Phys4051: C Lectures

Pointers and Address Operators
Pointer to a Variable
Function Calls: Passing by Value and Passing by Reference
Pointer to an Array

Pointers and Variables: Definition

Variable:
A variable refers to a memory location that contains a numerical value.

Pointer
A pointer refers to a memory location that contains an address.

Pointers: Operators (1)

Address Operator: &
Note: it looks identical to the bitwise AND operator but it is used in a completely different way!
Returns the address of a variable
Example: ptr_v = & x;

Pointers: Operators (2)

Indirection Operator: *
Note: it looks identical to the multiplication operator but it is used in a completely different way!
Retrieves a value from the memory location the pointer points to.
Example: *ptr_v = 77;

Pointer Declaration

A pointer must be declared and the variable type it points to must be specified:

short *aptr;  //pointer declaration
double *bptr;
float* fptr; //same as float *fptr

Assigning an Address to a Pointer (1)

An address is assigned to a pointer using the address operator: &
Assigning an Address to a Pointer (2)

Example:
short x = 33;
short *aptr; //pointer declaration
aptr = & x;

(The memory addresses are arbitrarily chosen)


Example:
short x = 33;
short *aptr; //declare the ptr
aptr = & x; //ptr points to x
*aptr = -123; //assign a value to “x”

(The memory addresses are arbitrarily chosen)

Pointer Usage with a Variable:

The following two segments are equivalent in respect what they do to variable “x”:

short x = 33;
aptr = & x;
*aptr = -123;
short x = 33;
aaptr = & x;
*xaptr = -123;
short x = 33;
x = -123;

Pointers: Additional Comment

Pointers refer to an address which is almost always the address of another variable.
An (arbitrary) address can be directly assigned to a pointer. Doing so makes the program less portable and can be very dangerous!
Example:
short *aptr = 0x300;
*aptr = 0xff;

Pointers: Function Calls and Function Arguments

Variables can be passed to a function (as function arguments) either:

a) by value
   (as a copy of a local variable)
b) by reference
   (by a pointer)

Function Arguments: Passing by Value

This is the method you have used so far in these examples.
A (local) copy of the variable is passed to function.
Changing the (passed) variable within the calling function has no effect on the (original) variable that was passed.
Ex. 1: Exchange Two Variables: By Value

Problem:

a) Write a function “Xchange” that will exchange two variables if the “first” variable is greater than the “second” one.

b) You are not allowed to use GLOBAL variables!

Ex. 1a: Exchange Two Variables (2): main

```c
void Xchange_ByVal(short u, short v);
main()
{
    short x = 10, y = 2;
    Xchange_ByVal(x, y);
    printf("x: %d  ", x);
    printf("y: %d\n", y);
}
```

Ex. 1a: Exchange Two Variables (3): Function

```c
void Xchange_ByVal(short u, short v)
{
    short stemp;
    if( u > v){
        stemp = v; v = u; u = stemp;
    }
    printf("u: %d  ", u);
    printf("v: %d\n", v);
}
```

Ex. 1a: Exchange Two Variables (4): Output

```
Output:

u: v:

Conclusion: Works? (y/n)

Why (not)?
```

Function Arguments: Passing by Reference

Allows you to change the value of a variable which is not local to the function without having to make it global

Pass a reference (a pointer) to the function which tells the function where “to find” that variable

Note: (usually) you don’t change the reference, you change only what the reference points to!

Ex. 1b: Exchange Two Variables (5): By Reference

Solution:

Pass the function arguments by reference!
Ex. 1b: Exchange Two Variables (6): main

```c
void Xchange_ByRef(short *u,
                    short *v);
main()
{
    short x = 10, y = 2;
    Xchange_ByRef(&x, &y);
    printf("x: %d  ", x);
    printf("y: %d\n", y);
}
```

Ex. 1b: Exchange Two Variables (7): Function

```c
void Xchange_ByRef(short *u,
                    short *v)
{
    short stemp;
    if( *u > *v){
        stemp = *v; *v = *u;
        *u = stemp;
    }
    printf("u: %d  ", *u);
    printf("v: %d\n", *v);
}
```

Ex. 1b: Exchange Two Variables (8): Output

*Output:

u: x:

v: y:

*Conclusion: Works? (y/n)

