**Lab 10/11**

**Pointers and Structures**

Recall the **Declaration** statement. The syntax is DataType Identifier.

What does the computer do when it see a declaration statement ?

Example: int x; // The type determines number of bytes alloated in memory (int has 4 bytes)

// The x variable name is an **alias** for the memory address of the first byte.

Recall that with a function there are two ways to pass a variable, ‘by value’ and ‘**by reference’**

What does the computer do when you pass ‘by value’ ?

Example: int age = 21;

Function header - void checkAge( int personsAge) { do something }

Call/Use the function - checkAge ( age)

The computer COPIES the VALUE from age to personsAge.

What does the computer do when you pass ‘**by reference’** ?

Recall that the & is called the Address operator.

It instructs the computer to pass ‘by reference’, to pass the address.

Example: int age = 21;

Function header - void checkAge( int & personsAge) { do something }

Call/Use the function - checkAge ( age)

The computer COPIES the ADDRESS used by the **age** variable,

and makes it the **same address** that the **personAge** variables points too.

When you change ‘personAge’, you are changing ‘age’,

since they both refer to the same address space in memory.

So, we have been dealing with **ADDRESSes** in Memory in several different ways.

***Pointers*** *deal with the addresses of declared objects in memory !*

*Pointers are declared variables, that that stores an address of another variable.*

*Symbols used: \* The asterisk*

*3 Basic uses of the asterisk: Declaration, and use in the left or right side of the assignment statement.*

***Declaration*** *- Declare a variable that contains an address that points to a variable of a given type.*

*Note: Variable that contain pointer commonly are named p or ptr followed by name.*

*Syntax : datatype \* Indentifier;*

*Example:*

*Int X = 3;*

*Int ptrInt = &X; // Places address of X into the pointer variable ‘ptrInt’.*

*// Note the type int for X, and the type int the address in ptrInt points too.*

***Use in Assignment Statement, Right side***

*Example:*

*int X = 3;*

*Int Y = 6;*

*Int Z;*

*int ptrX;*

*ptrX = &X; // address of X placed in ptrInt.*

*Z =* ***\*ptrX;*** *// value at address contained in ptrINT copied into Z.*

*// \*ptrInt – Use of the asterick is called ‘dereference’ or ‘indirection’*

*// Think of the \* cancelling out ptr, and all you are left with in a variable* = variable or

// or Z = X *which means copy the value in X to the variable Z.*

***Use in Assignment Statement, Left side***

*Example:*

*int X = 3;*

*Int Y = 6;*

*Int Z;*

*int ptrX;*

*ptrX = &X; // address of X placed in ptrInt.*

***\*ptrX*** *= Z ; // The value in variable Z copied into the address found in ptrX .*

*// \*ptrInt – Use of the asterick is called ‘dereference’ or ‘indirection’*

*// Think of the \* cancelling out ptr, and all you are left with in a variable* = variable or

// or X = Z *which means copy the value in Z to the variable X.*

***Structures***

*A structure is a method of combining several variables into a group/unit call struct.*

*The structure is called a User Defined Type.*

*Since a structure is a TYPE, you can use it in a declaration statement.*

*Syntax: struct structName { declarations …. };*

*Syntax Declaration statement: datatype Identifier.*

*Example:*

*struct Person {*

*string FirstName;*

*int Age;*

*};*

*Declaration of a structure*

*Syntax: structureName Identifier:*

*Example:*

*Person P1; // Declares an instance of struct.*

*// Recall a declaration allocates memory. In the case of a declaration of a struct*

*// the computer will allocate memory for each of the variable in memory*

*Recall the Dot Notation to access the specific member in the structure.*

*Syntax: DeclaredStructreName dot MemberName*

*Example: P1.Firstname = ‘Joe’;*

*cout << P1.Firstnamel*

***Declaration of a pointer to a Structure.***

*Syntax : datatype \* Indentifier;*

*Example:*

*Person P2 ;*

*P2.FirstName = ‘Tuyet’;*

*P2.Age = 21;*

*Person \* ptrPerson;*

*ptrPerson = &P2;*

*You can use ptrPerson to assign values to the variable whose address is in ptrPerson;*

*(\*ptrPerson).Firstname = ‘Paula’;*

*(\*ptrPerson).Age = 18;*

*Note that the expression ‘* ***(\*ptrName).****’ can be replace with an alternative set of symbols ‘****ptrName->****’.*

*Example:*

*(\*ptrPerson). Is the same as ptrPerson ->*

*So ptrPerson -> Firstname = ‘Howard’;*

*ptrPerson -> Age = 24;*

### Exercise 1: Printing Memory Addresses

1. Write a short program that declares and initializes (to any value you like) a double, an int, and a string. Your program should then print the address of, and value stored in, each of the variables. Use the format string "%p" to print the addresses in hexadecimal notation (base 16). You should see addresses that look something like this: "0xbfe55918". The initial characters "0x" tell you that hexadecimal notation is being used; the remainder of the digits give the address itself.
2. Since hexadecimal needs 16 digits (to represent digits 0-15), we use 0-9 and also a-f. Note that a single hex character can express the same values that a 4-bit "nibble" can. How many hex characters are needed to express a single byte? How many bytes are used to store each address? Are your variables located in the first or the second half of the set of all possible memory addresses?
3. Draw a small memory diagram showing the location of each of the variables in your program. (You need not convert the hex to decimal; just label your drawing in hex.) Are they allocated in the same order that you declared them? Is there any empty space between them?
4. Modify your program by rearranging the variable declarations and/or changing the length of the string. (In particular, try a string that uses 5 or 7 bytes, including the null terminator.) Does this change the results you got previously?

#### Explanation: Small changes within a program can change how memory is laid out for a given program. The compiler will try to arrange memory for optimal performance, and this may include aligning variables with 4-byte boundaries. For C++ programmers, this can sometimes mean that a program which appears to work correctly (but in fact overwrites the end of an array), can suddenly stop working due to seemingly innocuous changes -- for example, changing the order in which variables are declared.

### Exercise 2: Writing a short program that has a Swap Function

1. Write a function that accepts two variables of the same data type and swaps their values. Then add a "driver" function (i.e., main) to test your swap routine. Does your function work as you expected?
2. Note that the function will not work if you pass the variables themselves. If your function does not work, modify it such that you pass it the addresses of the variables you wish to swap. Using this approach, you should be able to get it to work correctly.

**Exercise 3:** Write a program that references a structure variable after it is de-allocated and describe what

happens and why it is good or not good to do it.

**Exercise 4:** Write a program that return a pointer to a local structure variable that was declared in a function

and try to use it. Describe what happens and why it is good or not good to do it.

**Exercise 5:** Write a program that reference a pointer to a structure before it is allocated and describe what

happens and why it is good or not good to do it.

### Exercise 6: Write a program that declares a pointer ( allocates Memory) in an infinite loop and

### describe what happens and why it is good or not good to do it.