Introduction to the C programming Language

Praktikum Kernel Programming
University of Hamburg
Scientific Computing
Winter semester 2015/2016

Konstantinos Chasapis
Konstantinos.chasapis@informatik.uni-hamburg.de

Outline

- Hello World
- Preprocessor/Compiler/Linker
- The basis
- Pointers
- Memory Layout/allocation
- Structures/unions
- Inline assembly

Hello World

- Preprocessor
  - cpp
  - #include <stdio.h>
- main function
  - Gets two arguments
    - argc, the #of command line arguments
    - argv, strings of the command line arguments
  - Returns an integer to denote the execution status
    - 0 in successfull execution
- Printf
  - Prints to the standard output

```
#include <stdio.h>

int
main (int argc, char*arv[]) {
  printf( "Hello world !\n");
  return 0;
}
```

Preprocessor (cpp)

- Executes before the compilation
- Include header files
  - #include <filename.h>
  - < > - search in standard directories
  - “ ” path to the file
- Macros expansion
  - #define EXT VALUE
  - #undef EXT
- Conditional Compilation
  - #if, #elif, #endif
  - #ifdef, #ifndef

Preprocessor (cpp)

- Other directives
  - # Replaces a macro parameter with a string constant
  - ## Token merge, creates a single token from two adjacent ones
- Macros continuation (\)
  - #define LONG_MACRO \

    Expand_also_here

Preprocessor (cpp)

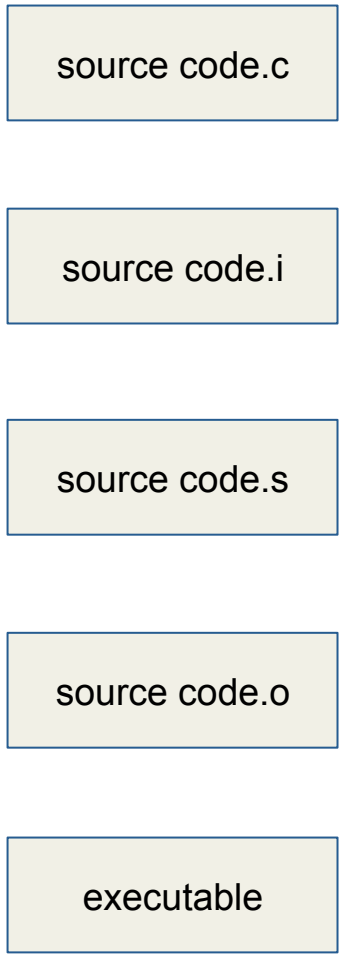
- Useful predefined macros
  - __DATE__
    - Current date as a string with "MMM DD YYYY" format
  - __TIME__
    - Current time as string with "HH:MM:SS" format
  - __FILE__
    - Current file name as a string
  - __LINE__
    - Current line number as a number
- https://gcc.gnu.org/onlinedocs/cpp/

Compile

- Many open source compilers
- gcc (GNU Compiler Collection)
  - -c, creates an object file
    - $filename.o
  - -o, name the executable
    - default name a.out
  - -g, add debugging information
  - -Wall, enables all warnings
- gcc -o hello_world hello_world.c
- http://pages.cs.wisc.edu/~beechung/ref/gcc-intro.html

Create the executable

- C preprocessor
  - Pass preprocessor directives
- Compilation
  - C source code into assembly code
  - Check for syntax errors
- Assembling
  - Assembly source code into assembly listing with offsets
- Linking
  - Takes one or many object files and libraries to create a single executable



Outline

- Hello World
- Preprocessor/Compiler/Linker
- The basis
- Pointers
- Memory Layout/allocation
- Structures/unions
- Inline assembly

Identifiers names

- Use to identify
  - Variable names
  - Function names
  - Other user-defined items
- Format
  - Starts with A-Z, a-z
  - Followed by letters, digits, underscores _
  - Case sensitive

