Program memory: pointers

COSC 208, Introduction to Computer Systems, 2020-09-22

Announcements

- Project 1 Part 2 due next Thursday (September 30)

Outline

- Warm-up
- Pointers
- Pointers as parameters
- Extra practice

Pointers

- A special type of variable that holds a memory address
  - A pointer "points to" a location in memory
- Declaration consists of:
  - Type of value stored in memory at the address stored in the pointer variable
  - Asterisk (`*`)
  - Name of pointer variable
  - Example: `int *p` --- declares a pointer variable `p` that refers to a memory location where an integer is stored
- Can be used to indirectly read and write another variable
- Address-of (`&`) operator obtains the memory address where a variable is stored Example:

```
int i = 42;
int *p = &i; // p now contains the address
             //where the value of variable i is stored
             //p “points-to” i
```

- De-reference (*) operator is used to refer to the value stored at the memory address contained in the pointer variable
  - Example:

```
printf("%d\n",*p); // Outputs 42, because p contains the memory
address where i is stored (i.e., p points-to i), and the variable
i currently contains the value 42
```

  - Example:

```
*p = *p + 1; // Updates the value in variable i to 43, because p
contains the memory address where i is stored (i.e., p points-to
i)
```

- *What happens if we assign a value to the pointer variable without dereferencing the pointer?* --- we update the memory address stored in the pointer variable; we point to a new location in memory

Practice with pointers

- Q1: *Write a snippet of code that:*
  1. *Declares a `char` variable called `orig` and initializes it with the value `'A'`*
  2. *Declares and initializes a pointer called `ptr` that points to `orig`*
  3. *Uses the pointer to update the value stored in `orig` to `'B'`*

```
char orig = 'A';
char *ptr = &orig;
*ptr = 'B';
```

- Q2: *What is the output of this program?* -- draw a memory diagram to help

```
int main() {
    int a = 1;
    int b = 2;
    char c = 'C';
    int *x = &a;
    int *y = &b;
    char *z = &c;
    printf("%d %d %c\n", *x, *y, *z);
    *x += 1;
    b += 2;
    *z = 'D';
    printf("%d %d %c\n", *x, *y, *z);
    printf("%d %d %c\n", a, b, c);
    x = y;
    *x += 10;
    a += 20;
    printf("%d %d\n", *x, *y);
    printf("%d %d\n", a, b);
}
```

Output

```
1 2 C
2 4 D
2 4 D
```

```
14 14
22 14
```

Pointers as parameters

- Functions can take pointers as parameters
  - Similar to a pointer declaration, include an asterisk before the paramter name: e.g., `void foo(int *a)`
  - Function gets a copy of the memory address stored in the pointer variable, not a copy of the value stored at that memory address
- Example

```c
void value(int a) {
    a = 2;
}
void pointer(int *b) {
    *b = 3;
}
int main() {
    int v = 1;
    int *p = &v;
    value(v);
    printf("%d\n", v);
    pointer(p);
    printf("%d\n", v);
}
```

Extra practice

- Q6: *Write a function called `transfer` that takes two integer pointers and an amount to transfer and moves the specified amount from one integer to the other.*

```c
void transfer(int *from, int *to, int amount) {
    *from -= amount;
    *to += amount;
}
```

- Q7: *Write a function called `swap` that takes two integer pointers and exchanges the integer values.*

```c
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```