§10 Data Structures

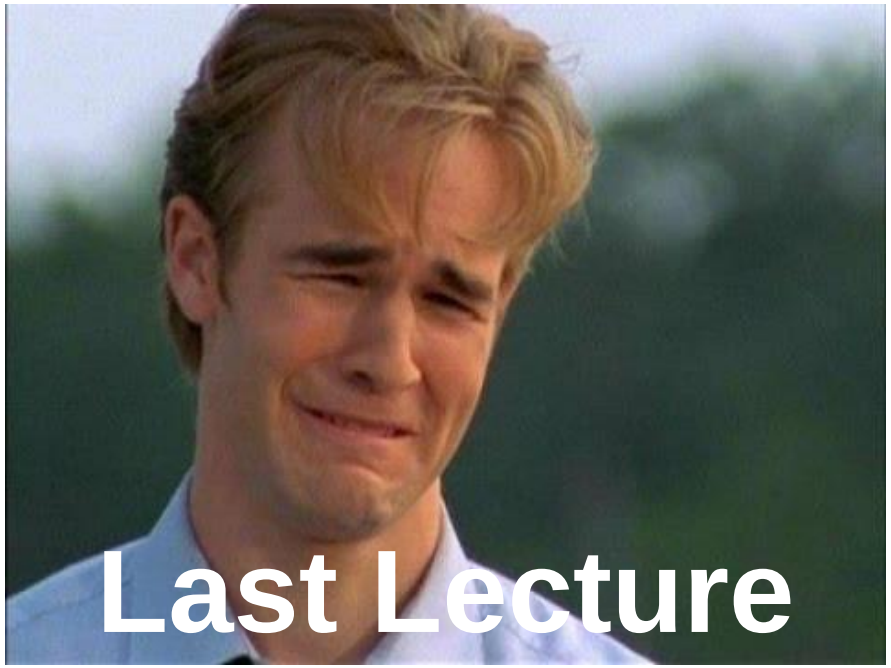
ENGG1111

Computer Programming and Applications

Dirk Schnieders

Outline

1. Dynamic Arrays
   - Search
   - Insert
   - Remove
2. Linked Lists
   - Search
   - Insert
   - Remove



1. Dynamic Arrays

Dynamically Allocated Arrays

- The new operator can be used to produce a new nameless array variable
- Unlike regular arrays, the size of a dynamically allocated array does not need to be a constant
  - It can be determined during program execution
- A dynamically allocated array can be destroyed using the delete operator to free up the memory allocated to it

Dynamically Allocated Arrays - Example

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4    int n;
5    cin >> n;
6    int *p = new int[n];
7    for (int i=0;i<n;i++)
8      cin >> p[i];
9    for (int i=0;i<n;i++)
10     cout << p[i] << " ";
11   delete[] p;
12 }
```

Dynamic Arrays

- A dynamic array is a variable-size list data structure that allows elements to be added or removed

ENGG1111 Lab 10

Problem 1

Write a function that takes an integer array and its size. The function will return a pointer to the largest element in the array.

Problem 2

One problem with dynamically allocated arrays is that once the array is created using the new operator, the size cannot be changed. For example, you might want to add or delete entries from the array. This problem asks you to create functions that uses dynamically allocated arrays and allows add and deletion of entries to the array.

Complete the following two functions

string* addEntry (string *array, int &size, string newEntry);

This function should create a new dynamically allocated array, one element larger than array, copy all elements from array into the new array, add the new entry onto the end of the new array, increment size, delete array, and return the new dynamic array.

string* deleteEntry(string *array, int &size, string entryToDelete);

This function should search array for entryToDelete. If not found, the request should be ignored and the unmodified array returned. If found, create a new dynamically allocated array one element smaller than array. Copy all elements except entryToDelete into the new array, delete array, decrement size, and return the new dynamically allocated array.

Problem 3

Write a program that will read in positive integer values provided by the user and store them in a dynamically allocated array. If the user enters -1, the list is complete, and the average value will be calculated and displayed.

Dynamic Arrays - Example

```
1   #include <iostream>
2   using namespace std;
3 › string* addEntry(string *array, int &size, string newEntry) {⋯
12 › void outputArray(string *array, int size) {⋯
17 › string* deleteEntry(string *array, int &size, string entryToDelete) {⋯
31  int main() {
32    int size = 3;
33    string *list = new string[size];
34    list[0] = "A";
35    list[1] = "B";
36    list[2] = "C";
37    outputArray(list, size);
38    list = addEntry(list, size, "D");
39    outputArray(list, size);
40    list = deleteEntry(list, size, "C");
41    outputArray(list, size);
42  }
```

Dynamic Arrays - Example

```
12  void outputArray(string *array, int size) {
13    for (int i=0;i<size;i++)
14      cout << array[i] << " ";
15    cout << endl;
16  }
```

Dynamic Arrays - Example

```
3   string* addEntry(string *array, int &size, string newEntry) {
4     string* newArray = new string[size+1];
5     for (int i=0;i<size;i++)
6       newArray[i] = array[i];
7     delete[] array;
8     newArray[size] = newEntry;
9     size++;
10    return newArray;
11  }
```

Dynamic Arrays - Example