Variables basic data types

- int, integers
  - signed
  - unsigned
  - short
  - long long
  - char, characters
    - signed char, -128 .. 127
    - unsigned char, 0 .. 256
- floating point (IEE754 standard)
  - float, singled precision
  - double, double precision

void type

- Multiple meanings
- When used in function argument list
  - funct(void)
  - Does not accept any argument
- Return value of a function
  - void fucnt(..)
  - Does not return any value
- Pointer to a value
  - void *v_pointer;
  - points to unspeciated data type

Variables qualifiers

- Const
  - Variable can not change after initialization
  - const float pi=3.14;
- Volatile
  - Extreme opposite of const.
  - Indicates that a variable may be changed in a way
  - Every reference to the variable will reload the contents from memory
  - volatile int v;

enum data type

- Abbreviation for enumerate
- Declare and initialize a sequence of integer constants
- Example:
  - enum colors {RED, YELLOW, GREEN};
  - enum colors {RED=0, YELLOW=1, GREEN=2};

Variable Storage Class

- auto
  - Default for storing the local variables
- register
  - Local variables that should be stored in registers instead of RAM
  - fast access
- static, two meanings
  - Global variables, can be seen in this file only.
  - Local variables, initialized once, retains value during various calls
- extern
  - Make a global variable visible to all program files

Variables size - sizeof()

- Special operator returns the size in char
- Examples
  - sizeof(char) // 1
  - sizeof(int) // 4
  - sizeof(float) // 4
  - sizeof(double) // 8

Arrays

- Made from elements of basic types
- Use square brackets [] to declare them
- Can have multiple dimensions
- Can not change size during execution time
- Allocated sequentially
- Start from 0
- Examples
  - int a[3] = {1, 2, 3};
  - char strings[2][5] = {“this”, “is”, “test” };

Strings

- Special case of char arrays
- Null terminated
- Example:
  - char string[5];
    - string[0] = `t`;
    - string[1] = `e`;
    - string[2] = `s`;
    - string[3] = `t`;
    - String[4] = `\0`;

Variable scope

- Global
  - Defined outside of a block/function
  - Can be accessed by any function/block
  - Retain value during the program execution
- Local
  - Defined inside a block (compound statement)
  - Mainly in the top of the function/block
  - Used only in this block
  - Overwrites global variables with the same name
  - Lose value at the end of the function/block
- Formal
  - Function parameters, used as local variables

Constants

- Numeric values
  - Octal starts with 0, 015
  - Hexadecimal starts with 0x, 0xA1
  - Long ends with L, 12312341124L
- Characters
  - \n, newline
  - \t, tab
  - \\, backslash
  - \’’, single quote
  - \0, string terminate character
  - NULL, integer value of 0

Operators

- Arithmetic Operators
  - +, -, *, /, %, ++, --
- Logical Operators
  - ==, !=, <=, <, >=, >, !, &&, ||
- Bitwise Operators
  - &, |, ^, ~, <<, >>
- Assignment Operators
  - =, +=, -=, *=, /=, %=, &=, ^=, |=
- Misc Operators
  - sizeof(), &, *, ?=

Statements

- Expression statements
  - Expression followed by semicolon ;
    - a = 1 + 3;
- Compound statements
  - One or more statements enclosed by a pair of braces { }
    - { pi=3.14; area = pi * radios * radios; }
- Control statements
  - Control the execution flow
    - if, else, case, for etc.

Conditional statements Contraction

- if (expression) statement;
  - if expression evaluated as non zero enters the if statement
- Example

```
if ( a < X ) { code_block }
else if ( a > X ) { code_block }
else { code_block }
```

Conditional statements Contraction

```
switch ( expression ) {
    case constant-expr1: statement1;
    [case constant-expr2: statement2;]
    default: statement3;
}
```

- First evaluates expression
- Then enters the case that matches the expression values
- Otherwise executes the default
- Continues the execution until it finds break;

Conditional statements Contraction

- Check if variable “a” is equal to X or Y or Z

```
switch ( a ) {
    case X: { code block }
    case Y: { code block }
    case Z: { code block }
    default:
        // not eqaul to X,Y,Z
}
```

Looping - for

- for (expr1; expr2; expr3) {compound statements}
  - Executes the expr1 in the beginning
  - Evaluates the expr2 and if non zero executes the compound statements
  - At the end executes the expr3 and reevaluates the expr2 continues until expr2 is zero.
- Example

```
for (i=0; i < ARRAYSIZE; i++) {
    code block;
}
```

