1 Skip List Erase [ / 23]

In this problem you will complete the implementation of a recursive function to erase a specific value from a skip list. Remember that a skip list stores data in sorted order. Each level of a skip list chains together approximately half as many nodes as the previous level (skipping approximately every other node). We will assume our skip list stores positive integers, which allows us to use '0' as a special dummy head node.

1.1 Diagram [ / 3]

First, edit the diagram below to erase the value '6' from this skip list.

```
+---+---+---+---+---+---+---+---+
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
```

1.2 Algorithm Analysis [ / 7]

Assuming there are $n$ elements in the skip list and the maximum height of any node is $k$, what is the order notation for the running time of the erase algorithm? In the average (well-balanced) case? In the worst case? What is the relationship between $n$ and $k$ in the average (well-balanced) case? Write 4-5 concise and well-written sentences. You may want to first complete your implementation on the next page.
1.3 Erase Implementation

Now complete the implementation of the recursive **erase** function. The function should return **true** if the element was successfully removed and **false** otherwise. Make sure to handle the general case, not just the example you diagrammed on the previous page.

```cpp
bool erase(Node* before, int value, int level) {
    // sample solution: 13 line(s) of code
}

bool erase(Node* head, int value) {
    return erase(head, value, head->height-1);
}
```
Write a recursive function named `copy_except` that takes in a pointer to the root of a binary search tree and a value and returns a pointer to a full copy of the data, except the indicated value has been removed from the tree.

```cpp
template <class T> class Node {
public:
    Node(const T& v) :
        value(v),left(NULL),right(NULL) {}
    T value;
    Node* left;
    Node* right;
};
```

*sample solution: 25 line(s) of code*
3 Censoring Science [ / 27]

Alyssa P. Hacker works in the technology department at the Center for Disease Control (CDC). But she is disgusted by the most recent programming assignment she’s been asked to do. Below is a sample CDC document she needs to process. She’s been asked to make a “helpful” index of all the words in the document sorted in reverse character order (starting with the last letter in each word). Each word is accompanied by the position(s) of the occurrences of each word in the file. The corresponding output is show on the right.

the center for disease control employs evidence-based approaches to reduce disease in vulnerable populations

the diversity of experience at the center for disease control supports innovations in public health

3.1 Forbidden Words (Not a Joke) [ / 3]

You will notice that some words are missing from the index. Yes, indeed, these words are now forbidden from appearing in CDC budget documents. Which words are missing? Hint: there are 3 in this sample.

3.2 Reverse String Helper Function [ / 4]

Write a recursive helper function named reverse that returns the flipped version of its single argument.

3.3 Runtime Analysis Order Notation [ / 8]

Finish the implementation (next page), then analyze each step. Assume $n$ total words and $u$ unique words in the document, a max of $k$ occurrences of any word, $f$ forbidden words, and $s$ letters in the longest word.
3.4 Implementation

Now complete the implementation below to produce this not-so-helpful index.

```cpp
std::map<> index;

// STEP 1: READ DOCUMENT
std::ifstream istr("cdc_doc.txt");
std::string s;
int i = 1;
while (istr >> s) {
    // sample solution: 1-3 line(s) of code
}

// STEP 2: PROCESS LIST OF FORBIDDEN WORDS
std::ifstream istr2("cdc_forbidden.txt");
while (istr2 >> s) {
    // sample solution: 1-3 line(s) of code
}

// STEP 3: OUTPUT THE INDEX
for (auto it = index.begin(); it != index.end(); ) {
    std::cout << std::setw(12) << it->first << " ";
    for (auto &item : it->second) {
        std::cout << std::setw(2) << item << " ";
    }
    std::cout << std::endl;
}
```


4 Challenge Accepted! [ / 6]

Ben Bitdiddle claims that it’s impossible to write a recursive program without writing any helper functions – he means a program with no functions except main! Can you do it?