```
17  string* deleteEntry(string *array, int &size, string entryToDelete) {
18    int index = -1;
19    for (int i=0;i<size;i++)
20      if (array[i] == entryToDelete) index = i;
21    if (index == -1) return array;
22    string* newArray = new string[size-1];
23    for (int i=0;i<index;i++)
24      newArray[i] = array[i];
25    for (int i=index+1;i<size;i++)
26      newArray[i-1] = array[i];
27    delete[] array;
28    size--;
29    return newArray;
30  }
```

Dynamic Arrays - Example

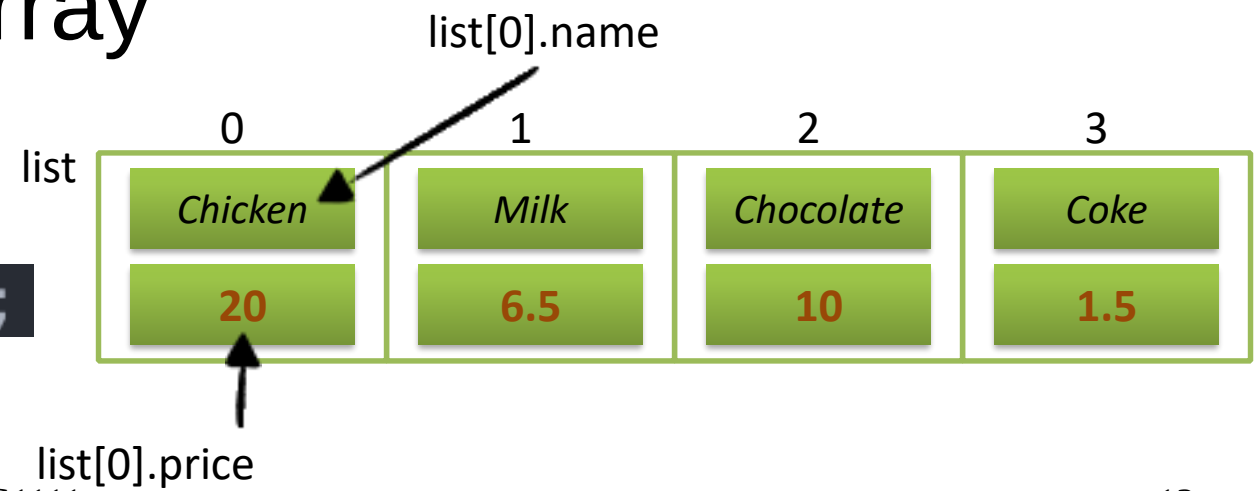
```
1  #include <iostream>
2  using namespace std;
3  string* addEntry(string *array, int &size, string newEntry) {…
12 void outputArray(string *array, int size) {…
17 string* deleteEntry(string *array, int &size, string entryToDelete) {…
31 int main() {
32   int size = 3;
33   string *list = new string[size];
34   list[0] = "A";
35   list[1] = "B";
36   list[2] = "C";
37   outputArray(list, size);
38   list = deleteEntry(list, size, "C");
39   list = deleteEntry(list, size, "B");
40   outputArray(list, size);
41   list = deleteEntry(list, size, "A");
42   list = addEntry(list, size, "A");
43   outputArray(list, size);
44   list = addEntry(list, size, "B");
45   outputArray(list, size);
46 }
```

Dynamic Array of Struct

- Let's write a program that stores a list of products, and allows:
  A. Product search
  B. Product insertion
  C. Product deletion
- We will use a dynamic array


```
struct Product {
  string name;
  double price;
};
```

```
Product *list = new Product[size];
```



A. Search

```
1  #include <iostream>
2  using namespace std;
3  struct Product {
4    string name;
5    double price;
6  };
7  void print(Product *list, int size) {
11 void search(Product *list, string searchName, int size) {
17 int main() {
18     int size = 4;
19     Product *list = new Product[size];
20     list[0].name = "Chicken";
21     list[0].price = 20;
22     list[1].name = "Milk";
23     list[1].price = 6.5;
24     list[2].name = "Chocolate";
25     list[2].price = 10;
26     list[3].name = "Coke";
27     list[3].price = 1.5;
28     print(list, size);
29     cout << "Milk costs ";
30     search(list, "Milk", size);
31 }
```

A. Search

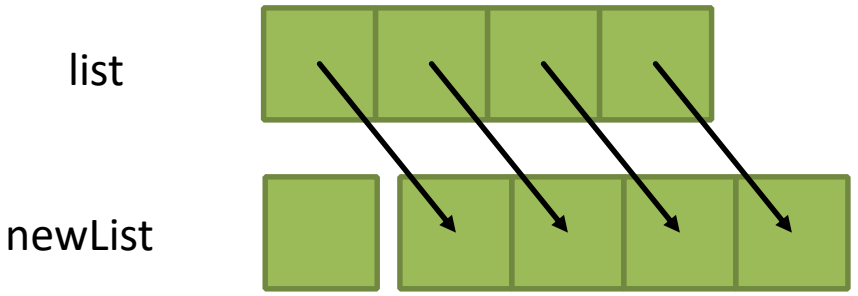
```
 7  void print(Product *list, int size) {
 8      for (int i=0;i<size;i++)
 9          cout << list[i].name << ": " << list[i].price << endl;
10  }
11  void search(Product *list, string searchName, int size) {
12      for (int i=0;i<size;i++) {
13          if (list[i].name == searchName)
14              cout << list[i].price << endl;
15      }
16  }
```

B. Insert

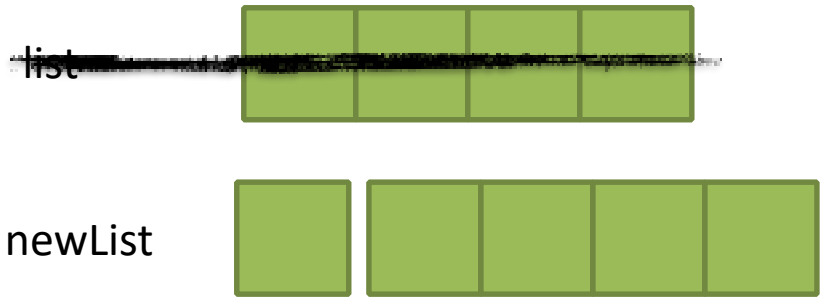
1. Create a larger newList
   - One more slot
   - Copy the content from list to newList
2. Delete list
   - As list is not used anymore, delete it and free the memory
3. Add the new product
   - Now the dynamically allocated array newList has enough slots to accommodate the new product

B. Insert

1.



2.



3.

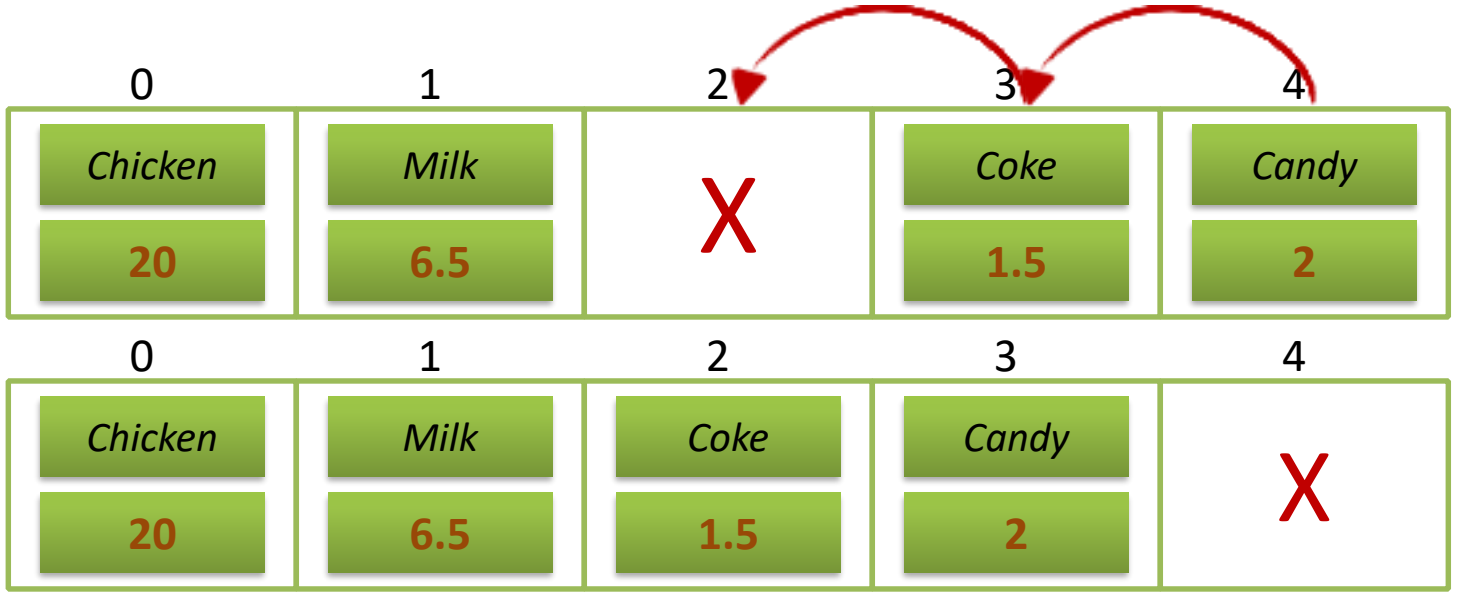
B. Insert

