# Advanced C Programming

## Basic C Setup

Sebastian Hack
hack@cs.uni-sb.de

Christoph Weidenbach
weidenbach@mpi-inf.mpg.de

Winter Term 2008/09



# Basic Module Set Up

- typical functionality of a data structure
- typically includes one (or more) structures
- always contains an init and a free function (even if not needed)
- encapsulates all access to the structure via functions (object centration)
- for the struct there exist create functions generating an object, free functions freeing an object and delete functions recursively freeing objects

# Modules: <module>.[ch]

## What goes into <module>.h

- includes of other module interfaces
- type and struct definitions all followed by documentation blocks
- function prototypes
- macros FORBIDDEN

## What goes into <module>.c

- include of module.h
- definition of functions
- definition of an init and a free function
- all functions not shown as prototypes in <module>.h are declared static

# Layout of <module>.h

```
<header>
#ifndef _<module uppercase name>_
#define _<module uppercase name>_
/*************/
/* Includes    */
/*************/
< all Includes >
/***************/
/* Structures    */
/***************/
< all Structures and Type Definitions >
/***************/
/* Functions    */
/***************/
< all Function prototypes >
#endif
```

# <header>

```
/*****************************************/
/*****************************************/
/**     <module uppercase name>    **/
/**                                **/
/**   <short module description>   **/
/**                                **/
/** <disclaimer, copyright statement> **/
/**                                **/
/**   Author: <author>           **/
/**                                **/
/**   Contact: <contact details> **/
/*****************************************/
/*****************************************/
```

# Layout of <module>.c

```
<header>
#include "<module>.h"
/**************/
/* Functions  */
/**************/
<all Function definitions>
```

# Names

- <module name> = typically 3-7 lowercase characters word without special characters, e.g., list
- <struct pointer name> = <type name> = typically 3-7 uppercase characters word without special characters, e.g., LIST
- <struct name> = <struct pointer name>_NODE, e.g., LIST_NODE
- <struct tag> = <struct pointer name>_HELP, e.g., LIST_HELP
- <function name> = <module name>_<function description>, e.g., list_DeleteDuplicates
- <function description> = typically 3-30 character long concatenation of words without special characters, e.g., DeleteDuplicates

# Struct Documentation

- follows directly each struct definition
- explains each field

```
/**************************************/
/**************************************/
/**     <field 1 description>      **/
/**            <...>               **/
/**     <field n description>      **/
/**************************************/
/**************************************/
```

# Function Interface Documentation

## <Function Definition>

```
<return type> <function name>(<argument declarations>)
<function interface documentation block>
{ <body> }
```

## <function interface documentation block>

```
/*************************************************************
 INPUT:  <argument descriptions>
 RETURNS: <description of the output>
 CAUTION: <anything special>
 MEMORY: <dynamic storage allocation effects>
 SUMMARY: <description of the function>
*************************************************************/
```

# Inside Function Documentation

- explain the non-obvious
- explain invariants
- less is sometimes better
- no duplicate information
- use identation

# Coding Style

- No macros (later we may relax this a little)
- Names are always meaningful (see before)
- No expressions on assignemnts left hand side: $a[j++] = 5$
- No gotos
- Structure arguments are always pointers
- No $->$ operator usage outside the <module>.c file where the structure is defined in <module>.h

# Libraries

## Permitted Library Usage (will be extended later on)

- stdio: input /output to terminal and file system
- string: string and character operations
- stdlib: memory access
- limits: min and max bounds for standard C data types

# Malloc and Free

## Prototypes

```
void *malloc(size_t size);
void free(void *ptr);
```

## Usage

- for structs malloc usage only in create functions of a struct called malloc(sizeof( <struct name>))
- for structs free usage only in free/delete functions

# Make

## 4 Basic Tasks

- make: compiles the application
- make clean: remove everything generated by make
- make depend: establish source file dependencies
- make archive: create a .tgz archive containing all necessary sources

## No Warnings for all Options

- always use options "-ansi -pedantic -Wall -Wshadow -Wpointer-arith -Wwrite-strings"
- use option "-O2" for optimized code generation
- use option "-g" for debug code generation