Pointers as Function Arguments

Chapter 11

Objectives

You will be able to

- Write functions that take pointers as parameters.
- Write programs that call functions that take pointers as parameters.
- Write and call functions that modify the caller’s local variables.

Passing Pointers to Functions

- Functions can be defined with addresses as parameters.

```
void Get_Sides (int *width, int *length)
```

- Caller passes *addresses* of variables that the function can then access.
  - Address &value
  - Pointer pValue

Function with Address Parameters

```
void Get_Sides (int *width, int *length)
{
    do
    {
        *width = -1;
        printf ("Please enter Width as positive integer: ");
        scanf ("%d", width);
        /* Clear input buffer.  Only do this if user   */
        /* is expected to input only one value per line*/
        while (getchar() != '\n');
        if (*width <= 0)
        {
            printf ("Received invalid value %d\n", *width);
        }
    } while (*width <= 0);

    /* Same for length */
    /* ... */
}
```

pointee, or target of the pointer, which is in the caller's space

No & (width *is* an address)

Check value entered by user

No “return” statement. Values are put directly into variables whose addresses are passed as arguments.

Passing Addresses as Arguments

```
int main (void)
{
    int width = 0;
    int length = 0;
    int area = 0;
    printf ("This program computes the area of a rectangle\n");

    Get_Sides (&width, &length);

    area = width * length;
    printf ("Area is %d\n", area);
    return 0;
}
```

Must pass in the location of the arguments,i.e., their addresses

Program Running

```
turnerr@login2:~/test
[turnerr@login2 test]$
[turnerr@login2 test]$
[turnerr@login2 test]$ gcc -Wall ptrargs.c
[turnerr@login2 test]$ ./a.out
This program computes the area of a rectangle
Please enter Width as positive integer: 2
Please enter Length as positive integer: 3
Area is 6
[turnerr@login2 test]$
```

Program Running

```
turnerr@login2:~/test
[turnerr@login2 test]$
[turnerr@login2 test]$
[turnerr@login2 test]$ ./a.out
This program computes the area of a rectangle
Please enter Width as positive integer: 2.5
Please enter Length as positive integer: abc
Received invalid value -1
Please enter Length as positive integer: 4.6
Area is 8
[turnerr@login2 test]$
```

Passing Addresses as Arguments

- We could also have passed pointers as arguments to Get_Sides.

Passing Pointers as Arguments

```
int main (void)
{
    int width = 0;
    int length = 0;
    int *pWidth = &width;
    int *pLength = &length;
    int area;
    printf ("This program computes the area of a rectangle\n");

    Get_Sides (pWidth, pLength);
    area = width * length;
    printf ("Area is %d\n", area);
    return 0;
}
```

No & this time!

Effect is identical to previous example, but
better to use previous approach

A Good Test Question

```
int main (void)
{
    int *pWidth;
    int *pLength;
    int area;
    printf ("This program computes the area of a rectangle\n");

    Get_Sides (pWidth, pLength);

    area = *pWidth * *pLength;

    printf ("Area is %d\n", area);
    return 0;
}
```

What’s wrong with this program?

Try It!

```
turnerr@login2:~/test
[turnerr@login2 test]$
[turnerr@login2 test]$
[turnerr@login2 test]$ gcc -Wall ptrargs.c
[turnerr@login2 test]$ ./a.out
This program computes the area of a rectangle
Segmentation fault (core dumped)
[turnerr@login2 test]$
```

Compiles without

Runtime error (Good!)

A Classic Example

- Function to swap two variables.
- Can’t do this without pointers.

swap.c

Caller passes addresses of two doubles.

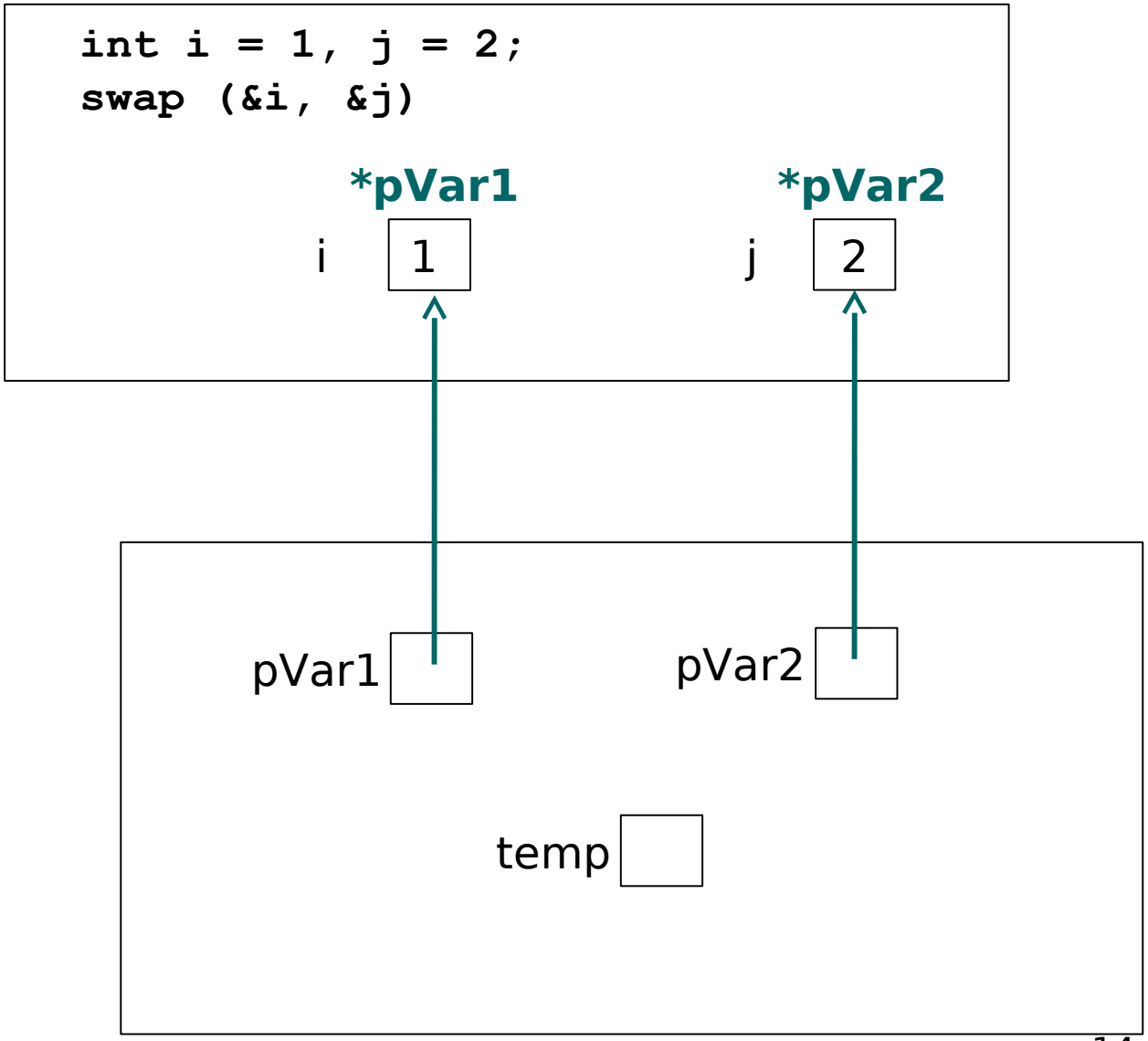
```
#include <stdio.h>

void swap (double *pVar1, double *pVar2)
{
    double temp;

    temp = *pVar1;
    *pVar1 = *pVar2;
    *pVar2 = temp;
}
```

swap.c

```
int i = 1, j = 2;
swap (&i, &j)
```



swap code:

```
pVar1 = &i;
pVar2 = &j;

temp = *pVar1;
*pVar1 = *pVar2;
*pVar2 = temp;
```

swap.c

```
int main (void)
{
    double var1 = 12.5;
    double var2 = -10.1;

    printf ("Before swap: var1 = %5.1f  var2 = %5.1f\n",
        var1, var2);

    swap (&var1, &var2);

    printf ("After swap:  var1 = %5.1f  var2 = %5.1f\n",
        var1, var2);

    return 0;
}
```

Run swap.c

turnerr@login2:~/test

```
int main( void )
{
    double var1 = 12.5;
    double var2 = -10.1;

    printf ("Before swap: var1 = %5.1f  var2 = %5.1f\n",
       var1, var2);

    swap (&var1, &var2);

    printf ("After swap: var1 = %5.1f  var2 = %5.1f\n",
       var1, var2);
    return 0;
}

[turnerr@login2 test]$ gcc -Wall sort.c
[turnerr@login2 test]$ ./a.out
Before swap: var1 =  12.5  var2 = -10.1
After swap: var1 = -10.1  var2 =  12.5
[turnerr@login2 test]$
```

A Very Simple Sort

- A very inefficient, but very simple, sorting algorithm.
  - Wouldn’t use to sort a large array.
  - Efficiency doesn’t matter for short arrays.

