



## 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 a new implementation for private member function `AddSigDigit` to reflect the change to the linked list implementation of `BigInt`.

Complete function `AddSigDigit` below.

```
void BigInt::AddSigDigit(int value)
// postcondition: value added to BigInt as most significant digit
{
    DigitNode *temp;
    temp = new DigitNode;
    temp->digit = value + '0'; // needed for ASCII char conversion from int
    temp->next = myDigits;
    myDigits = temp;
    myNumDigits++;
}
```

Part (b) begins on page 16.

- (b) Write a new implementation for private member function `GetDigit()` to reflect the change to the linked list implementation of `BigInt`. For this function only, if the precondition is false then `GetDigit` returns zero.

Complete function `GetDigit` below.

```

int BigInt::GetDigit(int k) const
// precondition: 0 ≤ k < NumDigits()
// postcondition: returns k-th digit (0 if precondition is false)
// Note: GetDigit(0) returns the least significant digit
{
    if (0 ≤ k && k < NumDigits())
    {
        int j;
        j = myNumDigits - 1 - k;
        DigitNode * temp;
        temp = myDigits;
        for (int count = 1; count ≤ j; count++) // counts nodes until |
            // kth node is reached.
            temp = temp -> next;
        return temp -> digit - '0'; // needed for int from ASCII char conversion
    }
    return 0;
}

```

- (c) Write the destructor for the `BigInt` class with the new linked list implementation. The destructor deletes all nodes and returns them to memory.

Complete the destructor below.

```
BigInt::~BigInt()
// postcondition: all the nodes in myDigits are returned to the heap
{
    DigitNode * pos, temp;
    pos = myDigits;
    while (pos != NULL)
    {
        temp = pos;           // goes node by node to delete each
        pos = pos->next;
        delete temp;
    }
}
```

- (a) Write a new implementation for private member function `AddSigDigit` to reflect the change to the linked list implementation of `BigInt`.

Complete function `AddSigDigit` below.

```
void BigInt::AddSigDigit(int value)
// postcondition: value added to BigInt as most significant digit
```

```
{
```

```
    DigitNode * temp;
```

```
    temp = new DigitNode ('0'+value, myDigits);
```

```
    myDigits = temp;
```

```
}
```

Part (b) begins on page 16.

- (b) Write a new implementation for private member function `GetDigit()` to reflect the change to the linked list implementation of `BigInt`. For this function only, if the precondition is false then `GetDigit` returns zero.

Complete function `GetDigit` below.

```
int BigInt::GetDigit(int k) const
// precondition: 0 ≤ k < NumDigits()
// postcondition: returns k-th digit (0 if precondition is false)
// Note: GetDigit(0) returns the least significant digit
DigitNode * temp = myDigits;
if (k > NumDigits())
    return 0;
for (int n = 0; n < k - 1; n++)
    temp = temp->next;
return temp->digit;
```



- (c) Write the destructor for the `BigInt` class with the new linked list implementation. The destructor deletes all nodes and returns them to memory.

Complete the destructor below.

```

BigInt::~BigInt()
// postcondition: all the nodes in myDigits are returned to the heap
{
    DigitNode *temp = myDigits;
    while (temp != NULL)
        myDigits = myDigits->next;
        delete temp;
        temp = myDigits;
}
}

```

- (a) Write a new implementation for private member function `AddSigDigit` to reflect the change to the linked list implementation of `BigInt`.

Complete function `AddSigDigit` below.

```
void BigInt::AddSigDigit(int value)
// postcondition: value added to BigInt as most significant digit
```

```
{ char temp;

  temp = myDigits->digit;
  * myDigits = char(value);
  while (myDigits != NULL)
  { * myDigits->next = temp;
    myDigits->next = myDigits;
    temp = myDigits->digit;
  }
}
```

Part (b) begins on page 16.

- (b) Write a new implementation for private member function `GetDigit()` to reflect the change to the linked list implementation of `BigInt`. For this function only, if the precondition is false then `GetDigit` returns zero.

Complete function `GetDigit` below.

```
int BigInt::GetDigit(int k) const
// precondition: 0 ≤ k < NumDigits()
// postcondition: returns k-th digit (0 if precondition is false)
// Note: GetDigit(0) returns the least significant digit
```

```
{ int cnt=0, ret;
  if (0 ≤ k < NumDigits())
  { return 0; } }
```

```
while ((cnt != k) && (myDigits != NULL))
```

```
{ ret = myDigits → digit;
```

```
  myDigits = myDigits → next;
```

```
  cnt++;
```

```
}
```

```
return ret;
```

```
}
```

- (c) Write the destructor for the `BigInt` class with the new linked list implementation. The destructor deletes all nodes and returns them to memory.

Complete the destructor below.

```
BigInt::~BigInt()  
// postcondition: all the nodes in myDigits are returned to the heap
```

```
{ while (myDigits != NULL)  
  { delete myDigits;  
    myDigits = myDigits -> next;  
  }  
}
```