Raspored i stil koda

Vanja Cvetković

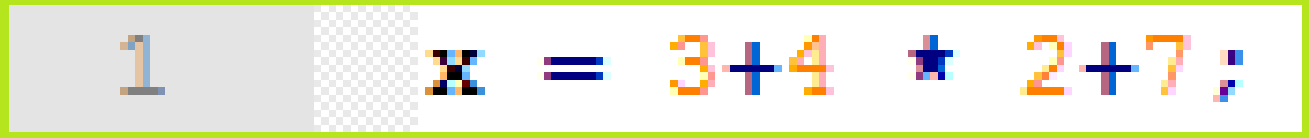
```
/* Use the insertion sort technique to sort the "data" array in ascending order.
This routine assumes that data[ firstElement ] is not the first element in data and
that data[ firstElement-1 ] can be accessed. */ public void InsertionSort( int[]
data, int firstElement, int lastElement ) { /* Replace element at lower boundary
with an element guaranteed to be first in a sorted list. */ int lowerBoundary =
data[ firstElement-1 ]; data[ firstElement-1 ] = SORT_MIN; /* The elements in
positions firstElement through sortBoundary-1 are always sorted. In each pass
through the loop, sortBoundary is increased, and the element at the position of the
new sortBoundary probably isn't in its sorted place in the array, so it's inserted
into the proper place somewhere between firstElement and sortBoundary. */ for (
int sortBoundary = firstElement+1; sortBoundary <= lastElement; sortBoundary++ )
{ int insertVal = data[ sortBoundary ]; int insertPos = sortBoundary; while (
insertVal < data[ insertPos-1 ] ) { data[ insertPos ] = data[ insertPos-1 ];
insertPos = insertPos-1; } data[ insertPos ] = insertVal; } /* Replace original
lower-boundary element */ data[ firstElement-1 ] = lowerBoundary; }
```

```java
/* Use the insertion sort technique to sort the "data" array in ascending
order. This routine assumes that data[ firstElement ] is not the
first element in data and that data[ firstElement-1 ] can be accessed.
*/

public void InsertionSort( int[] data, int firstElement, int lastElement ) {
    // Replace element at lower boundary with an element guaranteed to be
    // first in a sorted list.
    int lowerBoundary = data[ firstElement-1 ];
    data[ firstElement-1 ] = SORT_MIN;

    /* The elements in positions firstElement through sortBoundary-1 are
    always sorted. In each pass through the loop, sortBoundary
    is increased, and the element at the position of the
    new sortBoundary probably isn't in its sorted place in the
    array, so it's inserted into the proper place somewhere
    between firstElement and sortBoundary.
    */
    for ( int sortBoundary = firstElement + 1; sortBoundary <= lastElement;
        sortBoundary++ ) {
        int insertVal = data[ sortBoundary ];
        int insertPos = sortBoundary;
        while ( insertVal < data[ insertPos - 1 ] ) {
            data[ insertPos ] = data[ insertPos - 1 ];
            insertPos = insertPos - 1;
        }
        data[ insertPos ] = insertVal;
    }

    // Replace original lower-boundary element
    data[ firstElement - 1 ] = lowerBoundary;
}
```

Osnovna teorema formatiranja

“Dobar vizualni raspored pokazuje logičku strukturu programa.”

```
1   // swap left and right elements for whole array
2   for ( i = 0; i < MAX_ELEMENTS; i++ )
3       leftElement = left[ i ];
4       left[ i ] = right[ i ];
5       right[ i ] = leftElement;
```


1
x = 3+4 * 2+7;

```
1  if (0 == i)
2      FooBar();
3
4  if (i == 0)
5      FooBar();
```