```
#include <iostream>
using namespace std;
struct Product {…
void print(Product *list, int size) {…
void search(Product *list, string searchName, int size) {…
Product* insert(Product *list, string name, double price, int &size) {…
int main() {
    int size = 4;
    Product *list = new Product[size];
    list[0].name = "Chicken";
    list[0].price = 20;
    list[1].name = "Milk";
    list[1].price = 6.5;
    list[2].name = "Chocolate";
    list[2].price = 10;
    list[3].name = "Coke";
    list[3].price = 1.5;
    print(list, size);
    list = insert(list, "Candy", 90, size);
    print(list, size);
}
```

B. Insert

```
#include <iostream>
using namespace std;
struct Product {…
void print(Product *list, int size) {…
void search(Product *list, string searchName, int size) {…
Product* insert(Product *list, string name, double price, int &size) {
    Product* newList = new Product[size+1];
    for (int i=1;i<=size;i++)
        newList[i] = list[i-1];
    delete [] list;
    newList[0].name = name;
    newList[0].price = price;
    size++;
    return newList;
}
int main() {…
```

C. Remove


0
1
2
3
4
Chicken
20
Milk
6.5
X
Coke
1.5
Candy
2
0
1
2
3
4
Chicken
20
Milk
6.5
Coke
1.5
Candy
2
X

C. Remove

ENGG1111 Lab 11

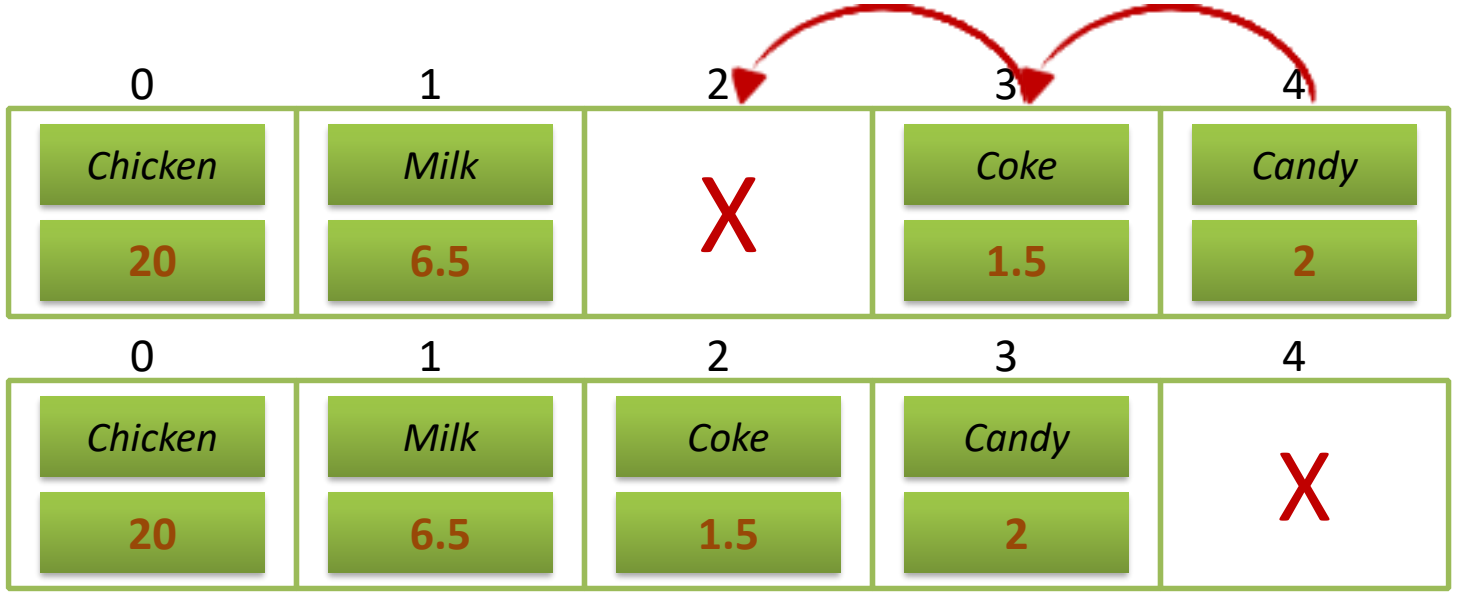
Problem 1

Consider the following program.

