APSC 160 review

Functions, variables, operators, loops
Modularity

Geoff’s self-checklist:
- ❑ Start iClicker Cloud
- ❑ Record lecture

Announcements

- Labs begin next week Monday Sep.14
  - Lab 1 activities will be available on the course webpage (over the weekend)
  - Please do pre-lab activities before attending your first lab session
    - I will do my best to have them released with sufficient time for Monday's sections

Variable swap

- Suppose that `var1` and `var2` are variables of type `int`.
- Which of the following code segments swaps the value of these two variables?

```
a) int temp = var1;
   var1 = var2;
   var2 = temp;
```

```
b) int temp = var1;
   var2 = var1;
   var1 = temp;
```

```
c) var1 = var2;
   var2 = var1;
```

```
d) int temp1 = var1;
   int temp2 = var2;
   temp1 = temp2;
   var2 = var1;
```

Operator precedence

- Assume that the following variable declarations have been made:

```
int a = 16;
int b = 4;
double c = 1.5;
```

- What value is assigned to the variable `d` by the following statement?

```
double d = c + a * b;
```

a) 65.0

b) 65.5

c) 65

d) 66

Division with integers and floating points

- Assume that the following variable declarations have been made:

```
int a = 16;
int b = 4;
double c = 1.5;
```

- What value is assigned to the variable `d` by the following statement?

```
double d = b / a;
```

a) 1

b) 0.0

c) 0.25

d) 4

Boolean logic

- Suppose that variable `t` is a variable that has a value evaluating to `true`, and `f` is a variable that has a value evaluating to `false`. Which one of the following expressions evaluates to `false`?

a) `t && !f`

b) `!t && f`

c) `t || f`

d) `!(t && f)`

e) `t || (!f && !t)`

Indentation

- Consider the following (poorly) indented code segment.
- What are the values of `a`, `b`, and `r` after this code segment has executed?

```
int r;
int a = -5;
int b = 6;
if (a < 0 || b > 0)
  r = 1;
else
  r = 2;
  a = 0;
```

a) `a = 0, b = 6, r = 2`

b) `a = 0, b = 6, r = 1`

c) `a = -5, b = 6, r = undefined`

d) `a = -5, b = 6, r = 2`

e) None of the above

Loops

- Consider the following code segment: What values do `i` and `j` have after this code segment has completed execution?

```
int i = 1;
int j = 0;

while (i < 5 && j < 4) {
  j = j + i;
  i++;
}
printf("i = %d, j = %d", i, j);
```

a) `i = 4, j = 5`
b) `i = 5, j = 5`
c) `i = 4, j = 6`
d) `i = 5, j = 6`
e) None of the above

Loops

- How many times is the `printf` statement executed in the following C code?

```c
int x = 1;
while ( x < 15/4 ) {
  printf("x = %d\n", x);
  x++;
}
```

a) never

b) once

c) twice

d) three times

e) four or more times (or infinite loop)

Nested loops

- Consider the following code segment.
- What values do count1 and count2 have after this code segment has completed execution?

a) `count1=3, count2=8`

b) `count1=3, count2=10`

c) `count1=2, count2=8`

d) `count1=2, count2=9`

e) None of the above

```
int i;
int j;

int count1 = 0;
int count2 = 0;

for (i = 0; i < 3; i++) {
  count1++;
  for (j = 1; j < 4; j++) {
    count2++;
  }
}
```