Write a complete C++ program (specify all of the #includes, etc.) to compute integer powers. After compiling it (e.g., g++ -std=c++11 integer_power.cpp -o int_pow.out), executing the program with 2 arguments on the command line (e.g., ./int_pow.out 3 4) should print the answer to stdout (e.g., 81 in this example because $3^4 = 81$).

integer_power.cpp

sample solution: 24 line(s) of code

Analyze the running time of your program using order notation. Assume $b$ is the base and $p$ is the power.
### 5 Short Answer [ /17]

#### 5.1 Comparing Vectors & Arrays [ /5]

The statements below can be used to compare and contrast arrays and vectors. For each statement, specify “ARRAY” if it is only true for arrays, “VECTOR” if it is only true for vectors, “BOTH” if it is true for both types, and “NEITHER” if it is true for neither type.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows how many elements it contains.</td>
<td>BOTH</td>
</tr>
<tr>
<td>Can be used to store elements of any type.</td>
<td>NEITHER</td>
</tr>
<tr>
<td>Prevents access of memory beyond its bounds.</td>
<td>ARRAY</td>
</tr>
<tr>
<td>Is dynamically re-sizable.</td>
<td>VECTOR</td>
</tr>
<tr>
<td>Can be passed by reference.</td>
<td>VECTOR</td>
</tr>
</tbody>
</table>

#### 5.2 Limited Looping [ /3]

**True or False** There are some algorithms that must be written using a `for` loop and *cannot* be written using a `while` or `do – while` loop.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Choice</th>
</tr>
</thead>
</table>

### 6 Concurrency and Asynchronous Computing [ /3]

Why might a group of dining philosophers starve?

A) Because it’s impossible to eat spaghetti with chopsticks.

B) Because they are all left-handed.

C) Because due to a bank error they didn’t have enough money in their joint account.

D) Because they didn’t all want to eat at the same time.
7 Superhero Division [ 14]

In this problem you will add a new operator to the Superhero class from lab. Remember that a superhero has a name, a true identity, and a power, but we cannot access the true identity of a Superhero object from the public interface. Here is the basic Superhero class declaration:

```cpp
class Superhero {
public:
    // ACCESSORS
    const string& getName() const { return name; }
    const string& getPower() const { return power; }
    // INPUT STREAM OPERATOR
    friend istream& operator>>(istream &istr, Superhero &hero);
private:
    // REPRESENTATION
    string name;
    string true_identity;
    string power;
};
// OUTPUT STREAM OPERATOR
ostream& operator<<(ostream &ostr, const Superhero &hero);
```

And here is part of the Superhero class implementation:

```cpp
ostream& operator<<(ostream &ostr, const Superhero &hero) {
    if (hero.getPower() == "")
        ostr << hero.getName() << " has no power" << endl;
    else
        ostr << "Superhero " << hero.getName() << " has power " << hero.getPower() << endl;
    return ostr;
}
```

Now let’s define the /= operator on Superhero. This operator can be used to defeat a hero by dividing them from their true identity. If an attacker learns a hero’s true identity and uses it against them, the superhero loses his power. A superhero must carefully guard his true identity to prevent this attack. If the attacker does not know and just incorrectly guesses the superhero’s true identity, this /= operation does nothing. For example, suppose elastigirl is a Superhero object with name equal to “Elastigirl”, true identity equal to “Zoe”, and power equal to “Flexible”. Then the statement:

```cpp
cout << elastigirl;
```

would print this on the screen:

Superhero Elastigirl has power Flexible

But after executing the statement:

```cpp
elastigirl /= ("Zoe");
```

the output of the variable elastigirl would print on the screen as:

Elastigirl has no power
7.1 Implementation Choices [ /5]

Name the three different ways we can implement operator overloading. Which of these three is the most appropriate choice for the /= operator described above? Why?

7.2 /= operator implementation [ /9]

Now implement the /= operator. Part of your job is to carefully define the prototype for this function. What should be added or changed in the superhero.h class declaration file? And what should be added or changed in the superhero.cpp class implementation file? Be specific.
8 Valet Parking Maps

You have been asked to help with a valet parking system for a big city hotel. The hotel must keep track of all of the cars currently stored in their parking garage and the names of the owners of each car. *Please read through the entire question before working on any of the subproblems.* Here is the simple `Car` class they have created to store the basic information about a car:

```cpp
class Car {
public:
    // CONSTRUCTOR
    Car(const string &m, const string &c) : maker(m), color(c) {}
    // ACCESSORS
    const string& getMaker() const { return maker; }
    const string& getColor() const { return color; }
private:
    // REPRESENTATION
    string maker;
    string color;
};
```

