Lectures/Week_14/1_compare_sort.c

```c
#include <stdio.h>

int main()
{
  int arr[10] = {23, 45, 12, 56, 9, 34, 67, 89, 2, 15}; // Example array
  // Assume the first element is the minimum
  int min = arr[0];

  // Loop through the array to find the minimum
  for (int i = 1; i < 10; i++)
  {
    if (arr[i] < min)
    {
      min = arr[i]; // Update min if a smaller value is found
    }
  }

  // use a while loop to find the maximum
  int max = arr[0];
  int index = 0;
  while (index < 10)
  {
    if (arr[index] > max)
    {
      max = arr[index];
    }
    index++;
  }
  printf("The max value in the array is: %d\n", max);

  // Bubble sort implementation
  for (int j = 0; j < 10; j++) // Outer loop for passes, we need to have 10 passed, or j<=9
  {
    for (int i = 0; i < 9; i++) // Inner loop for comparisons, we need to have 9 comparisons in each pass, more efficiently i<10-j-1
    {
      if (arr[i] > arr[i + 1]) // Swap if the current element is greater than the next
      {
        int temp = arr[i]; // Reserve a temporary variable
        arr[i] = arr[i + 1];
        arr[i + 1] = temp;
      }
    }
  }

  // Structure of swap two elements using a, b, and temp
  // int temp = a;
  // a = b;
  // b = temp;

  // Print the sorted array
  printf("The sorted array is: ");
  for (int j = 0; j < 10; j++)
  {
    printf("%d ", arr[j]);
  }
  printf("\n");
}
```

Lectures/Week_14/2_func.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// Function prototypes
void func1(void); // Function with no parameters and no return value
void func2(int, int);// Function with two parameters and no return value
double func3(void);// Function with no parameters and a return value
double func4(int, double, double); // Function with three parameters and a return value
void integrity(void);
void printname(char, char);
int cube_add(int, int);

int main()
{
  // Call func1
  func1();

  // call a function to print an integrity statement
  integrity();

  // call a function to print your two initials
  char op = 'L', op2 = 'A';
  printname(op, op2);

  // Call func2 with two arguments
  func2(5, 10);

  // or you can do this
  // int x=5, y=10;
  // func2(x, y);

  // Call func3 and store its return value
  double result1 = func3();
  printf("Result from func3: %.2f\n", result1);

  // Call func4 with two arguments and store its return value
  // When you call it, make sure (1) order of parameters is correct
  // and (2) types of parameters are correct
  double a = 5.5, b = 10.5;
  double result2 = func4(10, a, b);
  printf("Result from func4: %.2f\n", result2); // it would be just (10+a-b)*1.5



  int result3;
  int x = 4, y = 6;
  result3 = cube_add(x, y); // Call the function and store the result

  // Variables passed to/from the function and main do not need to have the same variable names
  // the program will look at the order of the variables, not the names

  return 0;
}

//-----------------------------------------------------------
// Below are function definitions
//-----------------------------------------------------------

void integrity(void)
{
  printf("I, have completed this assignment with integrity\n");
}

void printname(char op, char op2)
{
  printf("My name is %c.%c.\n", op, op2);
}

int cube_add(int a, int b)
{
  int result;
  result = pow(a, 3) + pow(b, 3);
  return result;
  // You must have a return statement in a function that has a return type other than void
  // The return statement must match the return type of the function
}

// 1. Function with no parameters and no return value
void func1(void)
{
  printf("This is func1: No parameters, no return value.\n");
}

