Functions w/ Pointers

Last Updated 9/7/21

Functions w/ Pointers

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

Functions w/ Pointers

- Function Input and Output

```
float update_acct(float bal, float ir);

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;
}
```

Functions w/ Pointers

- 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 of the values are made and then relinquished upon return) we can pass pointers
  - Pointers allow us to modify the passed variables (values) by memory reference

Functions w/ Pointers

- Function Declaration
  - Indicate that a pointer is being passed in the Formal Parameter List
    - The pointer parameter includes the *

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

Functions w/ Pointers

- Function Definition
  - Indicate that a pointer is being passed in the Formal Parameter List
    - The pointer parameter includes the *
  - Operate on the variables pointed to by the pointers via the dereference operator * (value pointed to by)

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

Functions w/ Pointers

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

Functions w/ Pointers

- 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( 0001 0000 r,  2.5 ){
   * 0001 0000  = *  0001 0000  + *  0001 0000  *  2.5
   return;
}
```

= 1000 + 1000 * 2.5

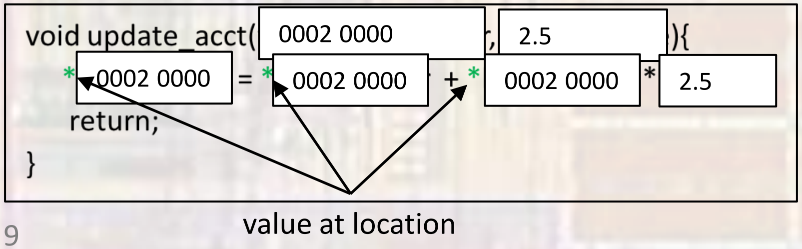
value at location

Functions w/ Pointers

- 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;
}
```



Functions w/ Pointers

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

```
void swap(int* x, int* y);

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;
}
```

Functions w/ Pointers

- Example
  - Provide the quotient and remainder of a division

```
void divide(int num, int den, int* quo, int* rem);

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;
}
```

Functions w/ Pointers

- 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!!!