The hotel staff have decided to build their parking valet system using a map between the cars and the owners. This map data structure will allow quick lookup of the owners for all the cars of a particular color and maker (e.g., the owners of all of the silver Hondas in the garage). For example, here is their data structure and how it is initialized to store data about the six cars currently in the garage.

```cpp
map<Car, vector<string>> cars;
cars[Car("Honda","blue")]\:.push\_back("Cathy");
cars[Car("Honda","silver")]\:.push\_back("Fred");
cars[Car("Audi","silver")]\:.push\_back("Dan");
cars[Car("Toyota","green")]\:.push\_back("Alice");
cars[Car("Audi","silver")]\:.push\_back("Erin");
cars[Car("Honda","silver")]\:.push\_back("Bob");
```

The managers also need a function to create a report listing all of the cars in the garage. The statement:

```cpp
print_cars(cars);
```

will result in this report being printed to the screen (`std::cout`):

```
People who drive a silver Audi:
    Dan
    Erin
People who drive a blue Honda:
    Cathy
People who drive a silver Honda:
    Fred
    Bob
People who drive a green Toyota:
    Alice
```

Note how the report is sorted alphabetically by maker, then by car color, and that the owners with similar cars are listed chronologically (the order in which they parked in the garage).
8.1 The Car class [ /6]

In order for the Car class to be used as the first part of a map data structure, what additional non-member function is necessary? Write that function. Carefully specify the function prototype (using const & reference as appropriate). Use the example above as a guide.

8.2 Data structure diagram [ /10]

Draw a picture of the map data structure stored by the cars variable in the example. As much as possible use the conventions from lecture for drawing these pictures. Please be neat when drawing the picture. 
Optional: You may also write a few concise sentences to explain your picture.
8.3 print_cars

Write the print_cars function. Part of your job is to correctly specify the prototype for this function. Be sure to use const and pass by reference as appropriate.

8.4 remove_cars

When guests pick up their cars from the garage, the data structure must be correctly updated to reflect this change. The remove_car function returns true if the specified car is present in the garage and false otherwise.

```cpp
bool success;
success = remove_car(cars, "Erin", "silver", "Audi");
assert (success == true);
success = remove_car(cars, "Cathy", "blue", "Honda");
assert (success == true);
success = remove_car(cars, "Sally", "green", "Toyota");
assert (success == false);
```
After executing the above statements the `cars` data structure will print out like this:

- **People who drive a silver Audi:**
  - Dan

- **People who drive a silver Honda:**
  - Fred
  - Bob

- **People who drive a green Toyota:**
  - Alice

Note that once the only blue Honda stored in the garage has been removed, this color/maker combination is completely removed from the data structure.

Specify the prototype and implement the `remove_car` function.
9 Garbage Collection [12]
For each of the real world systems described below, choose the *most appropriate* memory management technique. Each technique should be used exactly once.

A) Explicit Memory Management (C++)  B) Reference Counting
C) Stop & Copy  D) Mark-Sweep

9.1 Student Registration System [3]
Must handle the allocation and shuffling of pointers as students register and transfer in and out of classes. Memory usage will not be a deciding factor. Fragmentation of data should be minimized.

9.2 Playing Chess [3]
Implementation of a tree-based algorithm for searching the game space. Remember that a *tree* is a *graph* with no cycles.

9.3 Webserver [3]
A collection of infrequently changing interconnected webpages. Any memory usage overhead should be low. Pauses in service are tolerable.

9.4 Hand Held Game (e.g., GameBoy or PSP, etc.) [3]
Performance critical application with extremely limited memory resources.
10 Short Answer [ 22]

10.1 Garbage Identification [ 7]

To which address in the memory below should the root variable point so that exactly 2 cells are garbage? Draw a box and pointer diagram to justify your answer and state which 2 cells are garbage.

