



## AP Computer Science AB 2000 Student Samples

**The materials included in these files are intended for non-commercial use by AP teachers for course and exam preparation; permission for any other use must be sought from the Advanced Placement Program. Teachers may reproduce them, in whole or in part, in limited quantities, for face-to-face teaching purposes but may not mass distribute the materials, electronically or otherwise. These materials and any copies made of them may not be resold, and the copyright notices must be retained as they appear here. This permission does not apply to any third-party copyrights contained herein.**

These materials were produced by Educational Testing Service (ETS), which develops and administers the examinations of the Advanced Placement Program for the College Board. The College Board and Educational Testing Service (ETS) are dedicated to the principle of equal opportunity, and their programs, services, and employment policies are guided by that principle.

The College Board is a national nonprofit membership association dedicated to preparing, inspiring, and connecting students to college and opportunity. Founded in 1900, the association is composed of more than 3,900 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three million students and their parents, 22,000 high schools, and 3,500 colleges, through major programs and services in college admission, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT<sup>®</sup>, the PSAT/NMSQT<sup>™</sup>, the Advanced Placement Program<sup>®</sup> (AP<sup>®</sup>), and Pacesetter<sup>®</sup>. The College Board is committed to the principles of equity and excellence, and that commitment is embodied in all of its programs, services, activities, and concerns.

Copyright © 2001 by College Entrance Examination Board. All rights reserved. College Board, Advanced Placement Program, AP, and the acorn logo are registered trademarks of the College Entrance Examination Board.

- (a) Write overloaded operator<, as started below. If the precedence of the token lhs is less than the precedence of rhs, then operator< should return true. Tokens of type PLUS should have lower precedence than tokens of type TIMES, which should have lower precedence than tokens of type NUMBER.

Complete operator< below.

```
bool operator< (const Token & lhs, const Token & rhs)
```

```
{  
    if ((lhs.op == PLUS && rhs.op != PLUS) || (lhs.op == TIMES && rhs.op == NUMBER))  
        return true;  
    return false;  
}
```

Part (b) begins on page 12.

Complete function InfixToPostfix below.

```

void InfixToPostfix(const apqueue<Token> & infixQ,
                   apqueue<Token> & postQ)
// precondition: the sequence of tokens in infixQ represents a
//               valid infix expression using +, *, and integers;
//               postQ is empty
// postcondition: the sequence of tokens in postQ represents a
//               valid postfix expression equivalent to the
//               infix expression represented by infixQ
{
    apqueue<Token> tempQ = infixQ;
    Token temp, temp2; apstack<Token> tempS;
    while (!tempQ.isEmpty())
    {
        tempQ.dequeue(temp);
        if (temp.op == NUMBER)
            postQ.enqueue(temp);
        else
        {
            if (tempS.isEmpty())
                tempS.push(temp);
            else
            {
                while (!tempS.isEmpty() && (temp < tempS.top() || temp.op == tempS.top().op))
                {
                    tempS.pop(temp2); postQ.enqueue(temp2);
                }
                tempS.push(temp);
            }
        }
    }
    while (!tempS.isEmpty())
    {
        tempS.pop(temp2); postQ.enqueue(temp2);
    }
}

```

- (a) Write overloaded `operator<`, as started below. If the precedence of the token `lhs` is less than the precedence of `rhs`, then `operator<` should return `true`. Tokens of type `PLUS` should have lower precedence than tokens of type `TIMES`, which should have lower precedence than tokens of type `NUMBER`.

Complete `operator<` below.

```
bool operator< (const Token & lhs, const Token & rhs)
```

```
{
```

```
if (lhs.op == PLUS && rhs.op == PLUS)
    return true;
else if (lhs.op == PLUS && rhs.op == TIMES)
    return false;
else if (lhs.op == TIMES && rhs.op == PLUS)
    return true;
else if (lhs.op == TIMES && rhs.op == TIMES)
    return true;
else if (lhs.op == NUMBER && rhs.op == PLUS)
    return true;
else if (lhs.op == NUMBER && rhs.op == TIMES)
    return true;
else
    return false;
}
```

```
if (lhs.op < rhs.op)
```

```
    return true;
```

```
else
```

```
    return false;
```

```
}
```

Part (b) begins on page 12.

GO ON TO THE NEXT PAGE.

Complete function InfixToPostfix below.

```
void InfixToPostfix(const apqueue<Token> & infixQ,
                   apqueue<Token> & postQ)
// precondition: the sequence of tokens in infixQ represents a
//                valid infix expression using +, *, and integers;
//                postQ is empty
// postcondition: the sequence of tokens in postQ represents a
//                valid postfix expression equivalent to the
//                infix expression represented by infixQ
```

```
{
    apqueue<Token> tempQ = infixQ;
    astack<Token> tempS;
    token tempVar1, tempVar2;
    while (tempQ.isEmpty() != 0)
    {
        tempQ.dequeue(tempVar1)
        if (tempVar1.op == 2)
            postQ.enqueue(tempVar1);
        Else if (tempS.isEmpty() == true)
            tempS.push(tempVar1);
        else
        {
            while (tempVar1.op < tempS.top().op ||
                  tempVar1.op != tempS.top().op || tempS.isEmpty()
                  == true)
            {
                tempS.pop(tempVar2);
                tempQ.enqueue(tempVar2);
            }
            tempS.push(tempVar1);
        }
    }
}
}
}
}
```

GO ON TO THE NEXT PAGE.

- (a) Write overloaded operator<, as started below. If the precedence of the token lhs is less than the precedence of rhs, then operator< should return true. Tokens of type PLUS should have lower precedence than tokens of type TIMES, which should have lower precedence than tokens of type NUMBER.

NUM > \* > +

+ < \* < NUM

Complete operator< below.

```
bool operator< (const Token & lhs, const Token & rhs) {
    bool answer
    if (lhs.TokenType == PLUS)
        answer = true;
    if (lhs.TokenType == NUMBER)
        answer = false;
    if (lhs.TokenType == TIMES) {
        if (rhs.TokenType == PLUS)
            answer = false;
        if (rhs.TokenType == TIMES)
            answer = false;
        if (rhs.TokenType == NUMBER)
            answer = true;
    }
    return answer;
}
```

Part (b) begins on page 12.

Complete function InfixToPostfix below.

```

void InfixToPostfix(const apqueue<Token> & infixQ,
                   apqueue<Token> & postQ)
// precondition: the sequence of tokens in infixQ represents a
//               valid infix expression using +, *, and integers;
//               postQ is empty
// postcondition: the sequence of tokens in postQ represents a
//               valid postfix expression equivalent to the
//               infix expression represented by infixQ
{
    apqueue<Token> tempQ = infixQ;
    Token curr;
    apstack<Token> operators;

    tempQ.dequeue(curr);
    if (curr.TokenType == NUMBER) {
        postQ.enqueue(curr);
    }
    else {
        if (operators.isEmpty()) {
            operators.push(curr);
        }
        else {
            do {
                if (curr < operators.top() || -) {
                    postQ.enqueue(operators.pop());
                }
            }
            while (!operators.isEmpty() || !curr < operators.top());
        }
    }
    while (!operators.isEmpty()) {
        postQ.enqueue(operators.pop());
    }
}
// end

```

GO ON TO THE NEXT PAGE.