20

Programming Style

20.1 The operations depend on the class of the node. For classes with an attribute that contributes to a total, the operation *total()* is appropriate. For geometrical classes, the operations *move(offset)* and *rotate(angle)* could be applied.

We did not say in the problem statement how to pass any arguments of the operations. This could be done via a global variable or by passing arguments in a data structure. We will assume the latter, and have modified *orderedVisit* accordingly. The following is pseudocode for one way for performing *orderedVisit*:

```
orderedVisit (node, method, arguments)
{
   if node is not NULL
   {
      orderedVisit (node.leftSubtree, method, arguments);
      apply method (arguments) to node;
      orderedVisit (node.rightSubtree, method, arguments);
   }
}
```

20.2 The first operation is a special case of the second operation. The distinction between the two types of accounts can be eliminated by treating a normal account as a reserve account with a reserve limit of zero. Attributes needed to track accounts include *balance*, *reserveBalance*, and *reserveLimit*.

20.3a. This is an example of poor programming style. The assumption that the arguments are legal and the functions called are well behaved will cause trouble during program test and integration.

The following statements will cause the program to crash if the argument to *strlen* is zero:

```
rootLength = strlen(rootName);
suffixLength = strlen(suffix);
```

The following statement will assign zero to *sheetName* if the program runs out of memory, causing a program crash during the call to *strcpy* later in the function:

```
sheetName = malloc(rootLength + suffixLength + 1);
```

The following statements will cause the program to crash if any of the arguments are zero:

```
sheetName = strcpy(sheetName, rootName);
sheetName = strcat(sheetName, suffix);
```

If *sheetType* is invalid the switch statement will fall through leaving *sheet* without an assigned value. Also, it is possible that the call to *vertSheetNew* or the call to *horizSheetNew* could return zero for some reason. Either condition would make it possible for the following statement to crash:

```
sheet->name = sheetName;
```

b. The revised function is given below. The function *reportError(message)* reports errors.

```
Sheet createSheet (sheetType, rootName, suffix)
SheetType sheetType;
char *rootName, *suffix;
{   char *malloc(), *strcpy(), *strcat(), *sheetName;
    int strlen(), rootLength, suffixLength;
    Sheet sheet, vertSheetNew(), horizSheetNew();
    if (rootName)
        rootLength = strlen(rootName);
    else
    {
        rootName = "";
        rootLength = 0;
    }
    if (suffix)
        suffixLength = strlen(suffix);
    else
    {
        suffix = "";
        suffixLength = 0;
    }
    sheetName = malloc(rootLength + suffixLength + 1);
    if (sheetName)
```

```
      {
         sheetName = strcpy(sheetName, rootName);
         sheetName = strcat(sheetName, suffix);
      }
      else
      {
         sheetName = "";
         reportError("Out of memory.");
      }
      switch (sheetType)
      {  case VERTICAL:
            sheet = vertSheetNew();
            break;
         case HORIZONTAL:
            sheet = horizSheetNew();
            break;
         default:
            sheet = vertSheetNew();
            break;
      }
      if (sheet)
         sheet->name = sheetName;
      else
         reportError("Failure to create sheet.");
      return sheet;
   }
```

The revised function will not crash under error conditions. If a sheet is not created, zero is returned, and the calling routine must properly handle the failure.