```
1  #include <iostream>
2  using namespace std;
3  struct Product {
4    string name;
5    double price;
6  };
7  void print(Product *list, int size) {
8    for (int i=0;i<size;i++)
9      cout << list[i].name << ": " << list[i].price << endl;
10 }
11 void remove(Product *&list, int &size) {=
21 int main() {
22   int size = 4;
23   Product *list = new Product[size];
24   list[0].name = "Chicken";
25   list[0].price = 20;
26   list[1].name = "Milk";
27   list[1].price = 6.5;
28   list[2].name = "Chocolate";
29   list[2].price = 10;
30   list[3].name = "Coke";
31   list[3].price = 1.5;
32   print(list, size);
33   remove(list, size);
34   cout << endl;
35   print(list, size);
36 }
```

Implement the function void remove(Product *&list, int &size) that will remove the last element in the list of products.

C. Remove

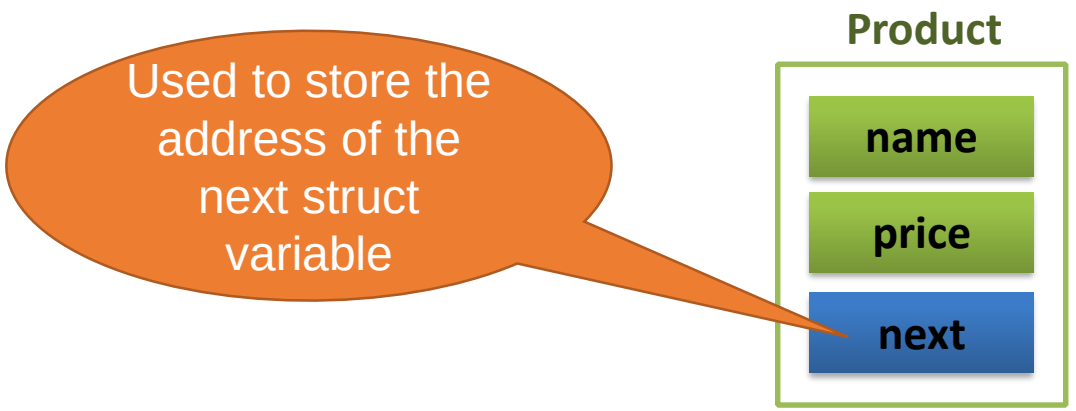


- With a long dynamic array, the insertion and removal of nodes can be time consuming
- On the following slides, we will explore an alternative data structure that does not suffer from this problem

2. Linked List

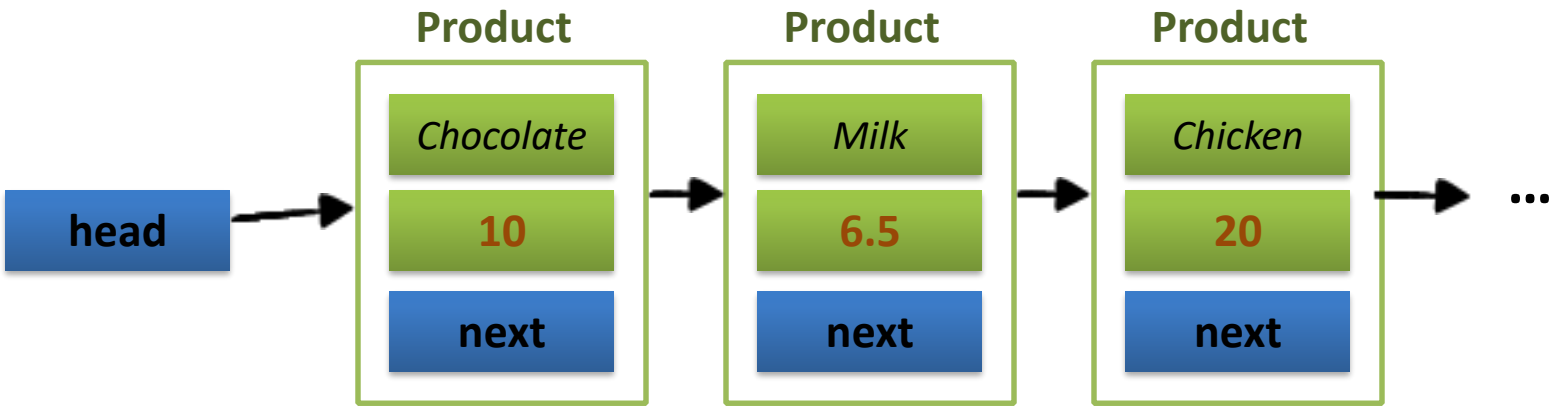
2. Linked List

- A fundamental data structure
- Allows insertion and removal of items at any point in the list in constant time
- Makes use of the concepts of structure, pointers and dynamically allocated variables

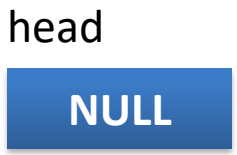



```
struct Product{
    string name;
    double price;
    Product *next;
};
```

Linked List


Product
Product
Product
Chocolate
Milk
Chicken
head
10
6.5
20
next
next
next
...

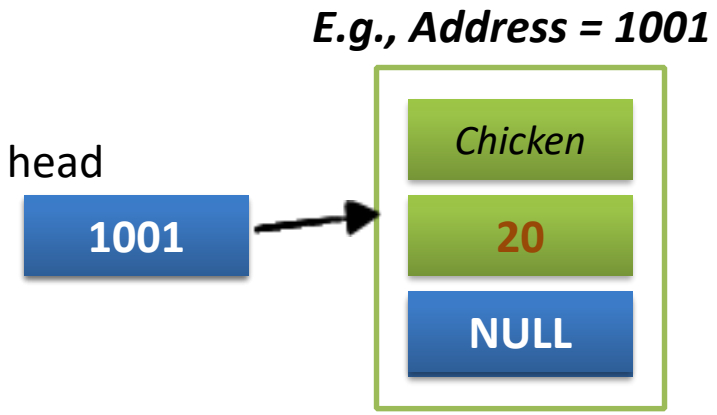
Declaration




