Function I/O

Last updated 6/16/23

These slides introduce pointers in functions

Function I/O

- Program Structure

Includes

Function Declarations

```
void main(void){
   …
   foo = fun1(a, b);
   fun2(2, c);
   if(fun1(c, d)) {
      …
   }
}
```

Function 1 Definition

Function 2 Definition

Function I/O

- Function Input and Output
  - Input – through actual parameters
  - Output – through return value
    - Only one value can be returned
  - User Input/Output – through side effects
    - printf
    - scanf

```
int main(void){
  float checking;
  float savings;
  float int_rate;
   ...
  checking = update_acct(checking, int_rate);
  savings = update_acct(savings, int_rate);
  return 0;
}

float update_acct(float bal, float ir){
  bal += bal * ir;
  return bal;
}
```

Function I/O

- Pointers and functions

  - Pointers allow us to use called functions to change values in the calling function

  - Instead of passing variables in the parameter list (remember copies are made and then relinquished) we can pass pointers

  - Pointers allow us to modify the calling programs variables by memory reference

Function I/O

- Function Declaration
  - Indicate that a pointer is being passed in the Formal Parameter List

```
void update_acct(float * balance_ptr,   float int_rate);
                   passing a pointer      passing a float
                   of type float
```

Function I/O

- Function Definition
  - Indicate that a pointer is being passed in the Formal Parameter List
  - Operate on the variables pointed to by the pointers via the dereference operator

```
void update_acct(float * balance_ptr, float int_rate){
    *balance_ptr = *balance_ptr  + *balance_ptr * int_rate;
     return;
}
```

the value pointed to by balance_ptr is assigned the value of the value pointed to by balance_ptr + the value pointed to by balance_ptr times int_rate

Function I/O

```
void update_acct(float* balance_ptr, float int_rate){
    *balance_ptr += *balance_ptr * int_rate;
    return;
}
```

- Function Call
  - Pass a pointer variable in the Actual Parameter List

    or
  - Pass the address to the variable in the Actual Parameter List

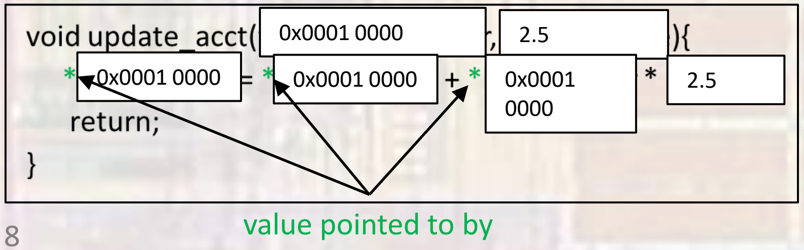
```
int main(void){
   float checking;
   float savings;
   float int_rate;
   float * check_ptr;                // ptr variable to a float variable
   check_ptr = &checking
   …
   update_acct(check_ptr, int_rate);        // using ptr variable
   update_acct(&savings, int_rate);         // using address
   return 0;
}
```

Function I/O

- Usage
  - Pass a pointer variable in the Actual Parameter List

```
int main(void){
  float checking;                    // stored in 0x0001 0000
  float int_rate;                     // stored in 0x0001 0004
  Int_rate = 2.5;
  checking = 1000;
  float * check_ptr;                 // ptr variable to a float variable
  check_ptr = &checking              // check_ptr has the value 0x0001 0000
  …
  update_acct(check_ptr, int_rate);  // looks like  update_acct(0x0001 0000, 2.5)
  return 0;
}
```

```
void update_acct(float* balance_ptr, float int_rate){
    *balance_ptr = *balance_ptr  + *balance_ptr * int_rate;
    return;
}
```

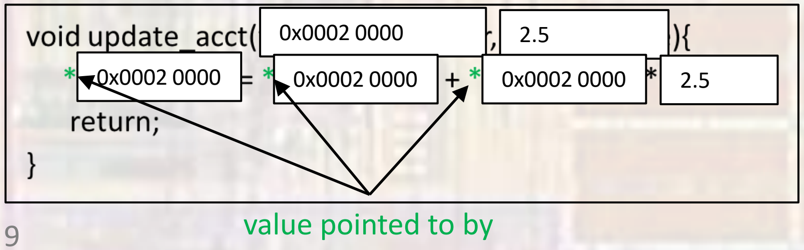

void update_acct(0x0001 0000 , 2.5){
* 0x0001 0000 = * 0x0001 0000 + * 0x0001 0000 * 2.5
return;
}
value pointed to by

Function I/O

- Usage
  - Pass the address to the variable in the Actual Parameter List

```
int main(void){
   float savings;                    // stored in 0x0002 0000
   float int_rate;                   // stored in 0x0001 0004
   Int_rate = 2.5;
   savings = 1000;
   …
   update_acct(&savings, int_rate);  // looks like  update_acct(0x0002 0000, 2.5)
   return 0;
}
```

```
void update_acct(float* balance_ptr, float int_rate){
    *balance_ptr = *balance_ptr + *balance_ptr * int_rate;
    return;
}
```



```
void update_acct(0x0002 0000, 2.5){
    *0x0002 0000 = *0x0002 0000 + *0x0002 0000 * 2.5
    return;
}
```

Function I/O

- Example
  - Swap 2 values – not possible with only 1 return value

```
int main(void)
    int a;
    int b;
     …
    swap(&a, &b);
    return 0;
}

void swap(int * x, int * y){
      int tmp;
      tmp = *x;
     *x = *y;
     *y = tmp;
     return;
}
```

let a = 5 at memory location 0x1000
let b = 8 at memory location 0x1004

swap(address of a, address of b) = swap(0x1000, 0x1004)

tmp = ?
tmp = value pointed to by 0x1000 = 5
value pointed to by 0x1000 = value pointed to by 0x1004 = 8
value pointed to by 0x1004 = tmp = 5

a is now 8
b is now 5

Function I/O

- Example
  - Provide the quotient and remainder of a division

```
int main(void){
   int numerator;
   int denominator;
   int quotient;
   int remainder;
…
   divide(numerator, denominator, &quotient, &remainder);
   return 0;
}

void divide(int num, int den, int * quo, int * rem){
   *quo = num / den;
   *rem = num % den;
   return;
}
```

Function I/O

- Reflection
  - Finally, we can understand our scanf() function
    - Reads in 1 or more values and stores them in variables
      - Cannot rely on a single return value

```
int foo;
float boo;
scanf("%i, %f", &foo, &boo);
```

scanf is very sophisticated but we can see that:

to allow more than 1 thing to be read (modified) at a time
scanf expects POINTERS for the variables passed in it's parameter list!!!