CS-116: Program 06

What is a **pointer**?

A **pointer** is the address of a location in computer memory.

Wikipedia explains eloquently. See italics below.

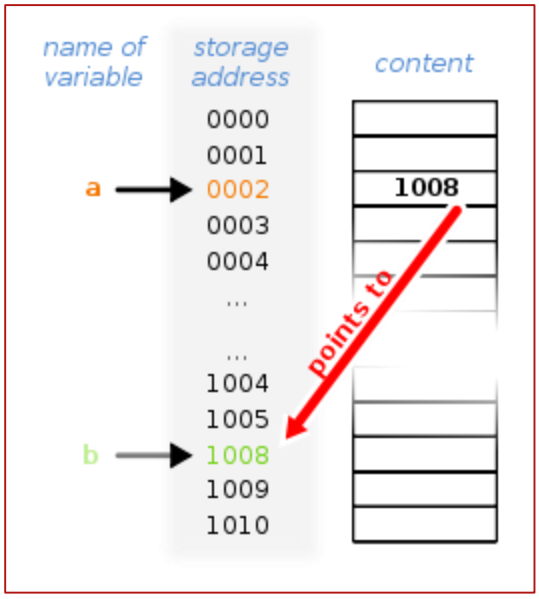

FIGURE 1: EXAMPLE OF A POINTER (WIKIPEDIA)

*In computer science, a **pointer** is a programming language object, whose value refers directly to (or "**points** to") another value stored elsewhere in the computer memory using its address. For high-level programming languages, pointers effectively take the place of general purpose registers in low-level languages such as assembly language or machine code, but may be in available memory. A pointer references a location in memory, and obtaining the value stored at that location is known as dereferencing the pointer. A pointer is a simple, more concrete implementation of the more abstract reference data type. Several languages support some type of pointer, although some have more restrictions on their use than others. As an analogy, a page number in a book's index could be considered a pointer to the corresponding page; dereferencing such a pointer would be done by flipping to the page with the given page number. Pointers to data significantly improve performance for repetitive operations such as traversing strings, lookup tables, control tables and tree structures. In particular, it is often much cheaper in time and space to copy and dereference pointers than it is to copy and access the data to which the pointers point.* Source: http://en.wikipedia.org/wiki/Pointer_(computer_programming)

How do I code using pointers?

When coding it is necessary to understand the difference between a value, an address, a pointer and a dereferenced pointer. A code example appears below.

```
int main() {
  int i = 0;                        // declare an integer
  int* p = &i;                      // declare a pointer to an integer

  // print i, the value at the memory location,
  // and &i, the address of the memory location,
  // and p, the pointer to the memory location,
  // and *p, the value at the memory location, as de-referenced by pointer
  cout << i << " " << &i << " " << p << " " << *p << endl;

  return 0;

}
```

The output of this code is: `0 0xffefbcc8 0xffefbcc8 0`

The `0` is just the value of the variable `i`. The `0xffefbcc8`is the hexadecimal memory address containing `i`.

What is a **linked list**?

Again, from Wikipedia:

In computer science, a linked list is a data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of a data and a reference (in other words, a link) to the next node in the sequence; more complex variants add additional links. This structure allows for efficient insertion or removal of elements from any position in the sequence.

Source: http://en.wikipedia.org/wiki/Linked_list.

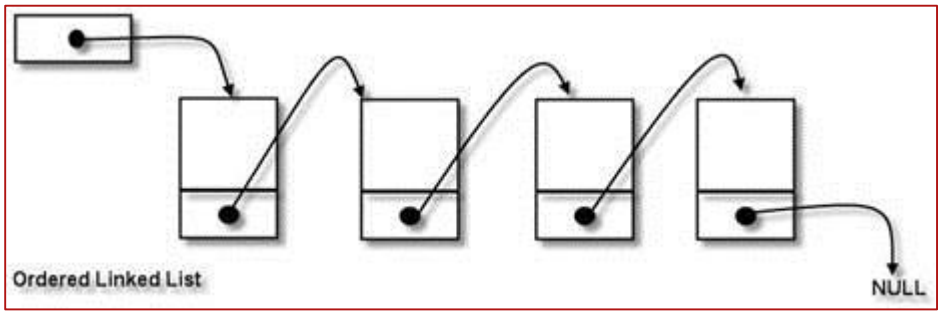

FIGURE 2: SOURCE: HTTP://WWW.SQA.ORG.UK/E-LEARNING/LINKEDDS03CD/IMAGES/PIC001.JPG

Why use linked lists? Why not always use arrays?

Here is an incomplete breakdown of pros and cons of arrays and linked lists.

	Array	*Linked List*
Pro	Convenient addressing: array[i] Performance (computes faster)	More complex data types Inserting into sorted list is easy Dynamic size
Con	Simple data types only Insert/delete in sorted list is hard Predefined size	Can't address *i*th ithem easily (This means some searches, like binary searches, are not possible) Extra memory required for pointers

Recall: What is a **struct**?

We can create our own data structures in C++ using the **`struct`** declaration. Example:

```
struct Book {
  string DeweyDecimalCode;
  string Author;
  string Title;
  string Publisher;

};
```

Recall: What is an **array**?

We can declare arrays by using **`[]`** after a variable name. Example:

```
int myArray[] = { 2, 4 ,6, 23, 19 };

};
```

How can structs and pointers together organize data into linked lists?

See code below. See also: http://cslibrary.stanford.edu/103/LinkedListBasics.pdf.

```
#include <iostream>
using namespace std;

void initNode(int n);
void addNode (int n);

struct Node {
  int x;
  Node *next;
};

Node *head = new Node;

int main() {

  Node *pn;

  initNode(5);
  cout << "head: " << head << endl;
  addNode(10);
  cout << "head: " << head << endl;
  addNode(20);
  cout << "head: " << head << endl;

  pn = head;
  while (pn)
  {
    cout << pn->x << " " ;
    pn = pn->next;
  }

  cout << endl;

  return 0;
}

void initNode(int n)
{
  head->x = n;
  head->next = NULL;
}

void addNode(int n)
{
  struct Node *NewNode = new Node;
  NewNode->x = n;
  NewNode->next = head;
  head = NewNode;
}
```

The output of this code is

```
head: 0x85c4438
head: 0x85c4568
head: 0x85c4590
20 10 5
```

Program 06 **Practice**:

Add function called deleteNode(int n)

Program 06 **Assignment**:

Classes work almost exactly like structs. The main difference is that struct members are public by default, where class members are private by default.

Your assignment is to create a linked list of patient records, sorted by patient ID.

The program should do the following.

1. Read each patient record from the input file into a linked list of objects. Class should have default constructor which sets all numbers to zero and strings to NULL, and override constructor which allows passing values contained in the input file record.
2. Sort the linked list by patient ID.
3. Print the list

Diagram must be in NS-Chart format

Code comments will be graded on how clearly they explain what code is doing.