Summary: Function Calls a) by Value

*Passing a Variable by Value:

```c
void FbyVal( int );
main()
{
    int y = 3;
    FbyVal(      );
}
```

*Passing a Pointer by Value:

```c
void FbyVal(int );
main()
{
    int x = 3;
    int* y = &x;
    FbyRef(      );
    FbyVal(      );
}
```

Summary: Function Calls a) by Reference

*Passing a Variable by Reference:

```c
void FbyRef( int* );
main()
{
    int y = 3;
    FbyRef(      );
}
```

*Passing a Pointer by Reference:

```c
void FbyRef(int* );
main()
{
    int x = 3;
    int* y = &x;
    FbyRef(      );
}
```
Arrays and Pointers

Pointers are most often used in function calls and with arrays.

Because pointers are so often used with arrays, a special pointer has been designated in C to point to the “zeroth” element in an array: **the array name itself**!

**Example 2a:**
```c
float w[128];
float *w_ptr;
w_ptr = &w[0];
```

**Example 2b:**
```c
float w[128];
float *w_ptr;
w_ptr = w;
```

**Pointer to the “Zeroth” Array Element: Summary**

Each time you declare an array, you also declare implicitly a pointer to the “zeroth” element!

The name of this pointer is the name of the array!

**Arrays: Memory Allocation**

```c
short x = 2;
short y[4];
y[x] = 12345;
*y = 5121;  //where does y point to?
```

(The memory addresses are arbitrarily chosen)

**Arrays: Memory Allocation: Pointer Math (1)**

```c
short x = 2;
short y[4];
y[x] = 12345;
*y = 5121;  //same as: y[0] = 5121;
*(y+1) = 5122;
```

(The memory addresses are arbitrarily chosen)

**Arrays: Memory Allocation: Pointer Math (2)**

```c
short x = 2;
short y[4];
*y = 5121;  //same as: y[0] = 5121;
*(y+1) = 5122;  //same as: y[1] = 5122;
*(y+x) = 5123;  //same as:
```

(The memory addresses are arbitrarily chosen)
The following two segments are equivalent statements:

**Segment 3a:**
```c
short x, y[ MAX], val;
y[x] = val;
```

**Segment 3b:**
```c
short x, y[ MAX], val;
*(y + x) = val;
```

---

A) Write a function that returns the average of an array of type double.
B) Write a program that uses above function.
C) Change the program above to account for the fact that we want to ignore the first 5 data points in the array when calculating the average.

---

A) A function that returns the average of an array of type double.
A) A program that uses above function.
A) Change the program above to account for the fact that we want to ignore the first 5 data points in the array when calculating the average.
Example 4: Function to Sort an Array (1): Problem

*Assignment:*

Write a function that sorts the values contained in an array.

---

Example 4: Function to Sort an Array (2): Solution

*Pass arrays whenever possible by reference! (Also, no need for global arrays!)*

*Passing an array by value takes a long time (and lots of space) because the computer has to make a copy of the array to pass it to the function.*

---

Ex. 4: Function to Sort an Array (3): main

```c
#define MAX 10
void SortAr( short *volt, short n );
main(){
    short i, sAr[ MAX ];
    for( i = 0; i < MAX; i++ ){
        sAr[i] = rand();
        printf("%d %d\n", i, sAr[i] );
    }
    SortAr( sAr, MAX ); // pass by ref
    for( i = 0; i < MAX; i++ )
        printf("%d %d\n", i, sAr[i] );
}
```

---

Ex. 4: Function to Sort an Array (4): Sort Function V1

```c
void SortAr( short *volt, short n ){   
    short x, y, stemp;
    for( y = 0; y < n - 1; y++ ){    
        for( x = 0; x < n - 1 - y; x++ ){ 
            if( volt[ x ] > volt[ x + 1 ] ){    
                stemp = volt[x];
                volt[x] = volt[x+1];
                volt[x+1] = stemp;
            }
        }
    }
}
```

---

Ex. 4: Function to Sort an Array (5): Sort Function V2

```c
void SortAr( short *volt, short n ){   
    short x, y, stemp;
    for( y = 0; y < n - 1; y++ ){    
        for( x = 0; x < n - 1 - y; x++ ){ 
            if( *(volt+x) > *(volt+x+1) ){    
                stemp = *(volt+x);
                *(volt+x) = *(volt+x+1);
                *(volt+x+1) = stemp;
            }
        }
    }
}
```