Looping - while

- while (expression) {compound statements}
  - Evaluates expression and if non zero executes the compound statements until expression is non zero
- Example

```
i = 0;
while ( i < ARRAYSIZE ) {
    code block;
    i++;
}
```

Looping – do while

- do { compound statements }

  while (expression);
  - Executes the compound statements
  - Evaluates the expression and if non zero repeats
- Example

```
do {
    i++;
} while ( i < ARRAY_SIZE );
```

break and continue

- break
  - Exits a loop or a case statement
- continue
  - Continues the execution from the beginning of the loop

```
while ( i < 10 ) {
      i = i + 1;
      if ( i < 10  ) continue;
      i = 0;
      break;
}
```

```
switch ( a ) {
      case Z: { code block
            break;
      } case Y: { code block
      } case X: { code block }
}
```

Commnets

- Single line comments
  - // comments
- Multiple lines comments
  - /* comments */

Functions

- Made up by four components
  - Return value
  - Function name
  - Arguments list
  - Function body
- Example
  - int function_name (int a, int b) { return a+b; }
- Parameters
  - Call by value, copies the value of the argument to the formal parameter
  - Call by reference, copies the address of the argument to formal parameter

Functions examples

- Returns an integer gets no argument
  - int foo(void);
- Returns the sum of two floats
  - float float_sum(float a, float b) { return a + b; }

Outline

- Hello World
- Preprocessor/Compiler/Linker
- The basis
- Pointers
- Memory Layout/allocation
- Structures/unions
- Inline assembly

Pointers

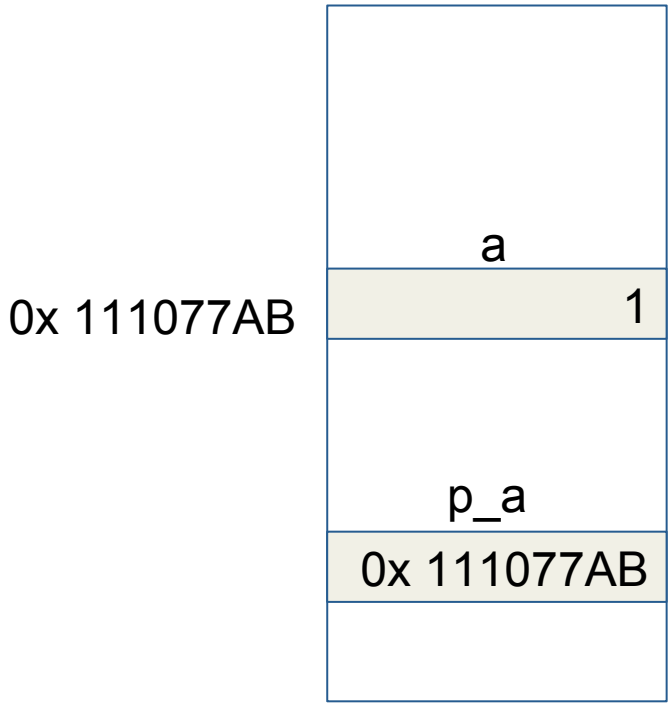
- Variable store addresses
  - variable addresses
  - functions
- Declaration
  - type *name; // variable
  - type (*f_name)(..); // function pointer
- Examples
  - int *p_i; // pointer to an int
  - double *p_d; // pointer to double
  - void (*p_f) (int); // function pointer to a function that returns void and has an integer parameter

Pointers cont.