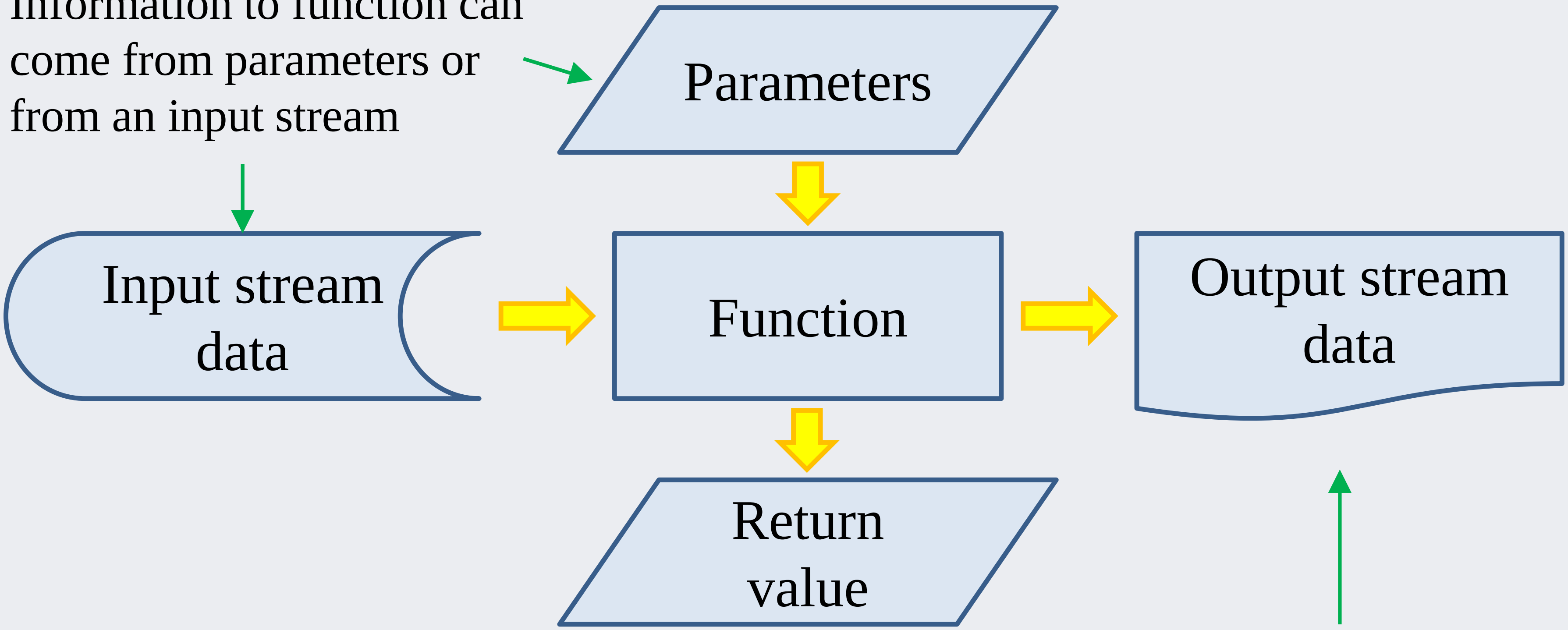
Modular programming

- Imagine a 10,000-line program that consists of only one function: main
  - Extremely difficult to **debug** and maintain
  - Extremely difficult to **re-use** any part of the code

- Modularity: A design principle used to manage the complexity of larger systems / programs
  - used in many engineering disciplines
  - in software development, modularity can be implemented using **functions**
    - break the program into smaller modules (functions)

Functions

Information to function can come from parameters or from an input stream



Information from function can come through a return value or an output stream

Function parameters

- **Actual** parameter
  - Value(s) or variable(s) specified by the function *caller*
- **Formal** parameter
  - Variables found in the signature/header of the *function* itself

- Formal parameters must match with actual parameters in *order*, *number*, and *data type*

Calling functions

What happens when a function is called

1. Copy parameter values/addresses (if any) from caller to function, regardless of variable names
2. Execute the function. Function ends when we reach *any* return statement
3. Pass back the answer (if any) via the return statement
4. Destroy all local variables in the *function*
5. Return control to the caller
6. Finish the rest of the calling statement (after replacing the function call with the return value, if any)

Function parameters

- Parameters may be **passed by value** ("call-by-value")
  - the *value* of the actual parameter is copied to the formal parameter when the function is called

- The actual parameters and formal parameters are *different variables in memory*, even if they are named the same

- If you change the value of the formal parameter, this does **not** affect the value of the actual parameter back in the caller's memory

