# 601.220 Intermediate Programming

Spring 2023, Day 7 (Feb 6th)

## Today's agenda

- Exercise 6 review
- Function declarations, passing arrays to functions, recursion
- Exercise 7

## Reminders

- HW1 due Friday, Feb 10th

## Note about HW1

Error messages will need to be printed to `stderr` using `fprintf`

E.g.:

```
fprintf(stderr, "Invalid input\n");
```

## Exercise 6 review

Opening input and output files:

```
FILE *in = fopen(filename, "r");
if (in == NULL) {
  fprintf(stderr, "Could not open '%s' for reading\n", filename);
  return 1;
}

FILE *out = fopen("output.txt", "w");
if (out == NULL) {
  fclose(in);
  fprintf(stderr, "Could not open 'output.txt' for writing\n");
  return 1;
}
```

## Exercise 6 review

```
while ( parse == 2 ) {
```

Reading principal and APR from input file:

```
parse = fscanf(in, "%f %f", &p, &r);
```

This needs to go before main loop starts, and also at end of body of main loop.

## Exercise 6 review

Computing accumulated principal (in `compound_interest` function):

```
const float t = 1.0;
if (n > 0) {
  return p * pow(1.0 + r/n, n*t);
} else {
  return p * exp(r*t);
}
```

## Exercise 6 review

close in & out
?

Checking for errors after main loop terminates:

-1

```
if (parse != EOF) {
  fprintf(stderr, "Error reading input\n");
  return 1;
}
if (ferror(in)) {
  fprintf(stderr, "Input error indicator was set\n");
  return 1;
}
if (ferror(out)) {
  fprintf(stderr, "Output error indicator was set\n");
  return 1;
}
```

?

*
```
fclose(in);
fclose(out);
```

## Exercise 6 review

Closing input and output files:

```
fclose(in);
fclose(out);
```

## Day 7 recap questions

1. How do you get the number of elements of an integer array?
2. Is the size of a string array the same as the string length?
3. What is the difference between a function declaration and a function definition?
4. Can you have two functions with the same function name in a program?
5. How does passing an integer array to a function differ from passing a single integer variable into a function?
6. How can you make an array that is passed into a function not modifiable?
7. What is the down-side to recursion?

1. How do you get the number of elements of an integer array?

```
int arr[10];

// ...

size_t num_elts = sizeof(arr) / sizeof(int);
printf("%lu\n", num_elts); // prints 10
```

Note that this will **not** work if `arr` is a function parameter. (Array parameters are not actually arrays. We'll see what they are soon.)

## 2. Is the size of a string array the same as the string length?

No.

If an array of `char` elements will be used to store a string value, its number of elements must be *at least* the string length plus 1. (Enough room to store the characters in the string, and the NUL terminator.)

It is totally fine for a `char` array to have more room than necessary. In this case, the elements after the NUL terminator aren't used.

## 3. What is the difference between a function declaration and a function definition?

Function declaration: just tells the compiler the important details about the function: its name, return type, and parameter types. It will use this information to check *calls* to the function. A.k.a. a "function prototype".

Function definition: defines the body (code) of the function.

Each use of a function in a program should be preceded by either a declaration or definition.

#include <stdio.h>

## 4. Can you have two functions with the same function name in a program?

No, not in C.

(C++ does allow this, and calls this possibility “function overloading.”)

## 5. How does passing an integer array to a function differ from passing a single integer variable into a function?

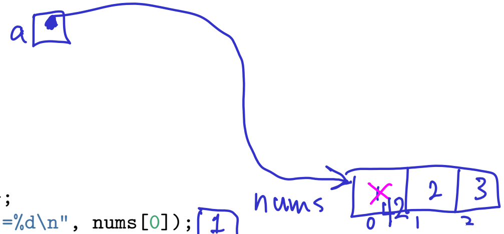
~~Yes~~: arrays are passed by *reference*, not by value. This means that the called function can change the values in the array.

# Example

```
// pbr.c:
#include <stdio.h>
void f(int a[]) {
  a[0] = 42;
}

int main(void) {
  int nums[3] = {1, 2, 3};
  printf("Before: nums[0]=%d\n", nums[0]);
  f(nums);
  printf("After: nums[0]=%d\n", nums[0]);
  return 0;
}
```

$\boxed{1}$ $\boxed{42}$



```
$ gcc -std=c99 -Wall -Wextra -pedantic pbr.c
$ ./a.out
Before: nums[0]=1
After: nums[0]=42
```

## 6. How can you make an array that is passed into a function not modifiable?

Declare the element type as being `const`. Example:

```
// constelem.c:
#include <stdio.h>
void f(const int a[]) {
  a[0] = 42;
}

int main(void) {
  int nums[3] = {1, 2, 3};
  f(nums);
  return 0;
}
```

```
$ gcc -std=c99 -Wall -Wextra -pedantic constelem.c
constelem.c: In function ‘f’:
constelem.c:3:8: error: assignment of read-only location ‘*a’
    3 |   a[0] = 42;
      |        ^
```

## 7. What is the down-side to recursion?

Each function call in C requires the creation of a run-time data structure called a *stack frame* to store parameter values, allocate storage for local variables, and keep track of other information about the function call.

The amount of memory available for stack frames is limited.

A “deep” recursion could create a large number of stack frames, which could require more memory than is available. This results in a “stack overflow” which will crash the program.

So, avoid deep recursion.

## Recursion tips

- The first thing a recursive function must do is to see whether a *base case* has been reached
- If a base case hasn't been reached, find a smaller instance of the problem, solve it recursively, then extend the solution to the smaller problem so that the entire problem is solved
  - Example:

```
int sum_ints(int n) { // compute sum of integers 1..n
  if (n == 1) { return 1; }
  return sum_ints(n-1) + n;
}
```

- The subproblem solved recursively should be as *large* as possible
  - So that very little work is required to extend the solution to be a solution to the overall problem

## Exercise 7

- Functions and recursion!
- Talk to us if youu have questions!

## Notes

## Notes

## Notes

## Notes

## Notes