600.120 Intermediate Programming, Spring 2017*

Misha Kazhdan

*Much of the code in these examples is not commented because it would otherwise not fit on the slides. This is bad coding practice in general and you should not follow my lead on this.
Why?

*ptr = *ptr++;  

In terms of precedence:

*ptr = *(ptr++);

“ptr++” increments the pointer but returns the old value, so the RHS side becomes

*(ptr++) → ra[0];
ptr → ra+1

This gives

ra[1]=ra[0];

#include <stdio.h>

int main( void )
{
    int ra[5] = { 1, 2, 3, 4, 5 };
    int *ptr = ra;
    for( int i=0 ; i<5 ; i++ ) *ptr = *ptr++;
    printf( "array is: [ " );
    for( int i=0 ; i<5 ; i++ ) printf( "%d", ra[i] );
    printf( " ]\n" );
    return 0;
}

>> ./a.out
array is: [ 1 1 1 1 1 ]
>>
Why?

In addition to giving the wrong result, this is bad code because in the last iteration we are setting

\[ \text{ra}[5] = \text{ra}[4]; \]

which is an out-of-bounds access!

RHS side becomes

\[ *(\text{ptr}++) \rightarrow \text{ra}[0]; \]
\[ \text{ptr} \rightarrow \text{ra}+1 \]

This gives

\[ \text{ra}[1] = \text{ra}[0]; \]

\texttt{#include <stdio.h>}
\texttt{int main( void )}
\{ 
  \texttt{int ra}[5] = \{ 1, 2, 3, 4, 5 \};}
\texttt{int *ptr = ra;}
\texttt{for( int i=0 ; i<5 ; i++ ) *ptr = *ptr++;}
\texttt{printf( "array is: [ " );}
\texttt{for( int i=0 ; i<5 ; i++ ) printf( "\%d ", ra[i] );}
\texttt{printf( "\n" );}
\texttt{return 0;}
\}

\texttt{\$ ./a.out}
array is: [ 1 1 1 1 1]

In addition to giving the wrong result, this is bad code because in the last iteration we are setting \texttt{ra}[5] = \texttt{ra}[4]; which is an out-of-bounds access!
Why?

When `swap` is called, it is called with copies of the pointers `ptr1` and `ptr2`
⇒ Swapping the addresses stored in `list1` and `list2` happens with the stack frame of `swap` and is not visible when the function returns
Why?

When `swap` is called, it is called with copies of the pointers `ptr1` and `ptr2`.

⇒ Swapping the addresses stored in `list1` and `list2` happens with the stack frame of `swap` and is not visible when the function returns.
Why?

When `swap` is called, it is called with copies of the pointers `ptr1` and `ptr2`.

⇒ Swapping the addresses stored in `list1` and `list2` happens with the stack frame of `swap` and is not visible when the function returns.
Why?

When `swap` is called, it is called with copies of the pointers `ptr1` and `ptr2`.

⇒ Swapping the addresses stored in `list1` and `list2` happens with the stack frame of `swap` and is not visible when the function returns.

```c
#include <stdio.h>
void swap( char * list1 , char * list2 )
{
    char * temp = list1;
    list1 = list2;
    list2 = temp;
}
int main( void )
{
    char ar1[] = { 'a', 'b', 'c', 'd', 'e' };
    char ar2[] = { 'f', 'g', 'h', 'i', 'j' };
    char *ptr1 = ar1, *ptr2 = ar2;
    swap( ptr1 , ptr2);
    for( int i=0 ; i<5 ; i++ ) printf( "%c" , ptr1[i] );
    for( int i=0 ; i<5 ; i++ ) printf( "%c" , ptr2[i] );
    printf( "\n" );
    return 0;
}
```
Why?

When `swap` is called, it is called with copies of the pointers `ptr1` and `ptr2`.

⇒ Swapping the addresses stored in `list1` and `list2` happens with the stack frame of `swap` and is not visible when the function returns.
Why?

When `swap` is called, it is called with the addresses of the pointers `ptr1` and `ptr2`.

⇒ Swapping the contents of data pointed to by `list1` and `list2` changes what `ptr1` and `ptr2` point to.