```
Product *head = NULL;
```

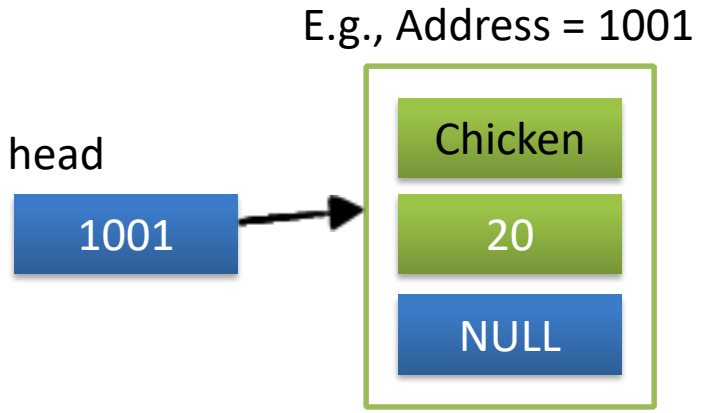
- In order to access a linked list, a pointer variable often called head is used to store a pointer to the first node of the list
- Initially when the linked list is empty (i.e., a linked list with no nodes), head will simply contain a NULL pointer

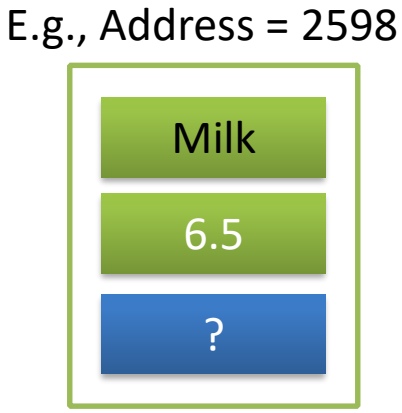
Insert



- Starting from an empty list, new nodes are created and inserted into the linked list
- A NULL pointer is assigned to the pointer variable of the last node to indicate the end of the linked list
- Node insertion in this example is taking place at the head of the linked list

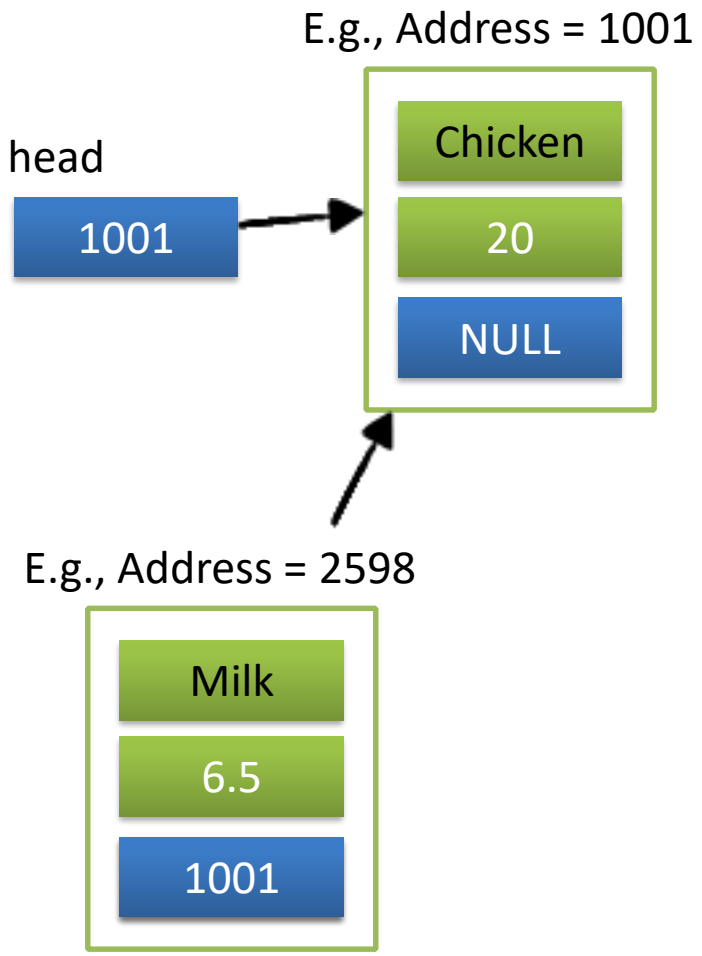
Insert





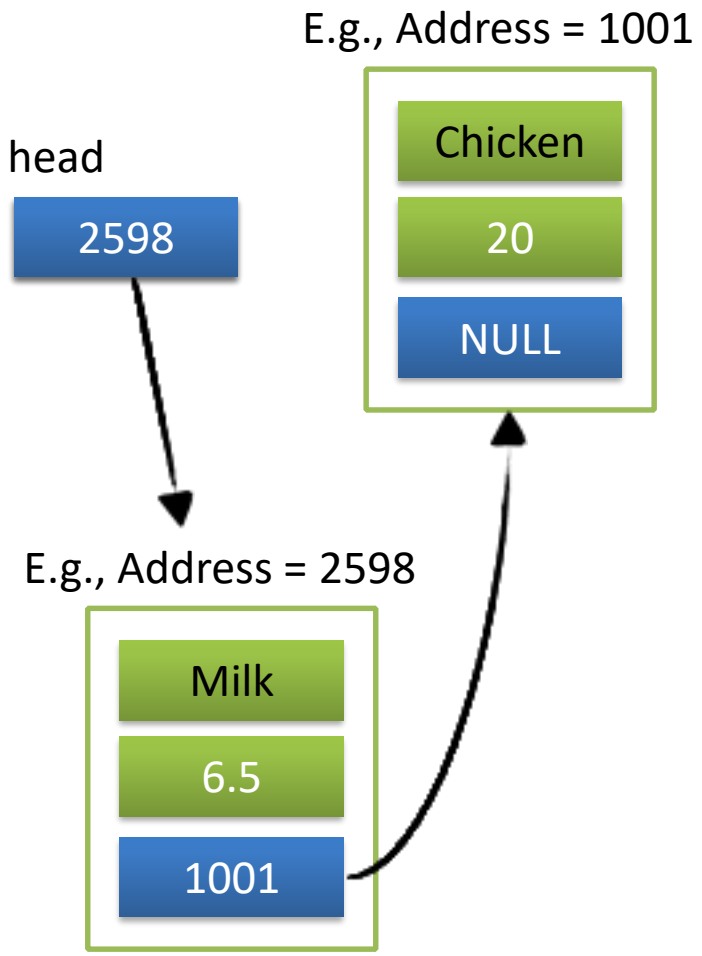
- Insertion steps
  1. Create a new node

Insert