<table>
<thead>
<tr>
<th>address</th>
<th>100</th>
<th>101</th>
<th>102</th>
<th>103</th>
<th>104</th>
<th>105</th>
<th>106</th>
<th>107</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
</tr>
<tr>
<td>left</td>
<td>106</td>
<td>106</td>
<td>107</td>
<td>100</td>
<td>102</td>
<td>101</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>right</td>
<td>103</td>
<td>105</td>
<td>105</td>
<td>0</td>
<td>105</td>
<td>101</td>
<td>0</td>
<td>101</td>
</tr>
</tbody>
</table>

10.2 Stop and Copy Garbage Collection [ 4]

What is the purpose of the forwarding address in Stop and Copy garbage collection? What will go wrong if you neglect to record this value? Write 2 or 3 concise and well-written sentences.
10.3  Concurrency and Asynchronous Computing

When programming with multiple threads or processes, the correct use of mutexes (locks) and condition variables will ensure that:

A) The program always returns the exact same answer.
B) The program returns an answer that was not possible if the program ran sequentially.
C) The entire program is atomic.
D) Each student in a large class will be able to successfully copy a complete set of lecture notes (with no repetitions), even if there are multiple professors.
E) Deadlock will be avoided if there are multiple mutexes, but may still happen in systems with a single lock.

10.4  Perfect Hashing for Image Compression

For the last homework, you implemented a compression scheme for 2D images. What are the drawbacks of using this format as the underlying representation for an image editing program? What types of edits to the image are simple? What types of edits will be comparatively inefficient to process? Write 3-4 concise and well-written sentences.
Indicate by letter the data structure(s) that have each characteristic listed below.

- A) vector
- B) list
- C) map
- D) set
- E) priority queue
- F) hash table
- G) leftist heap

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Data Structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>allows efficient (sublinear) removal of the first and last elements (or the minimum and maximum elements)</td>
<td>A, E</td>
</tr>
<tr>
<td>uses an array or vector as the underlying representation</td>
<td>A, F, G</td>
</tr>
<tr>
<td>uses a network of nodes connected by pointers as the underlying representation</td>
<td>B, D, G</td>
</tr>
<tr>
<td>the underlying data structure must be “balanced” or well-distributed to achieve the targeted performance</td>
<td>B, D, G</td>
</tr>
<tr>
<td>requires definition of <code>operator&lt;</code> or <code>operator&gt;</code></td>
<td>B, D, G</td>
</tr>
<tr>
<td>entries cannot be modified after they are inserted (requires re-insertion or re-processing of position)</td>
<td>D, G</td>
</tr>
<tr>
<td>duplicates are not allowed</td>
<td>B, D, G</td>
</tr>
<tr>
<td>allows sublinear merging of two instances of this data structure</td>
<td>D, G</td>
</tr>
</tbody>
</table>

17
Match the order notation with each fragment of code. Two of the letters will not be used.

A) \(O(n)\)  
B) \(O(1)\)  
C) \(O(n^n)\)  
D) \(O(n^2)\)  
E) \(O(2^n)\)  
F) \(O(\log n)\)  
G) \(O(n \log n)\)  
H) \(O(\sqrt{n})\)  

```cpp
vector<int> my_vector;
// my_vector is initialized with n entries
// do not include initialization in performance analysis
for (int i = 0; i < n; i++) {
    my_vector.erase(my_vector.begin());
}
```

```cpp
map<string, int> my_map;
// my_map is initialized with n entries
// do not include initialization in performance analysis
my_map.find("hello");
```

```cpp
int foo(int n) {
    if (n == 1 || n == 0) return 1;
    return foo(n-1) + foo(n-2);
}
```

```cpp
int k = 0;
for (int i = 0; i < sqrt(n); i++) {
    for (int j = 0; j < sqrt(n); j++) {
        k += i*j;
    }
}
```

```cpp
set<string> my_set;
for (int i = 0; i < n; i++) {
    string s;
    cin >> s;
    my_set.insert(s);
}
```

```cpp
float* my_array = new float[n];
// do not include memory allocation in performance analysis
my_array[n/2] = sqrt(n);
```
In this problem we will explore a simple class to manage the assignment of people to offices and desks. Each Office object stores its name, the number of desks it can hold, and the names of the people assigned to those desks. An office also stores a reference to a master queue of all the people who still need to be assigned to desks. When an office is constructed, people are assigned to the office from the front of this master queue. When an office is demolished, the people who were assigned to that office should be added to the end of the queue while they wait for a new office assignment. Here is the partial declaration of the Office class:

```cpp
class Office {
public:
    Office(const string& name, int num_desks, queue<string> &unassigned);
    friend ostream& operator<<(ostream &ostr, const Office &office);
private:
    // representation
    string _name;
    int _num_desks;
    string* _desks;
    queue<string> & _unassigned; // a reference to the master queue
};
```

