



AP[®] Computer Science AB 2001 Sample Student Responses

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[®], the PSAT/NMSQT[™], the Advanced Placement Program[®] (AP[®]), and Pacesetter[®]. 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 function `MinChild`, as started below. `MinChild` returns a pointer to the child node of `T` that contains the smaller value (or `NULL` if `T` is a leaf node). If `T` has only one child, `MinChild` should return a pointer to that child. If `T` is `NULL`, `MinChild` should return `NULL`.

The following table shows the results of several calls to `MinChild`.

<u>Function call</u>	<u>Return value</u>
<code>MinChild(T2)</code>	<code>P</code>
<code>MinChild(P)</code>	<code>Q</code>
<code>MinChild(Q)</code>	<code>R</code>
<code>MinChild(R)</code>	<code>NULL</code>
<code>MinChild(NULL)</code>	<code>NULL</code>

Complete function `MinChild` below.

```
HeapNode * MinChild(HeapNode * T)
```

```
{  
    if (T == NULL)  
        return NULL;  
    if (T->left == NULL && T->right == NULL)  
        return NULL;  
    if (T->left == NULL)  
        return T->right;  
    if (T->right == NULL)  
        return T->left;  
    if (T->left->val > T->right->val)  
        return T->right;  
    if (T->right->val > T->left->val)  
        return T->left;  
}
```

(b) Write function `IsHeapOrdered`, as started below. `IsHeapOrdered` returns `true` if `T` is a minheap and `false` otherwise.

`T` is a minheap under the following conditions.

- `T` is `NULL`, or
- `T` is a leaf node, or
- `T` is a nonleaf node, the value in `T` is smaller than the values in its children's nodes and its children are minheaps.

In writing `IsHeapOrdered`, you may call function `MinChild` specified in part (a). Assume that `MinChild` works as specified, regardless of what you wrote in part (a).

Complete function `IsHeapOrdered` below.

```
bool IsHeapOrdered(HeapNode * T)
```

```
{
    HeapNode * Child;
    Child = MinChild(T);
    if (Child == NULL)
        return true;
    if (Child->val < T->val)
        return false;
    if (Child->val > T->val)
        return true && IsHeapOrdered(T->left) && IsHeapOrdered(T->right);
}
```

Complete function RemoveMin below.

```
void RemoveMin(HeapNode * & T)  
// precondition: T is not NULL
```

```
{  
    HeapNode * Child;  
    HeapNode * temp = T;  
    Child = MinChild(T);  
    if (Child == NULL)  
    {  
        T = NULL;  
        delete temp;  
    }  
    else  
    {  
        T->val = Child->val;  
        if (Child == T->left)  
            RemoveMin(T->left);  
        else  
            RemoveMin(T->right);  
    }  
}
```

- (a) Write function `MinChild`, as started below. `MinChild` returns a pointer to the child node of `T` that contains the smaller value (or `NULL` if `T` is a leaf node). If `T` has only one child, `MinChild` should return a pointer to that child. If `T` is `NULL`, `MinChild` should return `NULL`.

The following table shows the results of several calls to `MinChild`.

<u>Function call</u>	<u>Return value</u>
<code>MinChild(T2)</code>	<code>P</code>
<code>MinChild(P)</code>	<code>Q</code>
<code>MinChild(Q)</code>	<code>R</code>
<code>MinChild(R)</code>	<code>NULL</code>
<code>MinChild(NULL)</code>	<code>NULL</code>

Complete function `MinChild` below.

```
HeapNode * MinChild(HeapNode * T)
if (T == NULL) return NULL;
else if (T->left == NULL && T->right == NULL)
    return NULL;
else if (T->left == NULL)
    return T->right;
else if (T->right == NULL)
    return T->left;
else if (T->right->val < T->left->val)
    return T->right;
else return T->left;
```

(b) Write function `IsHeapOrdered`, as started below. `IsHeapOrdered` returns true if `T` is a minheap and false otherwise.

`T` is a minheap under the following conditions.

- `T` is NULL, or
- `T` is a leaf node, or
- `T` is a nonleaf node, the value in `T` is smaller than the values in its children's nodes and its children are minheaps.

In writing `IsHeapOrdered`, you may call function `MinChild` specified in part (a). Assume that `MinChild` works as specified, regardless of what you wrote in part (a).

Complete function `IsHeapOrdered` below.

```
bool IsHeapOrdered(HeapNode * T)
```

```
if (T == NULL)
    return true;
```

```
if (T->right == NULL && T->left == NULL)
    return true;
```

```
else
```

```
return (T->val < MinChild(T)->val && IsHeapOrdered(T->right) &&
        IsHeapOrdered(T->left));
```

Complete function RemoveMin below.

```
void RemoveMin(HeapNode * & T)  
// precondition: T is not NULL
```

```
HeapNode * Temp = MinChild(T);
```

```
T->val = MinChild(T)->val;
```

```
RemoveMin(Temp);
```

- (a) Write function `MinChild`, as started below. `MinChild` returns a pointer to the child node of `T` that contains the smaller value (or `NULL` if `T` is a leaf node). If `T` has only one child, `MinChild` should return a pointer to that child. If `T` is `NULL`, `MinChild` should return `NULL`.

The following table shows the results of several calls to `MinChild`.

<u>Function call</u>	<u>Return value</u>
<code>MinChild(T2)</code>	<code>P</code>
<code>MinChild(P)</code>	<code>Q</code>
<code>MinChild(Q)</code>	<code>R</code>
<code>MinChild(R)</code>	<code>NULL</code>
<code>MinChild(NULL)</code>	<code>NULL</code>

Complete function `MinChild` below.

```
HeapNode * MinChild(HeapNode * T)
```

```
{
    if ((T->left == NULL) && (T->right == NULL))
    {
        return NULL;
    }
    else if (T->left == NULL)
    {
        HeapPtr P;
        P = T->right;
        return P;
    }
    else if (T->right == NULL)
    {
        HeapPtr P;
        P = T->left;
        return P;
    }
    else if (T->right->val > T->left->val)
    {
        return T->left;
    }
    else // T->right->val < T->left->val
    {
        return T->right;
    }
}
```

(b) Write function `IsHeapOrdered`, as started below. `IsHeapOrdered` returns true if `T` is a minheap and false otherwise.

`T` is a minheap under the following conditions.

- `T` is NULL, or
- `T` is a leaf node, or
- `T` is a nonleaf node, the value in `T` is smaller than the values in its children's nodes and its children are minheaps.

In writing `IsHeapOrdered`, you may call function `MinChild` specified in part (a). Assume that `MinChild` works as specified, regardless of what you wrote in part (a).

Complete function `IsHeapOrdered` below.

```
bool IsHeapOrdered(HeapNode * T)
{
    if (T == NULL)
    {
        return true;
    }
    if ((T->left == NULL) && (T->right == NULL))
    {
        return true;
    }
    if ((T->left->val > T->val) &&
        (T->right->val > T->val))
    {
        return true;
    }
    else
        return false;
}
```

Complete function RemoveMin below.

```
void RemoveMin(HeapNode * & T)  
// precondition: T is not NULL
```

```
{  
    if (T == NULL) return;  
    T = Minchild(T);  
    RemoveMin(T);  
    delete T;  
}
```