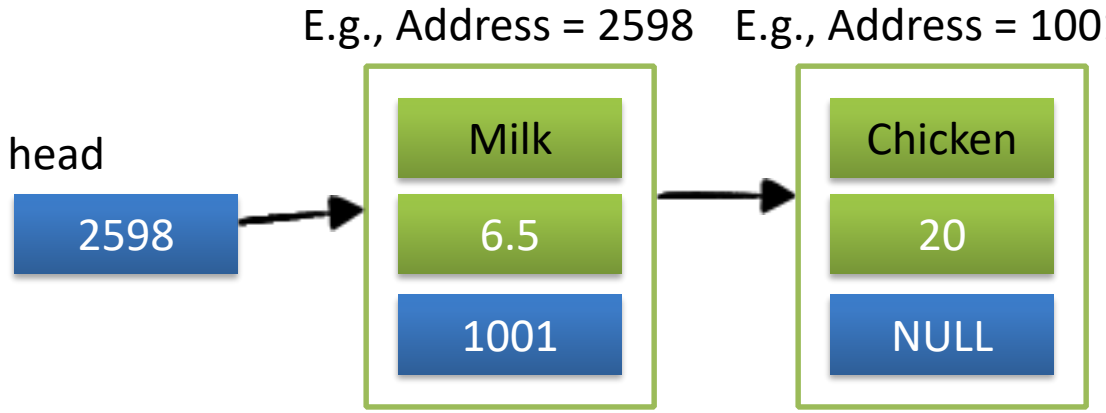
- Insertion steps
  1. Create a new node
  2. New node's next is the first node

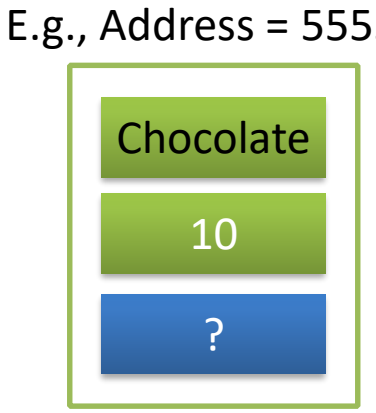
Insert



- Insertion steps
  1. Create a new node
  2. New node's next is the first node
  3. head points to the new node

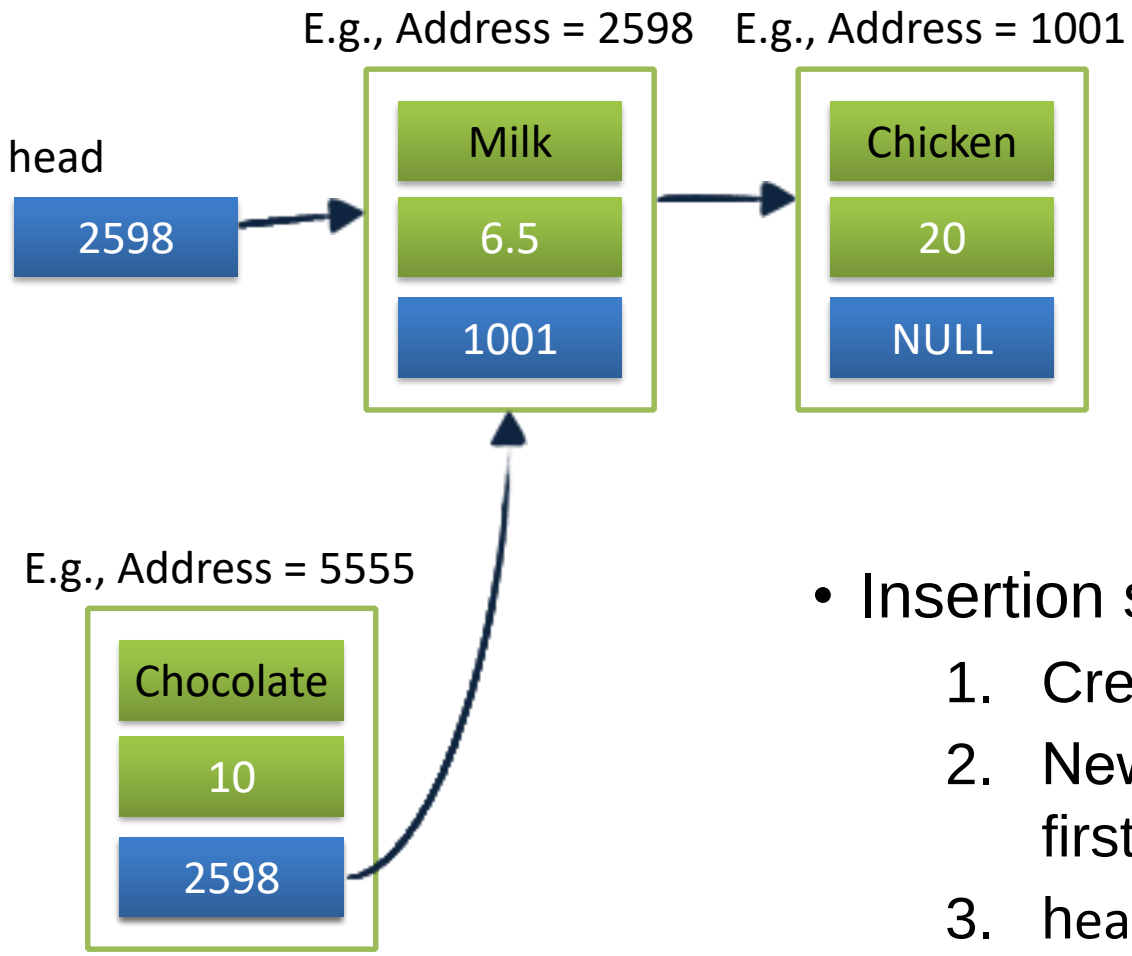
Insert





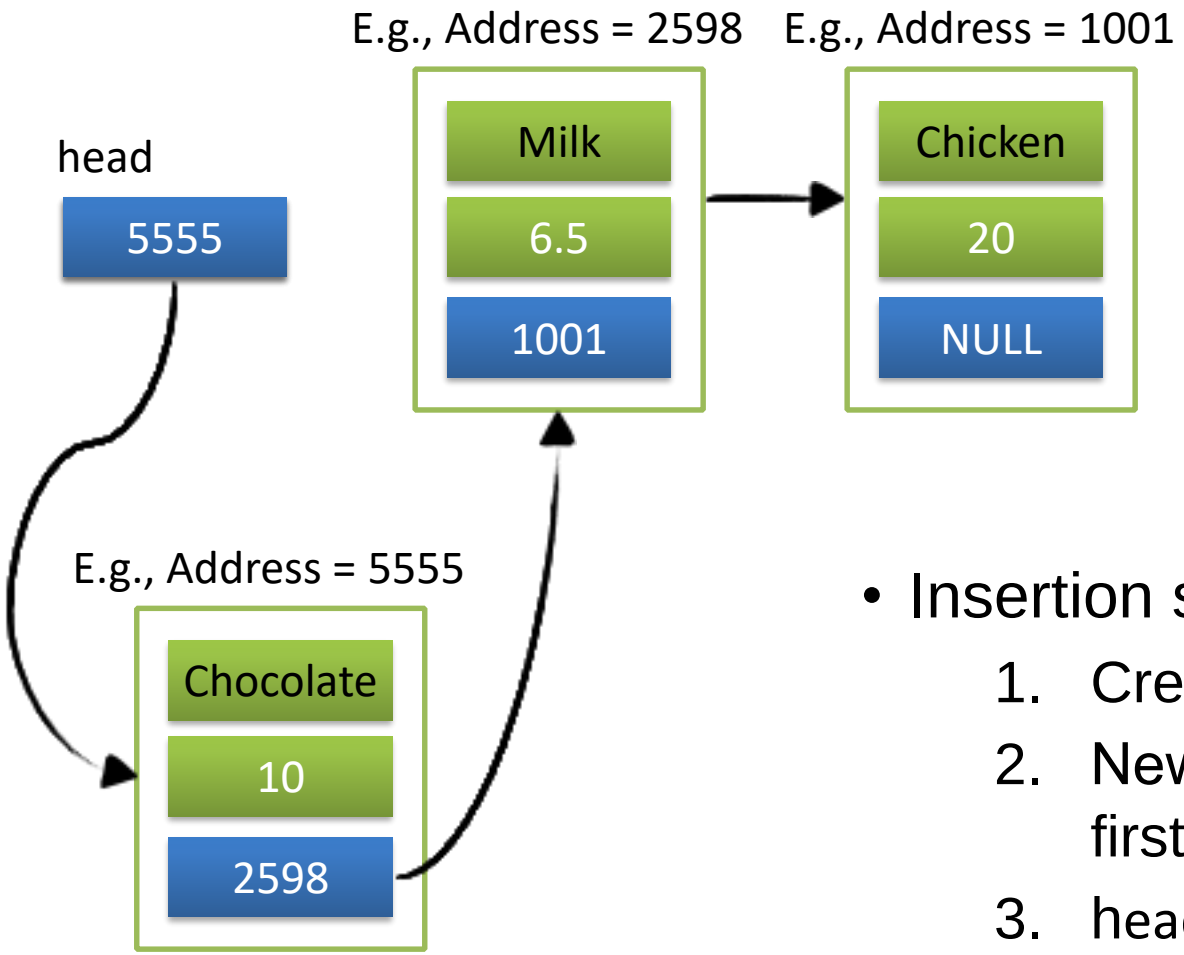
- Insertion steps
  1. Create a new node
  2. New node's next is the first node
  3. head points to the new node

Insert



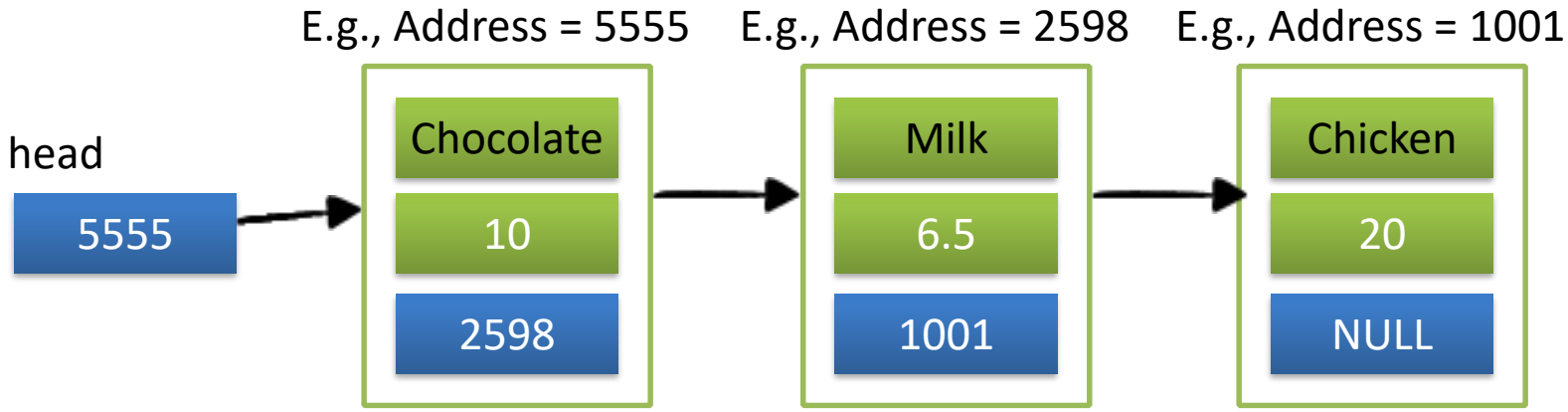
- Insertion steps
  1. Create a new node
  2. New node's next is the first node
  3. head points to the new node

Insert



- Insertion steps
  1. Create a new node
  2. New node's next is the first node
  3. head points to the new node

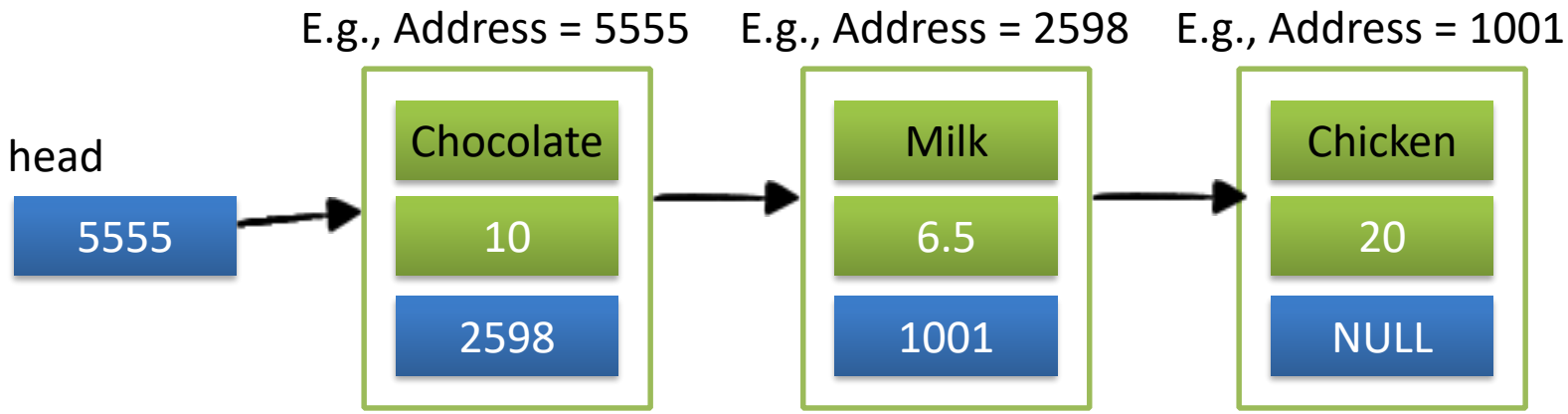
Insert