In the example below we create the master queue of people who need to be assigned to desks in offices, and create and delete several Office objects:

```cpp
queue<string> unassigned;
unassigned.push("Alice");
unassigned.push("Bob");
unassigned.push("Cathy");
unassigned.push("Dan");
unassigned.push("Erin");
unassigned.push("Fred");
unassigned.push("Ginny");

Office *red = new Office("red", 4, unassigned);
Office *green = new Office("green", 2, unassigned);
cout << *red << *green;
delete red;
cout << "After deleting the red office, "
    << unassigned.size() << " people are waiting for desks." << endl;

Office *blue = new Office("blue", 3, unassigned);
cout << *blue;
cout << "Before deleting the blue & green offices, "
    << unassigned.size() << " people are waiting for desks." << endl;
delete green;
delete blue;
cout << "After deleting all of the offices, "
    << unassigned.size() << " people are waiting for desks." << endl;
```

19
Here is the desired output from this example:

The red office has 4 desks:
desk[0] = Alice
desk[1] = Bob
desk[2] = Cathy
desk[3] = Dan

The green office has 2 desks:
desk[0] = Erin
desk[1] = Fred

After deleting the red office, 5 people are waiting for desks.

The blue office has 3 desks:
desk[0] = Ginny
desk[1] = Alice
desk[2] = Bob

Before deleting the blue & green offices, 2 people are waiting for desks.

After deleting all of the offices, 7 people are waiting for desks.

Here is the implementation of the constructor, as it appears in the `office.cpp` file:

```cpp
Office::Office(const string& name, int num_desks, queue<string> &unassigned)
    : _name(name), _num_desks(num_desks), _unassigned(unassigned) {
    _desks = new string[_num_desks]; // allocate the desk space
    for (int i = 0; i < _num_desks; i++) {
        if (_unassigned.size() > 0) { // assign from the master queue
            _desks[i] = _unassigned.front();
            _unassigned.pop();
        } else { // if there are no unassigned people, leave the desk empty
            _desks[i] = "";
        }
    }
}
```

13.1 Classes and Memory Allocation [ /10]

Anytime you write a new class, especially those with dynamically allocated memory, it is very important to consider the member functions that the compiler will automatically generate and determine if this default behavior is appropriate. List these 4 important functions by their generic names, **AND** write their prototypes as they would appear within the `Office` class declaration.
13.2 Declaring a Destructor [ /3]

The Office class is incomplete and requires implementation of a custom destructor so that people assigned to demolished offices are returned to the master queue and memory is deallocated as appropriate to avoid memory leaks. What line needs to be added to the header file to declare the destructor? Be precise with syntax. Where should this line be added: within the public, protected, or private interface?

13.3 Implementing a Destructor [ /12]

Implement the destructor, as it would appear in the office.cpp file.
13.4 Operator Overloading

Here is the implementation of the << stream operator as it appears within the office.cpp file:

```cpp
ostream& operator<<(ostream &ostr, const Office &o) {
    ostr << "The " << o._name << " office has "
    << o._num_desks << " desks:" << endl;
    for (int i = 0; i < o._num_desks; i++) {
        ostr << " desk[" << i << "] = " << o._desks[i] << endl;
    }
    return ostr;
}
```

There are three different ways to overload an operator: as a non-member function, as a member function, and as a friend function. Which method was selected for the Office object << stream operator? What are the reasons for this choice? Discuss why the other two methods are inappropriate or undesirable. Write 3 or 4 concise and thoughtful sentences.
Write a function that takes an STL list of integers, finds the even numbers, and places them in a dynamically-allocated array. Only the space needed for the even numbers should be allocated, and no containers other than the given list and the newly-created array may be used. As an example, given a list containing the values:

