CSE 142
Programming I

Pointers and Output Parameters

Review: Functions

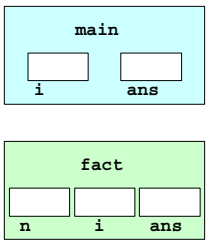
```
int factorial(int n){
  int product, i;
  product = 1;
  for (i=1; i<=n; i++)
    product = product * i;
  return product;
}
```


Review: Local Variables

```
int main(void){
  int i, ans;

  i = 3;
  ans = factorial(i+1);

  return 0;
}
```



Local Variables: Summary

- Parameters and variables defined in a function are local to it
- Created on entry and deleted on exit
- Arguments are copied into parameters
- No global variables in 142!

Call by Value

```
void swap
  (int x, y){
  int temp;

  temp = x;
  x = y;
  y = temp;
  return;
}
```

```
int main(void){
  int x = 4;
  int y = 5;

  swap(x, y);

  return 0;
}
```


Call by Value

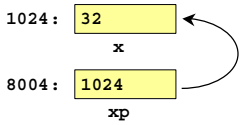
- Call by Value means that the value of the argument is found and copied into the parameter
- All functions in C use call by value
  - ➤What limitations do we have?
  - ➤How do we get around those limitations?

Values and Locations

- Swap needed to know more than the values of its arguments—it needed to know *where* they were
- Remember: each variable is stored in a place in memory, and each place in memory has an address

New Type: Pointer

- A pointer contains the address of a variable
- Given that address, the computer can find the variable



Memory Snapshot

Address	Value	Type
1	2	int *
2	6	int
3	214515513	int *
4	214515513	int
5	10	int
6	5	int
7	5	int *
8	5	int *

Memory Graphically

1(alice):	5(ely):
2(bob):	6(fran):
3(charlie):	7(gomer):
4(doug):	8(hank):

New Types

- We had
  - int
  - char
  - double
- Now we have
  - int *
  - char *
  - double *

New (Unary) Operators

- & is the "address of" operator—it finds the address of a variable
- * is the "dereference" operator—it finds the variable a pointer is pointing to

What's The Type?

- The & operator takes a variable and produces a pointer to it
  - if `x` is an int, `&x` is an int pointer
- The * operator takes a pointer and gives you the variable it points to
  - if `y` is an int pointer, then `*y` is an int

What's the Type?

- What are the types of
  - alice
  - bob
  - *charlie
  - &doug
  - *ely

Some Simple Commands

- What if we execute
  - `alice = &fran;`
  - `alice = fran;`
  - `doug = *gomer;`
  - `ely = gomer;`
  - `charlie = *ely;`
  - `ely = &bob;`

Why Use Pointers

- In 142 we'll use pointers for output parameters
  - We can solve the problem of only having one return value!
- In more advanced programming, pointers are used to create fancy data structures

Output Parameters

- If everything is pass by value, how can we change a variable in the calling function?
  - Do not pass the variable, pass a pointer to the function
  - The pointer is copied, but still points to the same place in memory
  - Now if we access that memory location, we can modify variables outside of our function

Swap Rewritten

- Remember what happened when we wrote swap:
  - The values got copied and swapped in the function
  - The values did not get swapped in main()
- Idea! Pass in pointers to the variables and use them to swap the values

Swap Rewritten

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

  temp = *x;
  *x = *y;
  *y = temp;

  return;
}
```


What's Happening Here?

- Remember: x and y are pointers
  - ➤If we changed x or y, we would change the location the pointed to
  - ➤If we change *x or *y, we change the value in the variable they point to

What Happens in main()?

```
int main(void){
  int a = 1;
  int b = 17;

  swap(&a, &b);
  printf("%d, %d\n", a, b);

  return 0;
}
```


Another Example

```
/* reflect point through origin */
void reflect(double *x, double *y){

  *x = *x * -1;
  *y = *y * -1;

  return;
}
```


Simple Sorting

```
void sort2(int *num1, int *num2){

  if (*num1 > *num2){
     swap(num1, num2);
  }

  return;
}
```


Why No & in Call to swap()?

- num1 and num2 were already pointers
  - ➤swap(&num1, &num2) would pass pointers to pointers to integers—yuck!
  - ➤We would end up swapping the pointers, not the values!

* and * and *

- * now has 3 meanings in C (don't hang the messenger!):
  - ➤Multiplication
  - ➤Pointer declarations
  - ➤Pointer dereference

& in scanf()

- The & in scanf() should make sense now!
  - ➤scanf() must be able to change the values of our variables
  - ➤so, scanf() takes as arguments pointers to the variables
  - ➤Why doesn't printf() use pointers?

A Screwy scanf() Example

```
void getCoords(double *x, double *y){
  printf("Enter coordinates:");
  scanf("%lf %lf", x, y);
}
```

- Why no ampersands?

Remember!

- A pointer stores an address
  - ➤Reassigning a pointer changes *where it points*
- The * operator finds the variable the pointer points to
- The & finds the address of a variable

A Test-Like Question (hint!)

```
void muddle(int x,
  int *y, int *z){

  y = z;
  x = *y;
  *z = x * 2;
  x = 3;
  return;
}
```

```
int main(void){
  int x=1, y=2;
  int z=3;

  muddle(z,&y,&x);
  printf("%d,%d,%d
     \n", x, y, z);
  return 0;
}
```


Larger Program

- Find out how much change the user has
- Tell the user if they have the optimal arrangement of coins
  - ➤By optimal, we mean the smallest number of coins
- If sub-optimal, then tell them the best arrangement

Exercise in Top-Down Design

- What we have to do: