Pointers and Functions with Passing by Pointers

Presenter: Alan Manuel Loreto Cornídez | MS in ECE Spring ’23 Class: ECE 175 | Computer Programming For Engineering Applications

Lecture Overview

- Limitations of reference by value
- The concept of Pointers
- Pointers in C
- Functions and Pointers

Limitations of Reference by Value

Example

Suppose we have a function `circle` that we want to use to calculate the area and circumference of a circle of radius, `r`.

```c
void circle(float r, float a, float c)
{
  a = PI * r * r; // computation of area
  c = 2 * PI * r; // computation of circumference
}
```

What happens if we call our circle function in our main function?

```c
int main(int argc, char **argv)
{
  // Create our variables.
  float r;
  float a = 0;
  float c = 0;

  // User enters the radius of the circle.
  printf("Enter the radius of the circle: ");
  scanf("%f", &r);

  // Call the circle function to calcualte the area and circumfrence.
  circle(r, a, c);

  // Print the area and circumfrence of the circle.
  printf("Area: %.1f, Circumference: %.1f\n", a, c);

  return 0;
}
```

Show `sample-01.c`

The Concept of Pointers

What is a Pointer?

Pointer: a variable containing the address of another variable.

A pointer can also be defined as such: A label used to refer to (point to) a memory address.

In other words, a pointer variable just holds a number value. This value just so happens to correspond to the address of other variables.

Examples of Pointers in Real Life:

1. **Page Numbers** in a book or power point slides.
   - Page numbers are pointers to the contents of a particular page.
2. **Mailing Addresses** and Mailbox Numbers
   - The contents (letters) of a particular mailbox can be retrieved by referring to the mail box number (P.O. Box) when you're at the post office

Pointers in C

In programming contexts, a pointer holds a numerical value.

![[Pasted image 20230919185216.png]]

Why do we need pointers?

- **Faster memory access and usage via memory referencing**
  - Memory access speed was critical when C was developed (This is still the case for high performance applications)
- **Saving memory space**
  - No need to hold local copies of variables inside functions
  - Still required for programming resource-constrained devices such as micro-controllers (`Arduinos` use micro-controllers)
  - Can pass values to a function by pointer
- **Downsides**
  - Pointers usually make program tracking/reading more complicated.

Syntax Definitions

Declaration of Pointers

`{data type} *{name of pointer};`

Examples:

```
FILE *input;
int *myIntPointer;
float *myFloatPointer;
```

`*` is used to denote that a variable is a pointer.

Why do we assign a data type to a pointer? Again, pointers are variables that hold integer values (memory addresses). So why do we need to assign them a memory address?

Answer: Pointers let you access the contents of the address that the pointer is holding. (Different data types are use different amounts of bits, so the compiler needs the data type to make sure it manipulates the correct number of bits). - `char`: 8 bits - `short`: 16 bits - `int`: 32 bits - `double`: 64 bits

More generally, the compiler uses the data type of the pointer to manipulate numbers correctly.

Declaration of Addresses:

Use an `&` operator to get the address of a variable.

Example:

```c
// Declare my pointer variable.
int *myIntPointer;

// Declare my integer variable.
int myInt;

// Set my pointer variable to the address of my integer variable.
myIntPointer = &myInt;
```

Access The Content of a Variable Pointed to by a Pointer

Use the indirection or deference operator `*`.

Example:

```c
// declare integer pointers
int *p, i, var1;

// p points to i
p = &i;

// the content of memory cell pointed by p is assigned the value 10
*p = 10;

// variable var1 is assigned the contents of memory cell pointed by p
var1 = *p;
```

What happens if we run this code?

Example 2 Show `sample-03.c`

Again, the pointer type and the value or variable must match

You have already used pointers!

When you're reading files:

```c
// declaration of a pointer of type FILE
FILE *inp;

// inp keeps the address and other information where the text file is located
inp = fopen("filename", "r");

// A variable to store file input.
int x;

// Using the file pointer to store the value at the location (address) of x.
fscanf(inp, "%d", &x);
```

To scan for keyboard input

```c
//integer variable i is declared. The memory space is allocated for i
int i;
```

```
// scanf receives as input the address of variable i. when it gets value entered by a user it stores th
scanf("%d", &i);
```

Common Errors When Using Pointers

Forgetting to initialize the pointer

```
int main(void)
{
    int *p;

    // Assigning the value at address p before the address has been assigned. This will result in an er
    *p = 10;

    return(0);
}
```

Having a mismatch between variable types and pointer types

```
int main(void)
{
    float *p;

    int i = 18;

    p = &i;

    *p = 10.5;

    return(0);

}
```

Exercises

Exercise 1 | An Example Using Pointers

```
#include<stdio.h>

int main(void)
{
    int var1 = 123;
    int var2 = 456;
    int *p = NULL;

    p= &var1;

    *p=10;
    var2 = *p+10;

    printf("%d %d %d\n", var1, var2, *p);
    printf("%p %p %p\n", &var1, &var2, p);

    return (0);
}
```

Example Show `exercise-01.c`

What gets printed if we run this example? Assuming that we know the address of each variable given below.

Note: in practice, the memory address of a variable is assigned by the operating system (OS).

The address of `var1` is `0x0110FEC4` and the address of `var2` is `0x0110FEC8`.

