# Data Structures CS-204

Lecture 3

# REVISION (Self Study)

Pointers
Function Templates
Class Templates

# POINTER OVERVIEW

- A pointer provides a way of accessing a variable (or a more complex kind of data, such as an array) without referring to the variable directly.
- The mechanism used for this is the address of the variable.
- In effect, the address acts as an intermediary between the variable and the program accessing it.

# POINTER OVERVIEW

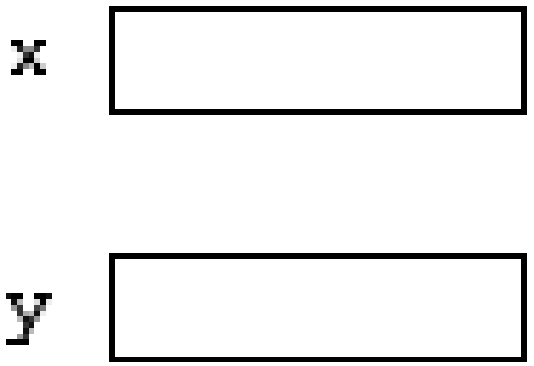
- In a similar way, a program statement can refer to a variable indirectly, using the address of the variable as a sort of post office box or hollow tree for the passing of information.
- Why are pointers used?
- Pointers are used in situations when passing actual values is difficult or undesirable.

# POINTER OVERVIEW

- Some reasons to use pointers are:
  - To return more than one value from a function.
  - To pass arrays and strings more conveniently from one function to another.
  - To manipulate arrays more easily by moving pointers to them (or to parts of them), instead of moving the arrays themselves.
  - To create complex data structures, such as linked lists and binary trees, where one data structure must contain references to other data structure.
  - To communicate information about memory, as in the function malloc( )/new, which returns the location of the free memory by using a pointer.

# Practicing Pointers

- Allocate two pointers x and y.
- Remember that allocating the pointers **does not** allocate any pointees.



# Practicing Pointers

- Allocate a pointee and set x to point to it.
- Memory is dynamically allocated for one pointee, and x is set to point to that pointee.

# Practicing Pointers

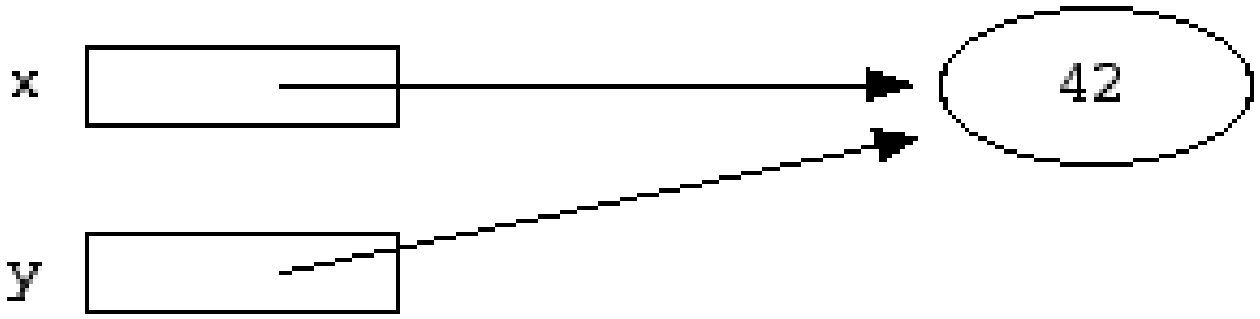
- Dereference x to store 42 in its pointee.

# Practicing Pointers

- Try to dereference y to store 13 in its pointee.
- This crashes because y does not have a pointee -- it was never assigned one.