// 2. Function with two parameters and no return value
void func2(int a, int b)
{
  printf("This is func2: Received parameters a = %d, b = %d, no return value.\n", a, b);
}

```

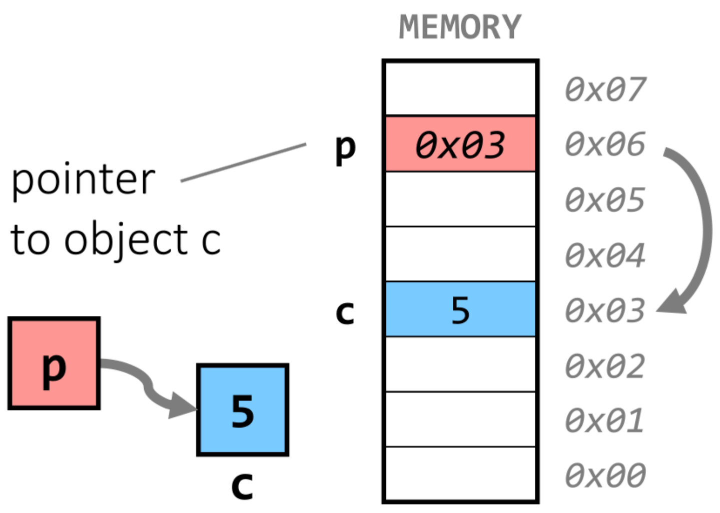
```c
// 3. Function with no parameters and a return value
double func3(void)
{
  printf("This is func3: No parameters, returns a value from scanf.\n");
  double a;
  printf("Enter a number: ");
  scanf("%lf", &a); // Example of using scanf
  return a;
}

// 4. Function with three parameters and a return value
double func4(int a, double b, double c)
{
  printf("This is func4: Received parameters a = %d, b = %lf. c=%lf, returns value (a+b-c)*1.5 to main.\n", a, b, c);
  return (a + b - c) * 1.5;
  // interpret the return statement as:
  // first parameter plus second parameter minus third parameter, then result times 1.5
  // name of these three parameters does not matter
}

// C programming with functions workflow
// 1. Function Declaration (Prototype)
// 2. Main (inside main, call the functions)
// 3. Function Definition

// What is a function prototype?
// A function prototype is a declaration of a function that specifies
// 1. the function's name,
// 2. return type,
// 3. parameters (if any) type

// It serves as a forward declaration,
// allowing the compiler to understand how to call the function before its definition appears in the code.
// Function prototypes are typically placed at the beginning of a source file.