- Dereference
  - Get the value of the variable pointer points to
  - Use (*)
- Address of a variable
  - Use (&)
- Example

  int a = 1;

  int *p_a = &a;



Pointers type casting

- Convert a variable from one type to an other
- Used to cast void *
- Examples

```
void *v; int *a;
v = (void *)a;
a = (int *)v;
```

Pointers example

- Function that will swap the value of two integer variables

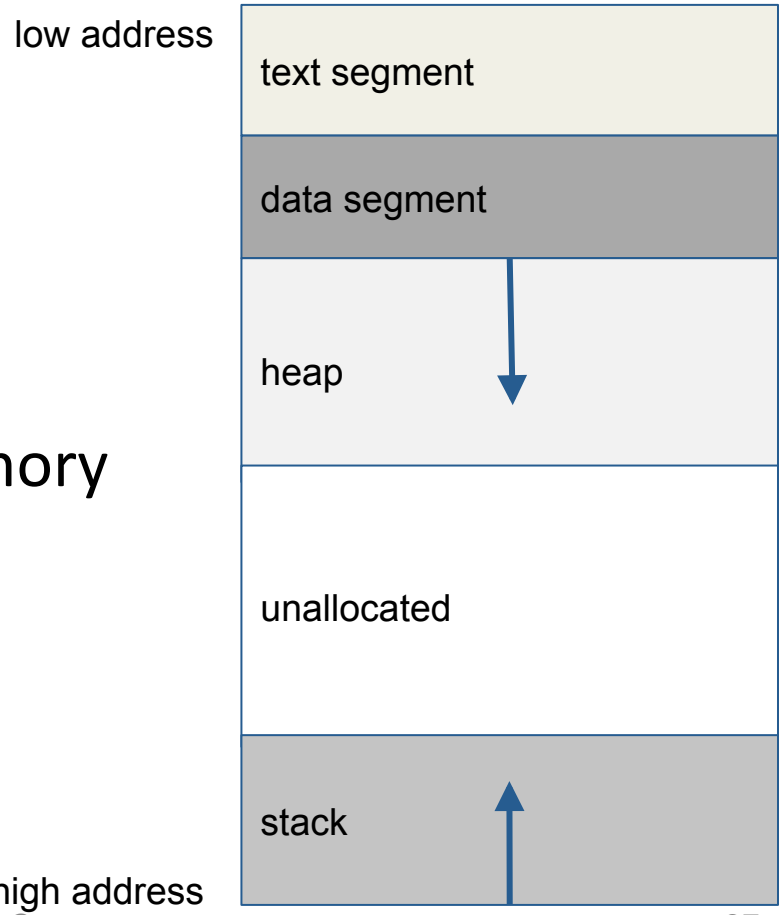
```
void swap(int *a, int *b) {
    int tmp;
    tmp = *a; // store the value that a points to at tmp
    *a = *b;
    *b = tmp;
}
```

- How to call the function

```
int a=3, b=4;
swap(&a, &b);
```

Memory Layout

- text segment
  - Source code
- data segment
  - Global variables
  - Static variables
- heap
  - Dynamically allocated memory
- stack
  - Local variables
  - Function parameters



