C: functions

COSC 208, Introduction to Computer Systems, 2021-09-03

Announcements

- Complete readings and pre-class questions before each class period
- Lab 1 due tonight: Friday, 11 p.m.

Outline

- Warm-up
- Defining functions
- Program stack

Warm-up

This warm-up focuses on the biological process of cell division: each cell splitting into two each round.

- Q1: *Write a function called `cells` that takes the number of rounds of cell division that occur and computes the total number of cells that will exist after the specified number of rounds (assuming you started with a single cell).*

```
int cells(int rounds) {
    int total = 1;
    for (int i = 0; i < rounds; i++) {
        total = total * 2;
    }
    return total;
}
```

- Q2: *Write a function called `rounds` that takes the number of cells that should exist and computes the number of rounds of cell division that must occur to have ta least that many cells (assuming you started with a single cell).*

```
int rounds(int cells) {
    int total = 1;
    int rounds = 0;
    while (total < cells) {
        total *= 2;
        rounds++;
    }
    return rounds;
}
```

Defining functions

- *What is the syntax for defining a function?*

```
return-type function-name(parameter-type parameter-name, ...) {
    /* STATEMENTS */
    return value;
}
```

```
int main() {
    int result = add(1,2); // Compiler doesn't know
    printf("%d\n", result);            // if this call is valid
}
int add(int x, int y) {
    return x+y;
}
```

- Function prototypes
  - C compiler must know a function's return type and the number and type of its parameters before it encounters any calls to that function
  - Function prototype provides a function's return type, name, and number and type of its parameters, but not its body

```
return-type function-name(parameter-types);
```

```
int add(int, int);
```

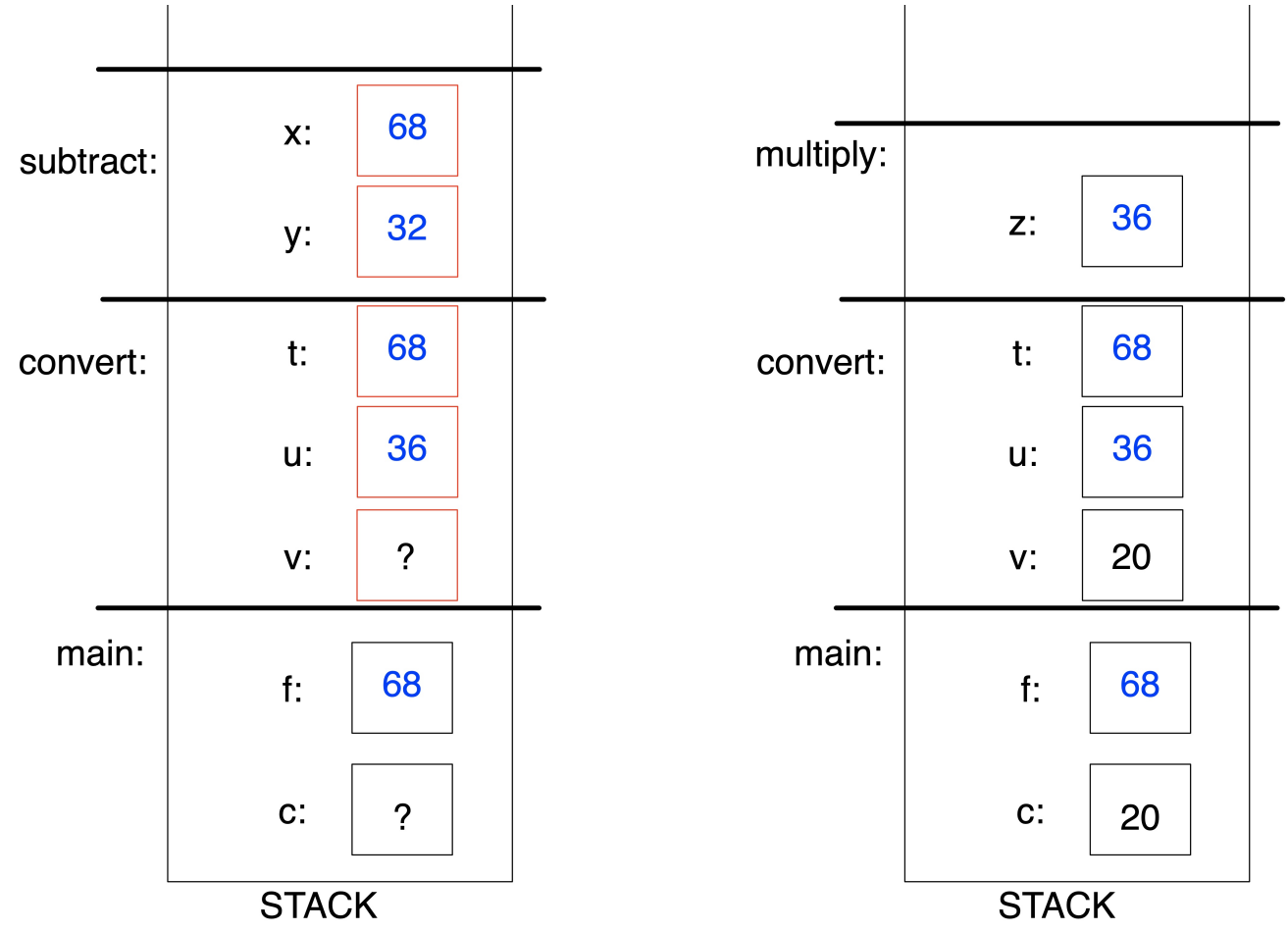
  - Include function prototype before any calls to the function --- usually at the top of a file
  - Standard header files (e.g., `stdio.h`) include such prototypes

