Computer Lab: C for Lab 5

2º L.EIC

Pedro F. Souto (pfs@fe.up.pt)

April 4, 2024

Contents

C Pointers

- A C pointer is a data type whose value is a memory address.
  - Program variables are stored in memory
  - Other C entities are also memory addresses
- C provides two basic operators to support pointers:

  `&` to obtain the address of a variable. E.g.

```
p = &n; /* Initialize pointer p with
           the address of variable n */
```

  `*` to dereference the pointer, i.e. to read/write the memory positions it refers to.

```
*p = 8; /* Assign the value 8 to memory position
           whose address is
           the value of p (variable n) */
```

- To declare a pointer (variable), use the `*` operator:

```
int *p; /* Variable/pointer p points to integers or
           the value pointed to by p is of type int */
```

- Use of pointers in C is similar to the use of indirect addressing in assembly code, and as prone to errors.

C Pointers and Arrays

- The elements of an array are stored in consecutive memory positions
- In C, the name of an array is the address of the first element of that array:

```
int a[5];
p = a;          /* set p to point to the first element */
p = & (a[0]); /* same as above */
```

- C supports pointer arithmetic – meaningful only when used with arrays. E.g. to iterate through the elements of an array using a pointer:

```
for( i = 0, p = a; i < 5; i++, p++) {
    ...
}
```

or, without using variable `i`:

```
for( p = a; p-a < 5; p++) {
    ...
}
```

IMP: Pointer `p` must be declared to point to variables of the type of the elements of array `a`.

C Pointers and Pointer Arithmetic: `vg_fill()`

- Actually, pointer arithmetic may be used when we want to access a collection of data items of the same type that are layed consecutively in memory. E.g., the pixels in VRAM in graphics mode.

```
static void *video_mem; /* Address to which VRAM is mapped */
static unsigned hres;   /* Frame horizontal resolution */
static unsigned vres;   /* Frame vertical resolution */

void vg_fill(uint32_t color) {
  int i;
  uint32_t *ptr;   /* Assuming 4 byte per pixel */
  ptr = video_mem;

  for(i = 0; i< hres*vres; i++, ptr++) {
    *ptr = color;    /* Handle a pixel at a time */
```

- Variables `video_mem`, etc. are global, but static
- `ptr++` takes advantage of pointer arithmetic (here adds 4, because each `uint32_t` takes 4 bytes)

Contents

video_test_rectangle()

- Draw a rectangle on the screen in the desired mode

```
int video_test_rectangle(uint16_t mode, uint16_t x, uint16_t y,
                         uint16_t width, uint16_t height, uint32_t color)
```

- Need to handle different graphical modes, i.e. different:

  Resolution both horizontal and vertical

  Bits per pixel And color models

  Indexed color modes also called packed-pixel by VBE, appear to have only 8 bits per pixel

  Direct color modes with a different number of bits per pixel

  - And sometimes, the number of bits per component may be different, even if the number of bits per pixel is the same.

- These affect the offset of 1) the memory location of a pixel, wrt the frame-buffer base address, or of 2) the RGB components.
- The goal is that your code be parameterizable so that it can easily handle these differences
- To facilitate testing, we suggest you use (see handout):

  vg_draw_hline(uint16_t x, uint16_t y, uint16_t len, uint32_t color)

  vg_draw_rectangle(uint16_t x, uint16_t y, uint16_t width, uint16_t height, uint32_t color)

Changing Pixel Values in Video RAM (1/2)

- Modes that use 4 bytes per pixel, are easy to handle
  - Check the code for `vg_fill()`, two slides before
- So are modes that use 2 or 1 byte per pixel
  - Can use `uint16_t` or `uint8_t`, respectively

Challenge what about modes that use 3 bytes?

- Compilers available in Minix do not have `uint24_t`

Solutions

Use `memcpy()` "copy memory area"

```
#include <string.h>

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

Use a struct

```
typedef struct{
   uint8_t comp[3];
} rgb_8_8_8_t;
```

Padding and Alignment

Issue C compilers layout data types in memory with the goal of making accesses faster

- Most ISA require data types to be aligned for faster access.

  `uint16_t` values start on even addresses;
  `uint32_t` values on addresses that are divisible by 4

But, sometimes, memory is at a premium, and you want to pack data as tight as possible. E.g.

- In VBE most data structures are packed to save memory

Thus, simply defining a C struct with one member per parameter with an appropriate type may not be enough.

Solution Use `#pragma pack`

- Must also reset to the default by adding after the structure:

  ```
  #pragma options align=reset
  ```

- Alternatively you can also use GCC's `__attribute__((packed))` , which is also supported by clang

Packing in `vbe_mode_info_t`

```
#pragma pack(1)
typedef struct {
    uint16_t ModeAttributes;
    [...]
    uint16_t XResolution;
    uint16_t YResolution;
    [...]
    uint8_t BitsPerPixel;
    [...]
    uint8_t RedMaskSize;
    uint8_t RedFieldPosition;
    [...]
    uint8_t RsvdMaskSize;
    uint8_t RsvdFieldPosition;
    [...]
    uint32_t  PhysBasePtr;
    [...]
}  vbe_mode_info_t;
#pragma options align = reset
```

For more info: The Lost Art of Structure Packing, Eric S. Raymond

Packing in `video_test_pattern()`

color(0,0)	color(0,1)	color(0,2)	color(0,3)
color(1,0)	color(1,1)	color(1,2)	color(1,3)
color(2,0)	color(2,1)	color(2,2)	color(2,3)
color(3,0)	color(3,1)	color(3,2)	color(3,3)

Purpose to learn how to change the color components in the different modes

Indexed modes instead of color use the color pallette index:

index(row,col) = (first + (row * no_rectangles + col) * step) % (1 << BitsPerPixel)

In mode 0x105, when you pick the arguments remember that all colors in the pallette with an index larger than 63 are black

Direct modes in this case you have to set each of the color components using:

R(row, col) = (R(first) + col * step) % (1 << RedScreeMask)
G(row, col) = (G(first) + row * step) % (1 << GreenScreeMask)
B(row, col) = (B(first) + (col + row) * step) % (1 << BlueScreeMask)

Hint Call function that you should implement for `video_test_rectangle()`:

```
int vg_draw_rectangle(uint16_t x, uint16_t y, uint16_t width,
                      uint16_t height, uint32_t color)
```

Contents

(Anonymous) Unions with Anonymous Structs

```
typedef struct reg86 {
  union {
    struct { /* 32-bit (double word) access*/
        [...]
    };
    struct { /* 16-bit (word) access */
        [...]
    };
    struct { /* 8-bit (byte) access */
      u8_t intno;  /* Interrupt number (input only) */
      u8_t : 8;    /* unused */
      u16_t : 16;  /* unused */
      [...]        /* unused */
      u8_t al, ah; /* 8-bit general registers */
      u16_t : 16;  /* unused */
      u8_t bl, bh; /* 8-bit general registers */
      u16_t : 16;  /* unused */
      u8_t cl, ch; /* 8-bit general registers */
      u16_t : 16;  /* unused */
      u8_t dl, dh; /* 8-bit general registers */
      [...]        /* unused */
    };
  };
} reg86_t;
```

Why the outer struct? "Better for forward declarations" (Kees J.Bot)