```
9   Product* insert(Product *head, string name, double price) {
10      Product *p = new Product;
11      p->name = name;
12      p->price = price;
13      p->next = head;
14      head = p;
15      return head;
16  }
```

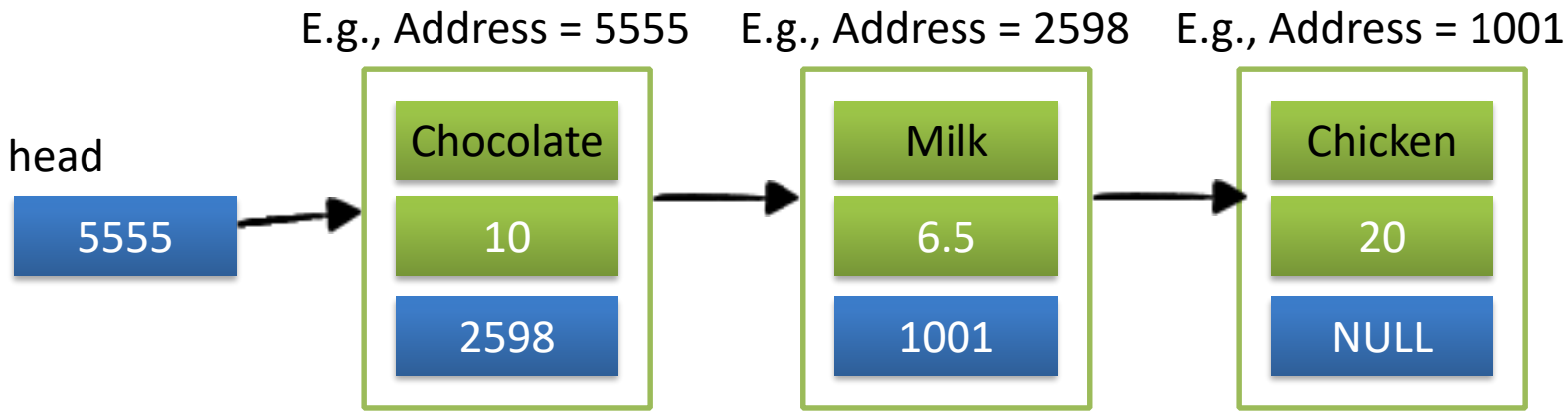
```
21  int main() {
22      Product *head = NULL;
23      head = insert(head, "Chicken" , 20);
24      head = insert(head, "Milk" , 6.5);
25      head = insert(head, "Chocolate" , 10);
26      print(head);
27  }
```

Search



- Searching steps
  1. Create a pointer current
  2. Initialize current=head
  3. Traverse the linked list using: current=current->next

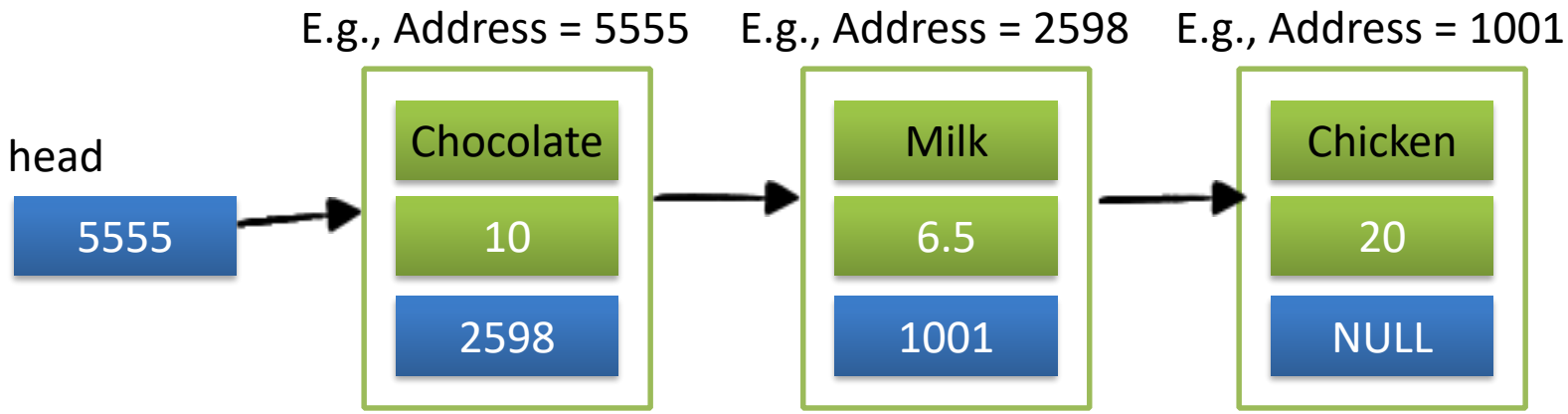
Search




