Coisas que ninguém deveria ter que saber sobre C

Nuno Lopes

Aliasing

```
void foo(char *a, char *b)
{
    *a = 1;
    *b = 2;

    if (*a == *b) {
        // dead code... or not!
    }
}
```

Aliasing

```
void foo(char *a, char *b)
{
    *a = 1;
    *b = 2;

    if (*a == *b) {
        // dead code... or not!
    }
}


int a, b;
foo(&a, &b);
foo(&a, &a);
```

Keyword 'restrict' - C99

```
void foo(char * restrict a, char * restrict b)
{
    *a = 1;
    *b = 2;

    if (*a == *b) {
        // dead code for sure
    }
}
```

Keyword 'restrict' - C99

```
void foo(char * restrict a, char * restrict b)
{
    *a = 1;
    *b = 2;

    if (*a == *b) {
        // dead code for sure
    }
}


int a, b;
foo(&a, &b);
foo(&a, &a); // undefined behavior!
```

Exemplo dos efeitos de Aliasing

```
void Update(int *a, int *b, int *val)
{
    *a += *val; // load val, store a
    *b += *val; // load val, store b
}
```

*val não pode ir para um registo

memcpy() vs memmove()

Man Page:

- The **memcpy**() function copies *n* bytes from memory area *src* to memory area *dest.*
- The **memmove**() function copies *n* bytes from memory area *src* to memory area *dest* .

memcpy() vs memmove()

```
void *memcpy(void * restrict dest, const void * restrict src, size_t n);

void *memmove(void *dest, const void *src, size_t n);
```

Static na teoria

Namespacing:

```
static int x;

static void foo();
```

Alocação Estática:

```
void xpto() {
    static int x = 0;
}
```

....

Static na prática

Permitir optimizações:
static int x;

Static na prática

Permitir optimizações:

```
static int x;

void foo() {
    static int my_const_vector[] = {1,2,3,4,5,....};
}
```

Static na prática

Permitir optimizações:

```
static int x;

void foo() {
    static int my_const_vector[] = {1,2,3,4,5,....};
}


 // IPO
static void xpto() {
     ...
}
```

Static na prática

Inlining com remoção de símbolo:

```
static void foo();

void a() { foo(); }

void (*ptr)() = foo;
```

Static na prática

Mais optimizações (C99):

void foo(int a[static 10]);

- array com pelo menos 10 entradas

Init arrays (C99)

```
int widths[] = {
    [0 ... 9] = 1,
    [10 ... 99] = 2,
    [100] = 3
};
```

Visibilidade de Símbolos

Visibilidade de Símbolos

```
#ifdef MSVC
# define EXPORT __declspec(dllexport)
#else
# define EXPORT __attribute__ ((visibility("default")))
#endif

EXPORT void foo()
{
   ....
}
```

```
gcc -fvisibility=hidden x.c
```

Const na prática

```
const int array[] = {1,2,3,4,5,...}
```

Const na prática

const int array[] = {1,2,3,4,5,...}

- DATA vs RODATA
- partilha em DSOs

Struct Layout

```
typedef struct {
    char c;
    int i;
    short s;
    double d;
} type;

type x = {'a', 1, sizeof(type), 0.1};
```

Struct Layout

```
.globl x
    .data
    .align  4
    .type   x, @object
    .size    x, 20
x:
    .byte    97
    .zero    3
    .long    1
    .value  20
    .zero    2
    .long    -1717986918
    .long    1069128089
```

```
typedef struct {
    char c;
    int i;
    short s;
    double d;
} type;
```

Struct Layout (GNU C)

```
.globl x
    .data
    .align  4
    .type   x, @object
    .size    x, 15
x:
    .byte    97
    .long    1
    .value  15
    .long    -1717986918
    .long    1069128089
```

```
typedef struct
__attribute__((packed))
{
    char c;
    int i;
    short s;
    double d;
} type;
```

++x vs x++

```
type operator++(type x) {
    type tmp = x;
    x = x + 1;
    return tmp;
}

type ++operator(type x) {
    x = x + 1;
    return x;
}
```

Pure Functions (GNU C)

```
int foo(int x) __attribute__((pure));

int bar()
{
    return foo(3) + foo(3);
}
```

Pure Functions (GNU C)

```
.globl bar
        .type bar, @function
bar:
        pushl %ebp
        movl %esp, %ebp
        subl $8, %esp
        movl $3, (%esp)
        call foo
        addl %eax, %eax
        leave
        ret
```

Branch Prediction (GNU C)

```
#define likely(x)      __builtin_expect(!!(x), 1)
#define unlikely(x)  __builtin_expect(!!(x), 0)

void foo(int x)
{
    if (likely(x > 1))
        return 3;

    ....
}
```

Prefetch (GNU C)

```
#define prefetch(x) __builtin_prefetch(x)

static inline void prefetch_range(void *addr, size_t len)
{
    char *cp;
    char *end = addr + len;

    for (cp = addr; cp < end; cp += PREFETCH_STRIDE)
        prefetch(cp);
}
```