

Function I/O

Last Updated 12/16/20

Function I/O

- These slides introduce function I/O using pointers
- Upon completion: You should be able to interpret and code functions with pointers

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
 - Input/Output – through side effects
 - printf
 - scanf

Function I/O

- Function Input and Output

```
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 passed variables by memory reference

Function I/O

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

```
void update_acct(float* balance_ptr, float int_rate);
```

Function I/O

- Pointers and functions

- 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;
}
void update_acct2(float* balance_ptr, float int_rate){
    *balance_ptr += *balance_ptr * int_rate;
    return;
}
```


Function I/O

- Pointers and functions

```
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

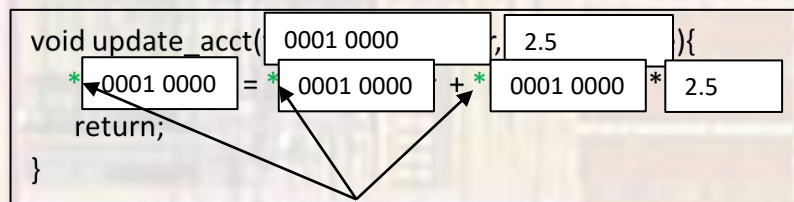
- Pointers and functions

- 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;
}
```



value at location

Function I/O

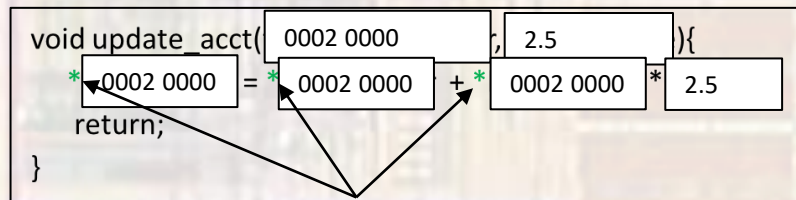
- Pointers and functions

- 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;
}
```



Function I/O

- Pointers and functions
 - 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;
}
```

Function I/O

- Pointers and functions
 - Provide the quotient and remainder of a division

```
int main(void){
    int num;
    int den;
    int quo;
    int rem;
    ...
    divide(num, den, &quo, &rem);
    return 0;
}

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

Function I/O

- Pointers and functions
 - Finally, we can understand our scanf() function
 - Reads in 1 or more values and stores them in variables

```
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!!!