Introduction to C

PS Vertiefung Betriebssysteme

FB Computerwissenschaften
University of Salzburg

Harald Röck

Introduction to C

- The Good:
  - Small set of portable instructions
    - Macro assembler
  - Control of low-level mechanisms
  - Performance
- The Bad:
  - Manual memory management
  - Manual error detection
  - Easier to make mistakes

Java == C

- Operators same as Java:
  - Arithmetic:
    - `int i = i+1; i++; i--; i *= 2; +, -, *, /, %,`
  - Relational and Logical
    - `<, >, <=, >=, ==, !=; &&, ||, &, |, !`
- Syntax same as in Java:
  - `if ( ) { } else { }`
  - `while ( ) { }; do { } while ( );`
  - `for(i=1; i <= 100; i++) { }`
  - `switch ( ) {case 1: … }`
  - `continue; break;`

Types

- Standard Types:
  - **`char, short, int, long, float, double`**
  - No boolean and no character type: **`char`** is a byte integer
- Arrays:
  - **`int a[10]; /* array size is always static */`**
- Strings:
  - `Null` terminated `char` arrays
    - **`"Hello World"[1]`** ⟶ `'e'`
    - Spinning wheel: **`putchar("\|/-"[i & 0x3]);`**

Complex types

- Example date type:
  - ```
    struct date {
        int day,
        int month,
        int year
    };
    [...]
    struct date today; /* on stack */
    today.day = 16;
    today.month = 10;
    today.year = 2008;
    ```
    ```

Pointers

- Simple example:
  - ```
    int j;
    int *ptr;

    ptr = &j;
    *ptr = 4; /* j = 4 */
    ```
- Struct pointers:
  - ```
    struct date today;
    struct date *ptr = &today;

    ptr->day = 16; /* not ptr.day! */
    ptr->month = 10;
    ptr->year = 2008;
    ```

Array == Pointer

- Array variables are pointer variables!
  - ```
    int v[5];
    int i;
    for (i=0; i<5; ++i)
        v[i] = 0;
    ```

    ```
    int *p;
    for (p = v; p < v + 5; ++p)
      *p = 0;
    ```
- More pointers:
  - ```
    int v[5]; int *p = v; int i;
    for (i=0; i<5; ++i)
        v[i] = i;
    ```

    ```
    for (i=0; i<5; ++i)
        *(p + i) = i;
    ```
  - → `v[0] == *p`
  - → `p[0] == *v`

More Pointers

```
int month[12]; /* month is a pointer to base address
430*/

month[3] = 7;  /* month address + 3 * int elements

                => int at address (430+3*4) is now 7 */

ptr = month + 2; /* ptr points to month[2],

              => ptr is now (430+2 * int elements)= 438 */

ptr[5] = 12; /* ptr address + 5 int elements

                => int at address (434+5*4) is now 12.

                   Thus, month[7] is now 12 */

ptr++;          /* ptr <- 438 + 1 * size of int = 442 */

(ptr + 4)[2] = 12; /* accessing ptr[6] i.e., array[9] */
```

- Now , `month[6]`, `*(month+6)`, `(month+4)[2]`, `ptr[3]`, `*(ptr+3)` are all the same integer variable.

Functions

- All functions are global and must be defined before called.
- Call by reference
  - ```
    int sum(int *a, int *b) {
        return *a + *b;
    }
    ```
- Struct parameters
  - ```
    void set_day(struct date *date, int day) {
        date->day = day;
    }
    [...]
    struct date today;
    set_day(&today, 16);
    ```

Call by Reference

- More Structs:
  - ```
    void add_month(struct date *in, struct date *out) {
        out->day = in->day;
        out->month = in->month + 1;
        out->year = in->year;
    }
    [...]
    struct date a, b;
    a.day = 16; a.month = 10; a.year = 2008;
    add_month(&a, &b);
    ```
  - → b.month == 11
-
    ```

Function Pointers

- A function is an address in the program
  - ```
    int sum(int a, int b) { return a + b; }
    int min(int a, int b) { a<b?a:b; }

    int (*fp)(int a, int b);
    int x, y, z;
    x = 3;
    y = 5;

    fp = sum;
    z = fp(x, y);

    fp = min;
    z = fp(x, y);
    ```
- What's this?
  - ```
    int call(int (*fp)(int a, int b), int a, int b);
    ```

More Function Pointers

- Callback:
  - ```
    int call(int (*fp)(int a, int b), int x, int y) {
        return fp(x, y);
    }
    [...]
    int x, y, z;
    z = call(sum, x, y);
    z = call(min, x, y);
    ```
    ```

Dynamic Memory

- Explicit memory management
  - No garbage collector
  - Memory is limited
    - → always deallocate not used memory
  - ```
    int *ptr;
    ptr = malloc(sizeof(int)); /* alloc space for int */
    *ptr = 5;
    free(ptr); /* free allocated space */
    ```
    ```

Multiple Files

- Split program into multiple source files and header files
  - Header files contain only interfaces
  - Source files contain program logic
- Encapsulation:
  - by default all functions are global
  - Use `static` modifier for file private functions or variables
  - Use `extern` modifier to access globals in another file

Multiple Files

- Example:
  - main.c

```
#include "prog.h"
[...]
int a, b, c;
a = 3;
b = 2;

c = sum(a, b);
```

prog.c

```
#include "prog.h"
int sum(int a, int b) {
    return a + b;
}
```

prog.h

```
int sum(int a, int b);
```

- Compile and link:

```
# gcc -c main.c -o main.o
# gcc -c prog.c -o prog.o
# gcc prog.o main.o -o prog
```

Live Demos