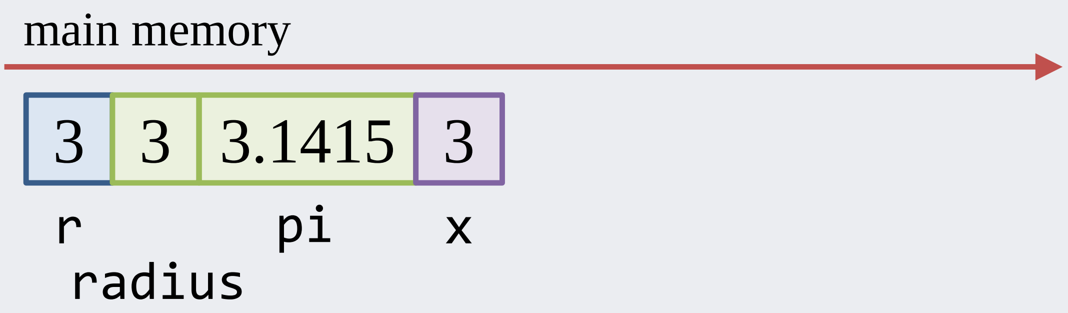
Function parameters and the call stack

Example

```
// ...
int r = 3;
double area = circleArea(r);
// ...
```

```
double circleArea(double radius){
     double pi = 3.1415;
     double sq_r = square(radius);
     return sq_r * pi;
}
```

```
double square(double x){
  return x * x;
}
```



main memory

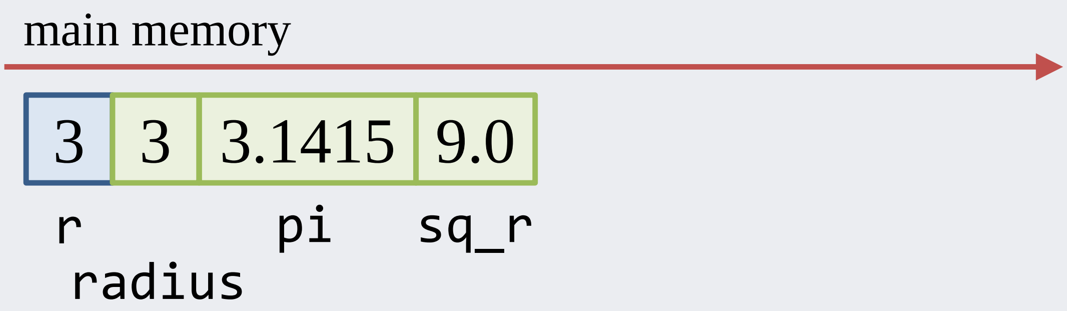
Function parameters and the call stack

Example

```
// ...
int r = 3;
double area = circleArea(r);
// ...
```

```
double circleArea(double radius){
     double pi = 3.1415;
     double sq_r = square(radius);
     return sq_r * pi;
}
```

```
double square(double x){
  return x * x;
}
```



main memory

3	3	3.1415	9.0

r
radius
pi
sq_r

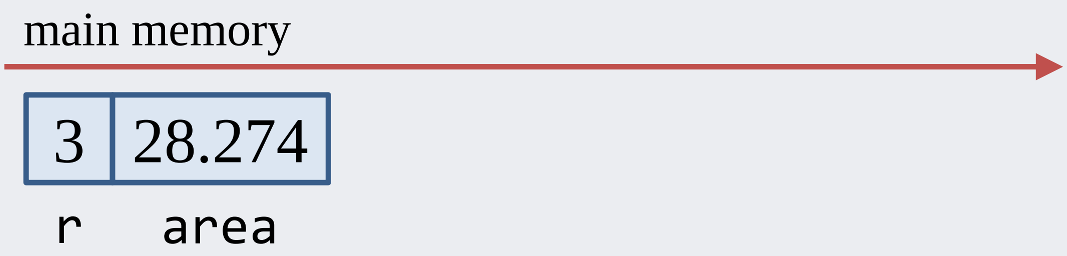
Function parameters and the call stack

Example

```
// ...
int r = 3;
double area = circleArea(r);
// ...
```

```
double circleArea(double radius){
      double pi = 3.1415;
      double sq_r = square(radius);
      return sq_r * pi;
}
```

```
double square(double x){
  return x * x;
}
```



main memory

r	area
3	28.274

Functions and parameters

- Consider the following code segment
- Fill in the blanks to show what is output to the screen when the program runs

```c
void myFunc(int a, int b) {
  a = a + 4;
  b = b – 4;
  printf("In myFunc a = %d b = %d\n", a, b);
}

int main() {
  int a = 5;
  int b = 7;
  myFunc(a, b);
  printf("In main a = %d b = %d\n", a, b);
  return 0;
}
```

```
In myFunc a = ___ b = ___
In main a = ___ b = ___
```

Next class

- APSC 160 review – arrays, algorithms, searching
- Labs start next week! Check website for details