// What is function "return"?
// A function "return" is a statement that specifies the value that a function will send back to the caller.
// It is used in functions that have a return type other than "void".
// Remember that our main function has a return type of int, so you saw "return 0;" at the end of main.
// Different than MATLAB, C functions can return only one variable at a time.
```

Lectures/Week_14/3_func_arr.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define ANNUAL_INTEREST 0.05
#define MONTHS 12

// Function prototypes
void func5(double, double, double, double, double[]);
void prod(double[][3], double[]);
// Function with array parameters

double amount(double);

int main()
{
  double a = 1.0, b = 2.0, c = 3.0, d = 4.0;
  double arr[4] = {0, 0, 0, 0}; // Declare an array to hold the results

  printf("org arr is %lf %lf %lf %lf\n", arr[0], arr[1], arr[2], arr[3]);
  // call function 5:
  func5(a, b, c, d, arr);
  printf("new arr is %lf %lf %lf %lf\n", arr[0], arr[1], arr[2], arr[3]);

  printf("global variable is %lf\n", ANNUAL_INTEREST);
  double investment = 1000.0;
  double total_amount = amount(investment);

  double bonus = 25.0; // local variable to main
  return 0;
}

// Function definitions
void func5(double a, double b, double c, double d, double arr[])
{
  // we'd like to return the array as follows:
  // First element to be maximum among a, b, c, d
  // Second element to be minimum among a, b, c, d
  // Third element to be the sum of a, b, c, d
  // Fourth element to be average of a, b, c, d

  // although we don't need to know ANNUAL_INTEREST in this function,
  // we can still access it since it's a global variable
  printf("global variable is %lf\n", ANNUAL_INTEREST);
  // but we don't have access to bonus, since it's a local variable to main
  // printf("local variable is %lf\n", bonus); // wrong

  double max = a;
  if (b > max)
  {
    max = b;
  }
  if (c > max)
  {
    max = c;
  }
  if (d > max)
  {
    max = d;
  }

  double min = a;
  if (b < min)
  {
    min = b;
  }
  if (c < min)
  {
    min = c;
  }
  if (d < min)
  {
    min = d;
  }

  double sum = a + b + c + d;
  double avg = sum / 4.0;
  arr[0] = max;
  arr[1] = min;
  arr[2] = sum;
  arr[3] = avg;
  // return arr[]; // wrong
  // return arr[4];   // wrong
  // return arr[0];   // wrong

  // NO NEED TO RETURN THE ARRAY, since arr[] is stored by address
  // and the main function can access it
}

void prod(double matrix[][3], double prd_arr[])
{
  // 3 4 5 ; prd_arr[0] = 1*3*4*5 = 60
  // 6 7 8 ; prd_arr[1] = 6*7*8 = 336
  // 1 2 3 ; prd_arr[2] = 1*2*3 = 6
  // we'd like to return the product of each row of a 3x3 matrix

  for (int i = 0; i < 3; i++) // row index 0,1,2
  {
    prd_arr[i] = 1;              // initialize the product for each row to 1
    for (int j = 0; j < 3; j++) // column index 0, 1, 2
    {
      prd_arr[i] *= matrix[i][j]; // multiply each element in the row
    }
  }
  // Working on array inside a function
  // no need to return the array, since it's passed by address

  // but you have to make sure to include the array as a parameter
}

double amount(double investment)
{
  // we'd like to print the total amount after each month
  // also return to main the total amount after 12 months
  // given an initial investment and an annual interest rate
  double monthly_interest = ANNUAL_INTEREST / MONTHS; // local variable to func

  // finish it together using a for-loop
  return 0; // placeholder now, need to return the total amount
}
```

Lectures/Week_14/4_pointer.c

```c
#include <stdio.h>

int main()
{
  int c = 5; // Declare an integer variable

  int *ptr_c; // Declare a pointer to an integer
  ptr_c = &c; // Assign the address of a to pointer ptr_c
  // this two lines can be combined as: int *ptr_c = &c;

  // Print the address of y using two methods
  // Note the placeholder %p
  printf("Address of c (using &c): %p\n", &c);
  printf("Address of c (using ptr_c): %p\n", ptr_c);

  // Print the address of the pointer itself
  // Skip if you don't understand
  printf("Address of ptr_c: %p\n", &ptr_c); // we don't want to confuse you

  // Print the value of a using two methods
  printf("Value of c (direct access): %d\n", c);
  printf("Value of c (using pointer): %d\n", *ptr_c); // note the %d here

  double b = 1.414;
  double *ptr_b = &b; // Declare a pointer to a double and assign the address of b
  printf("Address of b (using &b): %p\n", &b);
  printf("Value of b (using pointer): %lf\n", *ptr_b); // note the %f here

  // Just one more fancy thing about pointers
  // Skip if you don't understand
  *ptr_c = 100; // Change the value of a using the pointer
  printf("Value of c (after changing using pointer): %d\n", c);

  return 0;
}

/** Output of program

Address of c (using &c): 0x...03
Address of c (using ptr_c): 0x...03

Address of ptr_c: 0x...06

Value of c (direct access): 5
Value of c (using pointer): 5
```



```

Address of b (using &b): 0x...13
Value of b (using pointer): 1.414000

Value of c (after changing using pointer): 100


Key Notes:

(1) Pointer Declaration: Use * to declare a pointer (e.g., int *pointer;).

(2) Assigning an Address: Use & to get the address of a variable (e.g., pointer = &variable;).

(3) Dereferencing a Pointer: Use * to access the value stored at the address held by the pointer (e.g., *pointer).

(4) Printing Pointers: Use %p to print pointer addresses.

(5) Pointer Types: The type of the pointer must match the type of the variable it points to.
*/
```