Week 1, Sat.

Functions: A function is a group of related statements to complete a task.

```
return_type   func_name(parameterlist)
{
     . . .
     // body of func.
}
```

Ex#1,

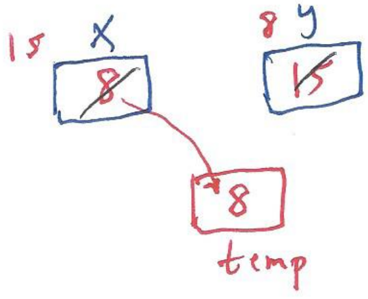
```
                       parameter list
int findMax(int num1, int num2) // func. Header
{
     if (num1 > num2)
             return num1;
     else
             return num2;
} //end findMax

int main()
{
   int x=10, y=20;
   intresult;              argument
   result = findMax( x , y ); // func call
   Cout<< "The Max value is: " << result<<endl;
   return 0;
} // end main
```

Function prototype: int findMax (int, int); // Should be above main func.

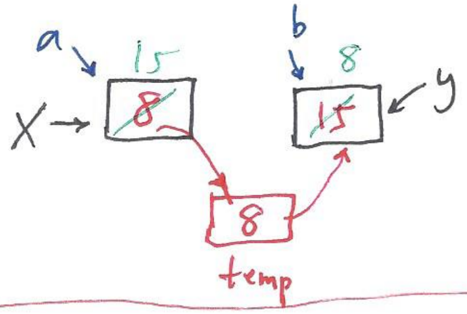
Ex#2,

```
#include<iostream>
using namespace std;
void swap(int, int); // func. prototype
int main()
{  int a=8, b=15;
   swap(a,b); // func. call, Call by value
   return 0;                   cout<<a <<b <<endl; //8,15
} //end main
void swap( int x , int y )
{   int temp;
    temp=x;
    x=y;
    y=temp;
    Cout<<x <<y<<endl; // x=15, y=8
} //end swap
```



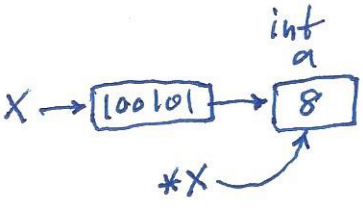
EX #3, Call by reference

```
    :        :
   void swap( int &x , int &y ); // func. prototype
    :             :           :
    swap(a, b); // func. Call
    Cout<<a <<b <<endl; // a=15, b=8
      :         :       :
    void swap ( int &x, int &y )
    { . . .
       . . .
       . . .
    } //end swap
```



EX #4, Call by pointer

```
      :           :          :         optional
     void swap ( int *x , int *y ); // func. prototype
      :            :          :
     swap( &a, &b ); // func. Call
       :               :
     void swap (int *x, int *y)
     {   int temp;
         temp = *x;
         *x = *y;
         *y = temp;
     } //endswap
```



ADT (Abstract Data Type): It is programmer's defined data type.

e.g. Class Person, class Rectangle, class shapes, ...

Data Type: Type of values can be stored and type of operations we are allowed.

e.g. We cannot divide or multiply two string values.

pointers: Each Variable is stored at a unique address in memory.

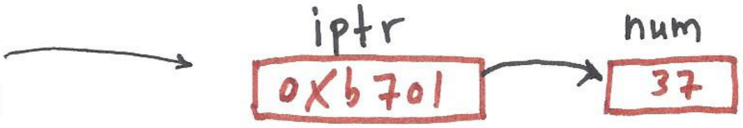
```
int num = 37;
                                      base16
cout << &num; // displays address (Hexadecimal, 0X4a02)
```

pointer variable: holds an address of memory location which it can also access the Content of the memory.
pointers are low level than arrays and reference variables.

```
int *iptr; // iptr can hold the address of the memory location which
           // we can store an int value.
```

```
int* iptr; ≡ int *iptr; ≡ int * iptr;
```

```
int num = 37;
int *iptr = &num;
```



```
int x = 25;
int *iptr;
iptr = &x;
cout << x << endl; // 25
cout << iptr << endl; // 0x7e05
```

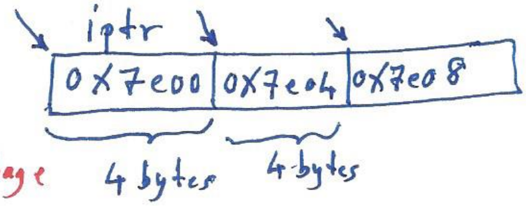
The indirection operator (*) dereferences pointer variable.

```
int x = 35;
int *iptr = &x;
cout << *iptr << endl; // 35
cout << x << endl; // 35
*iptr = 100;
cout << x << endl; // 100
```

```
Cout << *iptr <<endl; // 100
Cout << *iptr +1 <<endl; //101
Cout << *(iptr+1) <<endl; //Garbage
```

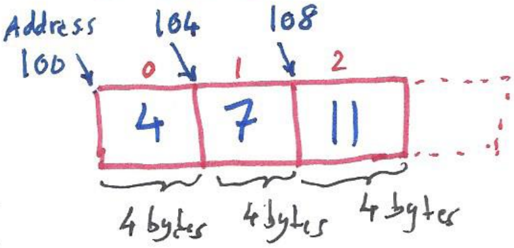
→ Size of int → 4bytes

→ Goes to the next memory location.



```
X= 700;
Cout << x <<endl; // 700
Cout << *iptr <<endl; //
```

Relationship between arrays and pointers :



```
int vals [] = { 4, 7, 11};
Cout << vals; // 100, display the address of
              // the first element of array
// Array name is starting address of the array
Cout << vals [0] ; // 4
cout << *vals << endl; // 4
int *valptr = vals;
Cout << valptr [1]; // 7
cout << *(valptr+1); // 7
Cout <<*(valptr+2); // 11
```

- We must use ( ) with * to get the values.

$$Vals[i] \equiv *(vals+i)$$

```
Cout << *(valptr+3); // Garbages , in C++ No bounds Checking
```

```
double Cost=15.75 ;
int *iptr = & Cost ; // We have different data types.
Cout << *iptr << endl; // Error
Cout << cost << endl; // Error
```

Solution ⟶ double *dptr = &Cost; // This works.

Dynamic Memory Allocation :

```
double *Sales ;
int numDays ;
Cout<< "Enter # of days : " <<endl;
Cin >> numDays ; // Size of the array
```

```
sales = new double [numDays]; //Dynamic Memory allocation
or // double *sales = new double [numDays];
```

OOP: We have a class and object. Please look at the Word lecture in Canvas.

template

Class is a blueprint for an object. An object is an instance of the class.

We tried three examples: 1- Class Person 2- Class BankAccount 3- Class Rectangle