The picture below shows what happens at the variable declaration (first 3 lines of the code).

![[Pasted image 20230919195945.png]]

Activity 1 | What gets printed?

```
#include<stdio.h>

int main(void)
{
    int *ptr, i = 10, j = 20, k =30;

    ptr= &i;

    *ptr = 40;

    ptr=&j;

    // *ptr = *ptr + I;
    *ptr += i;

    ptr = &k;

    *ptr = *ptr +k + j;

    printf("i = %d, j = %d, k = %d", i, j, *ptr);

    return (0);
}
```

Example

Exercise 2 | What Gets Printed?

```
#include<stdio.h>

void fun1(int x, int *y)
{
    x = x+(*y);
    *y = 5*(*y);
    x = (*y)*x;
}

int main(void)
{
    int x = 2, y = 8;
    fun1(x, &y);
    printf("%d\t%d\n", x, y);
    return (0);
}
```

Example

Activity 2 | Function Call using Pointers

1. Write a function sum_avg that has three input parameters with data type of double and two output parameters which are the sum and average of the three input parameters
2. In the main function, complete the code at the line with `//call a function here`

```c
void sum_avg()
{

}

int main(void)
{
    double z1, z2, z3, sumz, avgz;

    printf("Enter 3 numbers: ");

    scanf("%lf%lf%lf", &z1, &z2, &z3);

    //call a function here

    printf("sum and average are %.2lf,%.2lf", sumz, avgz);

    return 0;
}
```

Create a Common Fraction Calculator

![[Pasted image 20230919205221.png]]

![[Pasted image 20230919205250.png]]

Sample Executions

```
Enter the numerator and the denominator of the fraction: 1 8
Enter the operator: +
Enter the numerator and the denominator of the fraction: 3 8
1/8 + 3/8 = 1/2
```

```
Enter the numerator and the denominator of the fraction: 10 6
Enter the operator: -
Enter the numerator and the denominator of the fraction: 6 6
10/6 - 6/6 = 2/3
```

```
Enter the numerator and the denominator of the fraction: 1 8
Enter the operator: /
Enter the numerator and the denominator of the fraction: 2 3
1/8 / 2/3 = 3/16
```

Program Structure

![[Pasted image 20230919205456.png]]

Declaration of Functions

```c
void read_fraction(int *p_num, int *p_den);

int gcd(int x,int y);

void add_fractions(int n1, int d1, int n2, int d2, int *p_n, int *p_d);

void multiply_fractions(int n1, int d1, int n2, int d2, int *p_n, int *p_d);

void reduce_fraction(int *p_n, int *p_d);

void print_fractions(int n, int d);
```

Program Structure (Including Function Calls in Main)

![[Pasted image 20230919205657.png]]

gcd and print_fractions Functions

```c
int gcd(int x, int y)
{
    int rem = 1, gcd;
    while (rem!=0)
    {
        if (y == 0) {
            gcd = x;
            rem = 0;
        }

        else{
            gcd = y;
            rem = x%y;
            x = y;
            y = rem;
        }
    }
    return gcd;
}
```

```c
void print_fractions(int n, int d)
{
    if (n == 0)
        printf("0");
    else if (d == 0)
        printf("NaN");
    else if (d == 1)
        printf("%d", n);
    else
        printf("%d/%d",n, d);
}
```

Complete the following functions

```c
void read_fraction(int *p_num, int *p_den) {
    printf("Enter the numerator and the denominator of the fraction:");
    scanf("%d%d", /* complete me */ );
}

//Addition:
void add_fractions(int nl, int dl, int n2, int d2, int *p_n, int *p_d) {
    //write your code here
}
//Multiplication:
void multiply_fractions(int nl, int dl, int n2, int d2, int *p_n, int *p_d) {
    //write your code here
}
//Reduction: n = n /gcd(n, d), d = d/gcd(n, d).
// Recall, the function protptype of gcd is int gcd(int x, int y)
void reduce_fraction(int *p_n, int *p_d) {
    //write your code here
}
```

Write the Main Function

```
#include <stdio.h>

int main(void)
{
    int n1, d1, n2,d2, n, d;

    char op;

    read_fraction(/* complete me */ );

    printf("Enter the operator:");

    scanf("%*c%c", &op);

    read_fraction(/* complete me */ );

    switch(op){

        case '+': // call a function here

        break;

        case '-': // call a function here

        break;

        case '*': // call a function here

        break;

        case '/': // call a function here

        break;

        default:
```

```
        printf("This is not a valid operator");
    }

    //call a function here to make the answer into a reduced form

    // write your code here to complete the main function (call print_fractions as needed)

    return 0;

}
```

Quick Final Demo

Here is a quick demo to show how a pointer is used in a basic way.

```
int main()
{
    // Create a pointer.
    int* myPointer;

    // Initialize my number.
    int myNumber = 39;

    // Set my pointer equal to the address of myNumber.
    myPointer = &myNumber;

    // Print Statements
    printf("My Number: %d\n", myNumber);
    printf("Address of myNumber: %d\n", myPointer);
    printf("Using a pointer to print the value of myNumber: %d", *myPointer);

    return 0;
}
```

Note: All of the code is provided on the following github repository:

https://github.com/aloretocornidez/175-computer-programming/

You can also follow this QR code:

![[file.png]]