```
17  void search(Product *head, string searchName) {
18      for (Product *current = head;current != NULL;current = current -> next) {
19          if (current -> name == searchName)
20              cout << current->price << endl;
21      }
22  }
```

```
29  int main() {
30      Product *head = NULL;
31      head = insert(head, "Chicken" , 20);
32      head = insert(head, "Milk" , 6.5);
33      head = insert(head, "Chocolate" , 10);
34      print(head);
35      search(head, "Milk");
36  }
```

Remove

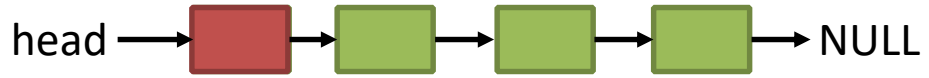


- Deletion steps (Except for first node)
  1. Create 2 pointers current, previous
  2. Search for the node to delete, and update the pointers: previous->next=current->next
  3. Remove the node: delete current
     - update the pointers

Remove

Question

- What if the node to be removed is the first node?



Remove

```
32  void remove(Product *&head, string n) {
33      Product *current, *previous;
34      previous = NULL;
35      current = head;
36      if (current->name == n){
37          head = current->next;
38          delete current;
39          return;
40      }
41      while (current != NULL){
42          if (current->name == n){
43              previous->next = current->next;
44              delete current;
45              return;
46          }
47          previous = current;
48          current = current->next;
49      }
50  }
```

Conclusion

Learning Outcomes

1. [Computational mind]
   Able to identify possible solutions for problems based on computer programs
2. [Program implementation]
   Able to implement solutions for problems using C++
3. [Program comprehension]
   Able to understand programs written by others and participate in larger scale system implementation

Examination Information

Examination Information

Date: THU, Dec. 14
Time: 9:30 am - 12:30 pm
Venue: Lindsay Ride Sp. Ctr.

Format: The paper consists of four parts.
[Part A] Write down the output of programs in the space provided. (40%)
[Part B] Questions with same scope as assignment 1 (20%)
[Part C] Questions with same scope as assignment 2 (20%)
[Part D] Questions with same scope as assignment 3 (20%)

Students are permitted to bring to the examination one sheet of A4-sized paper with printed or written notes on both sides.

No calculator allowed.

CCST9049 Scientific and Technological Literacy

From Human Vision to Machine Vision

THE UNIVERSITY OF HONG KONG

The SETL Online Evaluation

Instructions for Students

Please go to the weblink http://setl.hku.hk where you should see a list of all forms that you need to complete. Please note that your evaluation will be saved anonymously, without any identification. There are separate forms for the course, teacher (and if appropriate) tutor and demonstrator. There are instructions, an FAQ link and a link for you to report any missing courses at the weblink.

Students should:

- Access the weblink (http://setl.hku.hk/) and see the instructions, FAQ and a list of forms for completion (for each course, there are separate links to the course, teacher, tutor and field trip/demonstrator forms);
- Click the "submit" button when you have completed all the questions in a form (the form can no longer be retrieved once you have clicked the "submit" button.);
- Continue to complete other forms, if any, by repeating the above process;
- If you cannot find the course that you have enrolled in, send an email to SSRC using the link on the page showing the status of evaluation to inform them about the missing course. SSRC staff would rectify this promptly.

November 11, 2013