provide fun?) ;Allows use of function outside of program
2 (define lofn '(-3 7 42 136 1337 4010 8675309))
3 ;; (fun? n) determines if n is a fun integer
4 ;; fun?: Int -> Bool
5 (define (fun? n)
6   (not (false? (member n lofn))))
```

Definition 2.3. **require** is used to identify a module that the current program depends on.

implementation.rkt

```
1 (require "fun.rkt")
2 ;;Able to use provided functions in required file
3 (fun? 7) ; => #t
4 (fun? -7) ; => #f
```

2.1 Scope

- **Local:** Visible only in local region
- **Module:** Only visible in the module it is defined in
- **Program:** Visible outside the module.

Quote. **require** also outputs the final value of any of the top-level expressions in the module. Only definitions should be included in modules.

Definition 2.4. A module **interface** is the list of functions that a module provides. Documentation should be provided.

- Description of module
- List of functions provided
- Contract and purpose for each provided function

Definition 2.5. The **implementation** is the code for the module.

- Hides implementation details from client
- Security
- Flexibility to modify implementation

Definition 2.6. High cohesion means that all interface functions are related.

Definition 2.7. Low coupling means that there is little interaction between modules.

Quote. Always truncate decimals

```
1 int main (void) {
2   printf(``Hello World! \n'')
3 }
```

Definition 2.8. %d is used as a placeholder to the values that follow.

```
1 printf("%d plus %d is: %d\n", 1 + 1, 2, 2 + 2);
```

In racket, a is used as a placeholder.

```
1 (printf ``There are ~a lights!\n'' ``four'')
2 (printf ``There are ~a lights!\n'' 'four) ; Both lines are same
```

Thurs before JAn 20

Definition 2.9. Structures in C are very similar to racket.

```
1  struct posn {
2    int x;
3    int y;
4  }; //Do not forget the semicolon
5
6  const struct posn p = {3,4}; // Initialization
7  const struct posn pp = {y=4,x=3}; // This works too
8  const struct posn pp = {x=3}; // Uninitialized integers are set to 0.
9
10 const int a = p.x;
11 const int b = p.y;
```

Definition 2.10. **begin** produces the value of the last exprssion

```
1 (define (mystery)
2   (begin ; implicit, this line not needed
3     (+ 1 2) ; evaluated, not used
4     (+ 2 2))) ;outputs 4
```

Quote. Anything that is not #f in Racket is true.

Definition 2.11. The **functional programming paradigm** is to only use constant values that never change. Functions produce new values rather than changing existing ones. In functional programming, there are no side effects.

Definition 2.12. A **side effect** does more than produce a value it also changes the state of the program. Sometimes used to debug.

Definition 2.13. **printf** in C returns an int representing the number of characters printed.