Given an array of numbers,

- Compare each entry after the first to the preceding entry.
  - If the preceding entry is larger, exchange them.
- Repeat the above until no exchange is done.

The swap() Function

```
void swap (int* n1, int* n2)
{
    int temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
```

The sort() function

```
void sort (int numbers[], int length)
{
    int exchange_done;
    int i;

    do
    {
        exchange_done = 0;
        for (i = 1; i < length; i++)
        {
            if (numbers[i] < numbers[i-1])
            {
                swap(&numbers[i], &numbers[i-1]);
                exchange_done = 1;
            }
        }
    } while (exchange_done);
}
```

Function get_input()

```
int get_input(int numbers[], int length)
{
    int i;
    printf ("Please enter up to %d positive integers\n", length);
    printf ("Enter 0 to terminate input before %d entries\n", length);
    for (i = 0; i < length; i++)
    {
        do
        {
            printf ("%d: ", i);
            numbers[i] = -1;
            scanf("%d", &numbers[i]);
            if (numbers[i] < 0)
            {
                printf ("Please enter positive values only\n");
                while (getchar() != '\n');  // Clear input buffer
            }
        } while (numbers[i] < 0);

        if (numbers[i] == 0)
        {
            break;
        }
    }
    /* i is number of entries filled. */
    return i;
}
```

Function output_array()

```
/* Output to the console an array of integers, where caller
 * passes the array and the number of entries in the array. */
void output_array (int n[], int length)
{
    int i;
    for (i = 0; i < length; i++)
    {
        printf ("%2d: %8d\n", i, n[i]);
    }
}
```

Function main()

```
int main (void)
{
    int numbers[10] = {};
    int length;
    printf ("This program sorts an array of positive integers.\n");

    length = get_input(numbers, sizeof(numbers)/sizeof(numbers[0]) );

    if (length == 0)
    {
        printf ("No numbers were entered\n");
        return 1;
    }

    printf ("Unsorted values:\n");
    output_array(numbers, length);

    sort(numbers, length);

    printf ("\nSorted values:\n");
    output_array(numbers, length);
    return 0;
}
```

Sort Program Running

```
turnerr@login2:~/test
[turnerr@login2 test]$ gcc -Wall sort.c
[turnerr@login2 test]$ ./a.out
This program sorts an array of positive integers.
Please enter up to 10 positive integers
Enter 0 to terminate input before 10 entries
0: 5
1: 2
2: 12
3: 3
4: 2
5: 0
Unsorted values:
 0:          5
 1:          2
 2:         12
 3:          3
 4:          2

Sorted values:
 0:          2
 1:          2
 2:          3
 3:          5
 4:         12
[turnerr@login2 test]$
```

Summary

- Pointers hold addresses.
  - Specific to a type.
- Use & to get the address of a variable.
  - Only makes sense on RHS of =
  - You can’t *set* the address of something.
- Use * to refer to the contents of the address held by a pointer.
  - Either side of =

Summary

- You must be sure a pointer holds a valid address before using it.
  - Doesn’t happen automatically!

- A function can have pointer parameters.
  - Permits the function to modify caller’s variables.
  - Caller passes addresses of own variables in order to let the function modify them.

Assignment

- Read Chapter 11
- Do the examples from this presentation for yourself.

End of Presentation

Exercise

Exercise 11.5 Write the following function:

```
int *find_largest(int a[], int n)
```

When passes an array of length n, returns the **address** of the largest element of a.

```
int *find_largest(int a[], int n)
{
   int i = 0;
   int max = 0;
   for(i=1;i<n;i++) {
       if (a[i] > a[max])
       {
          max = i;
       }
   }
   return &a[max]
}
```