Program stack

- Stores data for functions that are currently executing
- Stored in random access memory (RAM)
- Composed of stack frames
  - A stack frame contains the values stored in a functions's local variables and formal parameters
  - The size of the stack frame is determined by the number and type of local variables and formal parameters
  - A stack frame is added to the top of the stack when a function is called
  - A stack frame is removed from the top of the stack when a function returns

- Example

```c
#include <stdio.h>
int multiply(int z) {
    return z * 5 / 9;
}
int subtract(int x, int y) {
    return x - y;
}
int convert(int t) {
    int u = subtract(t, 32);
    int v = multiply(u);
    return v;
}
int main() {
    int f = 68;
    int c = convert(68);
    printf("%dF is %dC\n", f, c);
}
```



- C is pass-by-value → changes made to a variable inside a function will not be preserved outside of the function

Q3:

Draw the contents of the stack immediately before the program prints "1 x 2"

```c
int squared(int base) {
    return base * base;
}
int dbl(int num) {
    printf("%d x 2\n", num);
    return num * 2;
}
int two(int exponent) {
    int result = 1;
    for (int i = 0; i < exponent; i++) {
        result = dbl(result);
    }
    return result;
}
int main() {
    int n = 3;
    int s = squared(3);
    printf("%d^2 is %d\n", n, s);
    int t = two(3);
    printf("2^%d is %d\n", n, t);
}
```

Q4:

What is the output of this program?

```c
#include <stdio.h>
int copy(int a, int b) {
    a = b;
    return b;
}
int main() {
    int x = 3;
    int y = 7;
    int z = copy(x, y);
    printf("%d %d %d\n", x, y, z);
}
```

```
3 7 7
```

Q5:

Draw the contents of the stack immediately before the program prints "n=2"

```
int recurse(int n) {
    printf("n=%d\n", n);
    if (n == 1) {
        return 0;
    }
    else {
        return 1 + recurse(n/2);
    }
}
int main() {
    int x = 16;
    int r = recurse(x);
    printf("result=%d\n", r);
}
```

Q6:

If `main` initialized `x` to `64` (instead of `16`), how many stack frames would exist immediately before the program printed "n=2"?_

```
7 —  (bottom) `main`, `recurse(64)`, `recurse(32)`, `recurse(16)`,
`recurse(8)`, `recurse(4)`, `recurse(2)` (top)
```

- *What does your answer to questions 5 and 6 tell you about program efficiency?*

— the more nested function calls there are, the more stack memory the program requires;

- creating an iterative (instead of recursive) implementation would be more memory efficient

Extra practice

- *Write a function called `print_x` that takes an integer n that prints the letter `X` n lines high and n char wide. You can assume n is an odd number. For example, if n = 9, the program's output would be:*

```
\       /
 \     /
  \   /
   \ /
    X
   / \
  /   \
 /     \
/       \
```

```c
#include <stdio.h>
void print_x_helper(int n, int r) {
    for (int c = 0; c < n; c++) {
        if (c == r) {
            printf("\\");
        } else if (c == n-1-r) {
            printf("/");
        } else {
            printf(" " );
        }
    }
    printf("\n");
}
void print_x(int n) {
    int r = 0;
    while (r < n/2) {
        print_x_helper(n, r);
        r++;
    }
    for (int c = 0; c < n; c++) {
        if (c == r) {
            printf("X");
        } else {
            printf(" ");
        }
    }
    printf("\n");
    r++;
    while (r < n) {
        print_x_helper(n, r);
        r++;
    }
}
```