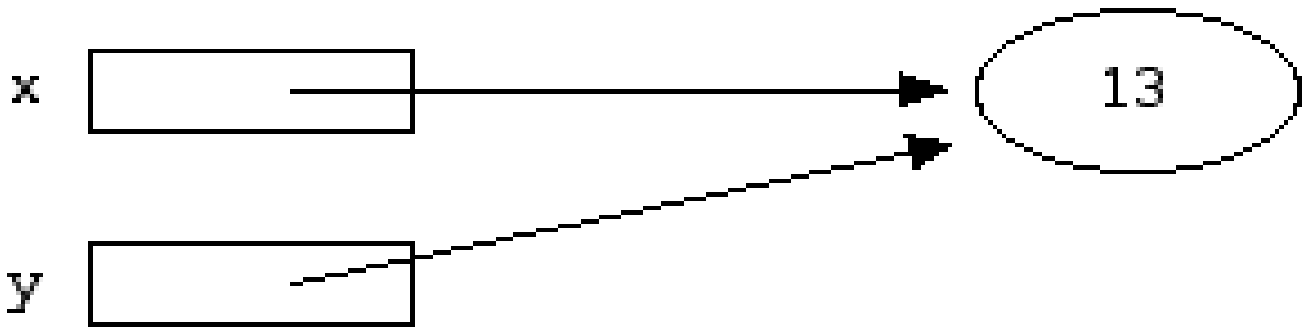
# Practicing Pointers

- Assign y = x; so that y points to x's pointee.
- Now x and y point to the same pointee -- they are "sharing".



# Practicing Pointers

- Try to dereference y to store 13 in its pointee.
- This time it works, because the previous assignment gave y a pointee.



# Pointer Variables

```
int* ptr;
```

- Asterisk means  pointer to
- Statement defines ptr as a pointer to int
- This variable can hold the address of integer variables

```
char* cptr;                 //pointer to char
int* iptr;                  //pointer to int
float* fptr;                //pointer to float
Distance* distptr;          //pointer to user-defined
                                Distance class
```

# Accessing the Contents

- If we know the address of a variable and don’t know its name. Can we access the contents of this variable?
- There is a special syntax to access the value/contents of a variable using its address.

# Accessing the Contents

```
int main()
{
int var1=11;
int* ptr;
ptr=&var1;
cout << *ptr << endl;//prints contents of pointer(?)
                     //* acts as indirection operator
}
```

# Declaration Vs Indirection

- Asterisk used as the indirection operator has a different meaning then the asterisk used to declare pointer variables.
- Indirection operator precedes the variable and means value of the variable pointed to by.
- The asterisk used in the declaration means pointer to

```
int* ptr;                    //declaration: pointer to int
*ptr = 37;          //indirection: value of variable
                              pointed to by ptr
```

# Pointers and Arrays

- There is a close association between pointers and arrays
- These two expressions have exactly the same effect:

```
intarray[j]
*(intarray+j)          //using indirection
  operator
```

# Pointers and Functions

- Passing arguments by pointer

```
#include <iostream.h>
void increment(int *);
 void main(void)
{
int tvar = 10;
increment(&tvar);
cout << tvar;
}
void increment(int *counter)
{
*counter = 11;//acts as tvar = 11
}
```

# Passing Arrays

```
const int MAX=5;
void main()
{
void centimize(double*);                    //prototype
double varray[MAX] = {10.0,43.1,95.9,59.7,87.3};
centimize(varray);
//since the name of an array is array's address, there is no need
    for &
…
}
Void centimize(double* ptrd)
{
for(int j=0; j<MAX;j++)
    *(ptrd+j) *= 2.54;                  //ptrd points to element of array
}
```

# const Modifier and Pointers

- There are two possibilities

```
const int* ptr;          // ptr is a pointer to constant int

int* const ptr;          // ptr is a constant pointer to int
```

- In first declaration, you can't change the value of whatever ptr points to, although you can change ptr itself.
- In second declaration, you can change what ptr points to, but you can't change the value of ptr itself.

# Memory Management

- If we use arrays for data storage, we must know at the time we write the program how big the array will be.

```
Cin>>size;                    //get size from user
Int arr[size];/               /error: array size must be a constant
```

- C++ provides a new approach to obtaining blocks of memory: the **new** operator.
- This versatile operator obtains memory from the operating system and returns a pointer to its starting point.

# The new Operator

```
#include<iostream.h>
#include<cstring>
int main()
{
char* str="self conquest is the greatest victory";
int len = strlen(str);
char* ptr;
ptr = new char[len+1];  //set aside memory: string + '\0'
strcpy(ptr, str);
cout << ptr << endl;
delete[] ptr;                          //release ptr's memory
return 0;
}
```

# Practicing Pointers in C++

- Allocate two pointers x and y.
- Allocate a pointee and set x to point to it.
- Dereference x to store 42 in its pointee.
- Try to dereference y to store 13 in its pointee.
- Assign y = x; so that y points to x's pointee.
- Try to dereference y to store 13 in its pointee.

```
int* x; int* y; // (but not the pointees)
x = new int;

*x = 42;

*y = 13;
        y = x;

*y = 13;
```

# Exercise for own learning

- At the end of the above code, y is set to have a pointee and then dereferenced it store the number 13 into its pointee. After this happens, what is the value of x's pointee?
- Write code segment in C++ for the following diagram.