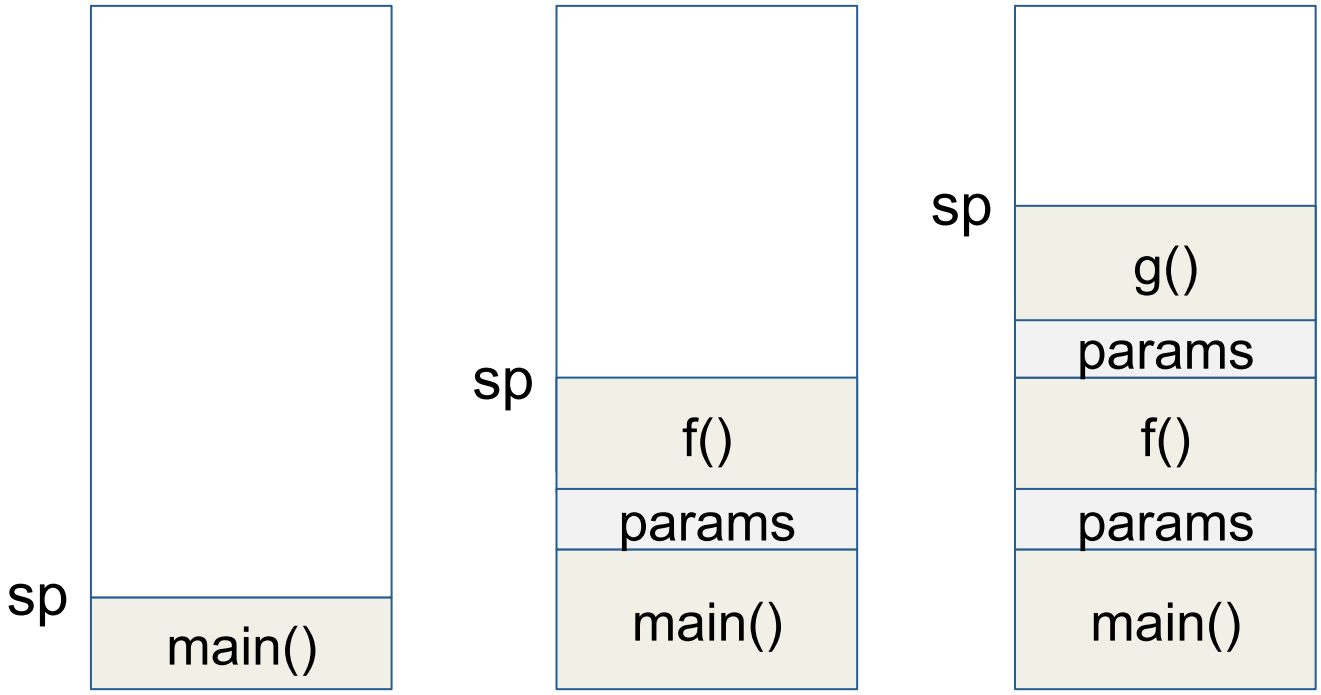
Stack allocation

- stack pointer (sp)
  - top of the stack

```
g() { ; }

f() {
      g();
}

main(){
      f();
}
```



Heap allocation

- Stack allocates static size variables
  - Size knowed at compile time
- Dynamically memory stored in heap
- Allocate using
  - malloc() // allocates requested amount
  - calloc() // allocates and zeros memory
- You have to explicitly free the memory
  - Does not have garbage collection
  - free()

Heap allocation Examples

- Allocate an array of 10 integers

```
int *a;
a = (int *) malloc (10 * sizeof(int));
free(a);
```

- Allocate an array of 10 floats and zero them

```
float *a;
a = (float *) calloc (10, sizeof(float));
free(a);
```

Outline

- Hello World
- Preprocessor/Compiler/Linker
- The basis
- Pointers
- Memory Layout/allocation
- Structures/unions
- Inline assembly

Structures

- Collection of items of different types
  - Items called members/fields
- Similar to Classes of Java
- Example

```
struct person {
        char name [255];
        int height;
}
```

- Access struct members
  - For reference/pointer to a struct use ->
    - struct person *p_person; p_person->height;
  - For the actual struct use .
    - struct person person; person.height;

Unions

- Store different data type in a single memory location
- Similar definitions of structs
- Example

```
union Data {
   int i;
   float f;
   char  str[20];
}
```

- Access fields using .
  - Data.i = 3;

Typedef

- Name a new type
- Mainly used for struct but not limited
- Syntax
  - typedef type type_name
- Examples
  - typedef unsigned char BYTE;
  - typedef struct person person_t;

Inline assembly

- asm - gcc extension
  - Read and write C variables from assembler
  - Perform jumps from assembler code to C labels
  - For ansi and std use _asm_
- Basic syntax
  - asm [volatile] (Assembler Instructions)
  - Add multiple assembly lines with '\n\t'
- Extended asm syntax
  - asm [volatile] (AssemblerTemplate
    : OutputOperands
    [: InputOperands
    [: Clobbers]])

asm examples

- Copy src to dst and add 1 to dst
  - asm ("mov %1, %0\n\t"
    "add $1, %0"
    : "=r" (dst)
    : "r" (src));

Keywords

auto	extern	signed	while
else	return	void	do
long	union	continue	int
switch	char	goto	struct
break	float	sizeof	double
enum	short	volatile	
register	unsigned	default	
typedef	const	if	
case	for	static	

Questions?

Konstantinos Chasapis

Konstantinos.chasapis@informatik.uni-hamburg.de