3 10 -1 5 6 9 13 14

the function should allocate an array of size 3 and store the values 10, 6 and 14 in it. It should return, via arguments, both the pointer to the start of the array and the number of values stored. No subscripting may be used — not even *(a+i) in place of a[i]. Here is the function prototype:

void even_array(const list<int>& b, int* & a, int& n);
A ternary tree is similar to a binary tree except that each node has at most 3 children. Write a recursive function named `EqualsChildrenSum` that takes one argument, a pointer to the root of a ternary tree, and returns true if the value at each non-leaf node is the sum of the values of all of its children and false otherwise. In the examples below, the tree on the left will return true and the tree on the right will return false.
template <class T> class priority_queue {
public:
    // CONSTRUCTOR
    priority_queue() {}
    // ACCESSORS
    int size() { return m_heap.size(); }
    bool empty() { return m_heap.empty(); }
    const T& top() const { assert(!m_heap.empty()); return m_heap[0]; }
    // MODIFIERS
    void push(const T& entry) {
        m_heap.push_back(entry);
        this->percolate_up(int(m_heap.size()-1));
    }
    void pop() { // find and remove the element with the smallest value
        assert(!m_heap.empty());
        m_heap[0] = m_heap.back();
        m_heap.pop_back();
        this->percolate_down(0);
    }
    void pop_max() { /* YOU WILL IMPLEMENT THIS FUNCTION */ }
private:
    // HELPER FUNCTIONS
    void percolate_up(int i) {
        T value = m_heap[i];
        while (i > 0) {
            int parent = (i-1)/2;
            if (value >= m_heap[parent]) break; // done
            m_heap[i] = m_heap[parent];
            i = parent;
        }
        m_heap[i] = value;
    }
    void percolate_down(int i) {
        T value = m_heap[i];
        int last_non_leaf = int(m_heap.size()-1)/2;
        while (i <= last_non_leaf) {
            int child = 2*i+1, rchild = 2*i+2;
            if (rchild < m_heap.size() && m_heap[child] > m_heap[rchild])
                child = rchild;
            if (m_heap[child] >= value) break; // found right location
            m_heap[i] = m_heap[child];
            i = child;
        }
        m_heap[i] = value;
    }
    // REPRESENTATION
    vector<T> m_heap;
};
16.1 Implementing \texttt{pop\_max}

Write the new priority queue member function named \texttt{pop\_max} that finds and removes from the queue the element with the largest value. Carefully think about the efficiency of your implementation. Remember that a standard priority queue stores the smallest value element at the root.

16.2 Analysis

If there are \( n \) elements in the priority queue, how many elements are visited by the \texttt{pop\_max} function in the worst case? What is the order notation for the running time of this function?
What is the output of the following program?

class A {
public:
    virtual void f() { cout << "A::f\n"; }
    void g() { cout << "A::g\n"; }
};

class B : public A {
public:
    void g() { cout << "B::g\n"; }
};

class C : public B {
public:
    void f() { cout << "C::f\n"; }
    void g() { cout << "C::g\n"; }
};

int main() {
    A* a[3];
    a[0] = new A();
    a[1] = new B();
    a[2] = new C();

    for (int i = 0; i < 3; i++) {
        cout << i << endl;
        a[i]->f();
        B* b = dynamic_cast<B*>(a[i]);
        if (b) b->g();
    }
}
For the *last expression* in each fragment of code below, give the *type* (int, vector<double>, Foo*, etc.) and the *value*. If the value is a legal address in memory, write “memory address”. If the value hasn’t been properly initialized, write “uninitialized”. If there is an error in the code, write “error”. You may want to draw a picture to help you answer each question, but credit will only be given for what you’ve written in the boxes.

```cpp
double a = 5.2;
double b = 7.5;
a + b
```

<table>
<thead>
<tr>
<th>Type:</th>
<th>Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```cpp
int *d;
int e[7] = { 15, 6, -7, 19, -1, 3, 22 };  
d = e + e[5];  
*d
```

<table>
<thead>
<tr>
<th>Type:</th>
<th>Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```cpp
bool *f = new bool;  
*f = false;  
f
```

<table>
<thead>
<tr>
<th>Type:</th>
<th>Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```cpp
int g = 10;  
int *h = new int[g];  
h[0]
```

<table>
<thead>
<tr>
<th>Type:</th>
<th>Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```cpp
map<string, int> m;  
m.insert(make_pair(string("bob"), 5551111));  
m.insert(make_pair(string("dave"), 5552222));  
m.insert(make_pair(string("alice"), 5553333));  
m.insert(make_pair(string("chris"), 5554444));  
(*m.find("bob"))->second
```

<table>
<thead>
<tr>
<th>Type:</th>
<th>Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>