⇒ The effects are visible even after the function returns.

```c
#include <stdio.h>

void swap( char ** list1 , char ** list2 )
{
    char * temp = *list1;
    *list1 = *list2;
    *list2 = temp;
}

int main( void )
{
    char ar1[] = { 'a', 'b', 'c', 'd', 'e' };
    char ar2[] = { 'f', 'g', 'h', 'i', 'j' };
    char *ptr1 = ar1, *ptr2 = ar2;
    swap( &ptr1 , &ptr2);
    for( int i=0 ; i<5 ; i++ ) printf( "%c " , ptr1[i] );
    for( int i=0 ; i<5 ; i++ ) printf( "%c " , ptr2[i] );
    printf( "\n" );
    return 0;
}
```

```
>> ./a.out
fg h i j a b c d e
>>
```
Why?

When swap is called, it is called with the addresses of the pointers ptr1 and ptr2

⇒ Swapping the contents of data pointed to by list1 and list2 changes what ptr1 and ptr2 point to

⇒ The effects are visible even after the function returns

```c
#include <stdio.h>

void swap( char ** list1 , char ** list2 )
{
    char * temp = *list1;
    *list1 = *list2;
    *list2 = temp;
}

int main( void )
{
    char ar1[] = { 'a' , 'b' , 'c' , 'd' , 'e' };
    char ar2[] = { 'f' , 'g' , 'h' , 'i' , 'j' };
    char *ptr1 = ar1 , *ptr2 = ar2;
    swap( &ptr1 , &ptr2);
    for( int i=0 ; i<5 ; i++ ) printf( "%c " , ptr1[i] );
    for( int i=0 ; i<5 ; i++ ) printf( "%c " , ptr2[i] );
    printf( "\n" );
    return 0;
}
```
Why?

When `swap` is called, it is called with the addresses of the pointers `ptr1` and `ptr2`.

⇒ Swapping the contents of data pointed to by `list1` and `list2` changes what `ptr1` and `ptr2` point to.

⇒ The effects are visible even after the function returns.

```c
#include <stdio.h>

void swap( char ** list1 , char ** list2 )
{
    char * temp = *list1;
    *list1 = *list2;
    *list2 = temp;
}

int main( void )
{
    char ar1[] = { 'a' , 'b' , 'c' , 'd' , 'e' };
    char ar2[] = { 'f' , 'g' , 'h' , 'i' , 'j' };
    char *ptr1 = ar1 , *ptr2 = ar2;
    swap( &ptr1 , &ptr2);
    for( int i=0 ; i<5 ; i++ ) printf( "%c " , ptr1[i] );
    for( int i=0 ; i<5 ; i++ ) printf( "%c " , ptr2[i] );
    printf( "\n" );
    return 0;
}
```
Why?

When `swap` is called, it is called with the addresses of the pointers `ptr1` and `ptr2`.

⇒ Swapping the contents of data pointed to by `list1` and `list2` changes what `ptr1` and `ptr2` point to.

⇒ The effects are visible even after the function returns.

```c
#include <stdio.h>
void swap( char ** list1 , char ** list2 )
{
    char * temp = *list1;
    *list1 = *list2;
    *list2 = temp;
}
int main( void )
{
    char ar1[] = { 'a' , 'b' , 'c' , 'd' , 'e' };
    char ar2[] = { 'f' , 'g' , 'h' , 'i' , 'j' };
    char *ptr1 = ar1 , *ptr2 = ar2;
    swap( &ptr1 , &ptr2);
    for( int i=0 ; i<5 ; i++ ) printf( "%c " , ptr1[i] );
    for( int i=0 ; i<5 ; i++ ) printf( "%c " , ptr2[i] );
    printf( "\n" );
    return 0;
}
```
Why?

When `swap` is called, it is called with the addresses of the pointers `ptr1` and `ptr2`.

⇒ Swapping the contents of data pointed to by `list1` and `list2` changes what `ptr1` and `ptr2` point to.

⇒ The effects are visible even after the function returns.

```c
#include <stdio.h>

void swap( char ** list1 , char ** list2 )
{
    char * temp = *list1;
    *list1 = *list2;
    *list2 = temp;
}

int main( void )
{
    char ar1[] = { 'a' , 'b' , 'c' , 'd' , 'e' };
    char ar2[] = { 'f' , 'g' , 'h' , 'i' , 'j' };
    char *ptr1 = ar1 , *ptr2 = ar2;
    swap( &ptr1 , &ptr2);
    for( int i=0 ; i<5 ; i++ ) printf( "%c " , ptr1[i] );
    for( int i=0 ; i<5 ; i++ ) printf( "%c " , ptr2[i] );
    printf( "\n" );
    return 0;
}
```
Piazza → Resources section → Resources tab → Exercise 5-1