Bracket matching using stacks

- suppose we want to check if all the different bracket forms in our program match up correctly: { }, ( ), [ ]
- refer to { ([as *opening* brackets, })] as *closing* brackets
- for each bracket type the number of closing brackets must equal the number of opening brackets
- order within a bracket type matters: can't have a closing bracket before the thing it is meant to close, e.g.] [isn't valid
- order between bracket types matters: can't have a closing bracket "cross the boundary" of another bracket type waiting to be closed, e.g. not valid: ([ )]

Idea for checking

- we can keep track of which brackets are currently open and waiting to be closed, and what order they're in
- new opening brackets can come in at any time
- when a closing bracket is encountered we can match it against the most recent open bracket, e.g.
  - currently open: (({ ([
    - if we see) or } it is invalid, since we have to close] first
    - if it matches ok then we can throw away the opener, e.g.
  - revised open list after seeing a] would be (({ (

stack-based algorithm

- have a stack of chars, will use to store opening brackets
- read input file one char at a time
- ignore characters that aren't brackets
- if we see an opening bracket we push it on the stack
- if we see a closing bracket we check against top of stack
  - if they match then we pop the top open bracket off the stack
  - otherwise it's an error (bracket mismatch)
- if the stack isn't empty when we reach the end of the file that's an error (unmatched opening brackets still on the stack)

Example:

```
int main()
{
   int arr[3] = { 1, 2, 3 };
   float y = sqrt(arr[0]);
}
```

ignoring non-brackets, sequence to process is
( ) { [ ] { } ([ ]) }

action sequence:	updated stack with top on the right ->
push (	(
match) against top, so pop	
push {	{
push [	{ [
match] against top, so pop	{
push {	{ {
match } against top, so pop	{
push (	{ (
push [	{ ([
match] against top, so pop	{ (
match) against top, so pop	{
match } against top, so pop	
end of input, stack is empty, pass!	

sample code: the stack interface

```
// assume a typical stack interface, pop/top/push return true iff successful

class stack {
   private:
       // could be array or list approach for a stack of chars

   public:
       stack();
      ~stack();
      bool pop();
      bool top(char &b);
      bool push(char b);
      int size();
};
```

sample code: main routine

```
// main gets the filename and handles opening/closing, checkbrackets does rest
int main() {
   ifstream infile;
   string fname;
   cout << “Enter the filename”;
   cin >> fname;
   infile.open(fname);
   if (infile.is_open()) {
      if (checkbrackets(infile)) {
         cout << “file passed: all brackets matched” << endl;
      } else {
         cout << “file failed” << endl;
      }
      infile.close();
   } else {
      cout << “Unable to open file “ << fname << endl;
   }
}
```

sample code: helper functions

- use three helper functions to check if given char is a bracket, if it is opening bracket, if it is closing bracket
- bool isbracket(char b)
  - return true if b is any of { [(}])
- bool isopener(char b)
  - return true if b is any of { [(
- bool iscloser(char b)
  - return true if b is any of }])

sample code: bracket checker

```
void bracketchecker(ifstream &infile) {
   stack brackets;  // stack of opening brackets, initially empty
   // read each char in file, ignoring anything that isn't a bracket
   while (!infile.eof()) {
        char b;
        infile >> b;
        if (!infile.eof() && isbracket(b)) {
           if (!updatestack(b, brackets)) {
              return false; // quit now and return false, we've already detected a bracket issue
           }
        }
   }
   // reached end of file, see if anything leftover in stack
   if (brackets.size() > 0) {
      cout << “Error: “ << brackets.size() << “ unmatched brackets in the file” <, endl;
      return false;
   }
   return true;
}
```

sample code: check/update stack

```
bool updatestack(char b, stack brackets)
{
   // handle case where b is an opening bracket
   if (isopen(b)) {
      if (!brackets.push(b)) {
         cout << “Error: unable to finish processing, stack full?” << endl;
         return false;
      } else {
         return true;
      }
   }

   // continued on next slide
```

stack update continued

```
  else {
     char openB;  // see which open bracket is on top of stack
     if (!brackets.top(openB)) {
        cout << “Error: found “ << b << “ when no brackets were open” << endl;
        return false;
     }
     // check for mismatch between opener and closer
     if ((((openB == '{') && (b != '}')) || ((openB == '[') && (b != ']')) || ((openB == '(') && (b != ')'))) {
        cout << “Error: tried to close “ << openB << “ with “ << b << endl;
        return false;
     }
     // otherwise the closing bracket matched the open one, pop the opener (should succeed)
     if (!brackets.pop()) {
        cout << “Error: unexpected failure to pop from a non-empty stack?” << endl;
        return false;
     }
  }
  return true; // processed b, no errors were detected
}
```