Ciljevi dobrog rasporeda

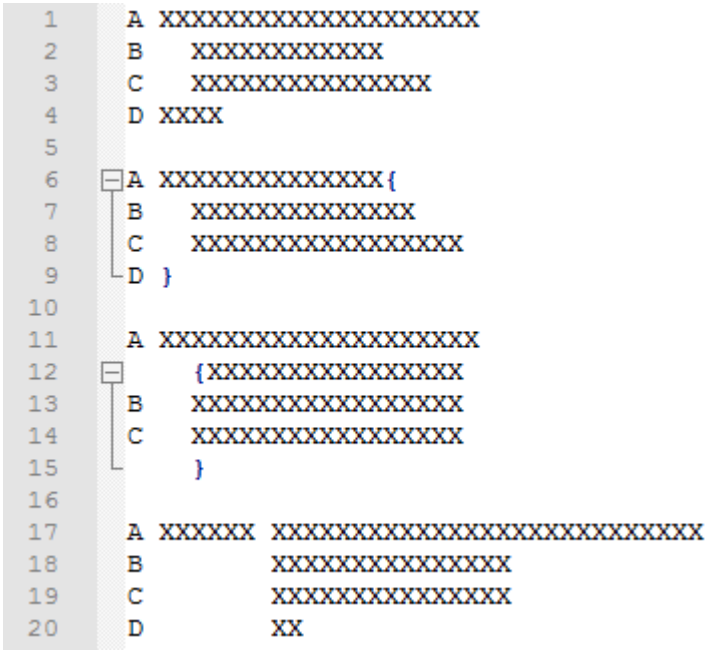
- Tačno predstaviti logičku strukturu koda
- Konzistentno predstaviti logičku strukturu koda
- Unaprediti čitljivost
- Izdržati modifikacije

Alati za uređenje rasporeda

- Beline
  - Razmaci
  - Tabulatori
  - Prazni redovi
- Zagrade

Stilovi raspoređivanja koda

- Čisti blokovi
- Emuliranje čistih blokova
- Korišćenje begin-end
- parova
- Raspoređivanje po
- krajnjoj liniji



```
If ( soldCount > 10 And prevMonthSales > 10 ) Then
   If ( soldCount > 100 And prevMonthSales > 10 ) Then
      If ( soldCount > 1000 ) Then
                                 markdown = 0.1
                                 profit = 0.05
                               Else
                                 markdown = 0.05
                               End If
                                                    Else
                                                      markdown = 0.025
                                                    End If
                                                  Else
                                                    markdown = 0.0
                                                  End If
```

```
cursor.start = startingScanLine;
cursor.end   = endingScanLine;
window.title = editWindow.title;
window.dimensions      = editWindow.dimensions;
window.foregroundColor = userPreferences.foregroundColor;
cursor.blinkRate       = editMode.blinkRate;
window.backgroundColor = userPreferences.backgroundColor;
SaveCursor( cursor );
SetCursor( cursor );
```


```
window.dimensions = editWindow.dimensions;
window.title = editWindow.title;
window.backgroundColor = userPreferences.backgroundColor;
window.foregroundColor = userPreferences.foregroundColor;

cursor.start = startingScanLine;
cursor.end = endingScanLine;
cursor.blinkRate = editMode.blinkRate;
SaveCursor( cursor );
SetCursor( cursor );
```

```
1   if ( expression )
2       one-statement;
3
4   if ( expression ) {
5       one-statement;
6   }
7
8   if ( expression )
9       {
10      one-statement;
11      }
12
13  if ( expression ) one-statement;
```

```
if ((('0' <= inChar) && (inChar <= '9')) || (('a' <= inChar) &&
   (inChar <= 'z')) || (('A' <= inChar) && (inChar <= 'Z')))
     ...

if ( ( ( '0' <= inChar ) && ( inChar <= '9' ) ) ||
   ( ( 'a' <= inChar ) && ( inChar <= 'z' ) ) ||
   ( ( 'A' <= inChar ) && ( inChar <= 'Z' ) ) )
     ...
```

```
void PurgeFiles( ErrorCode & errorCode ) {
    FileList fileList;
    int numFilesToPurge = 0;
    MakePurgeFileList( fileList, numFilesToPurge );
    errorCode = FileError_Success;
    int fileIndex = 0;
    while ( fileIndex < numFilesToPurge ) {
        DataFile fileToPurge;
        if ( !FindFile( fileList[ fileIndex ], fileToPurge ) ) {
            errorCode = FileError_NotFound;
            goto END_PROC;
        }
        if ( !OpenFile( fileToPurge ) ) {
            errorCode = FileError_NotOpen;
            goto END_PROC;
        }

        if ( !OverwriteFile( fileToPurge ) ) {
            errorCode = FileError_CantOverwrite;
            goto END_PROC;
        }

        if ( !Erase( fileToPurge ) ) {
            errorCode = FileError_CantErase;
            goto END_PROC;
        }
        fileIndex++;
    }

END_PROC:
    DeletePurgeFileList( fileList, numFilesToPurge );
}
```

```
switch ( ballColor ) {
    case BallColor_Blue:              Rollout();
                                      break;
    case BallColor_Orange:            SpinOnFinger();
                                      break;
    case BallColor_FluorescentGreen: Spike();
                                      break;
    case BallColor_White:             KnockCoverOff();
                                      break;
    case BallColor_WhiteAndBlue:      if ( mainColor == BallColor_White ) {
                                          KnockCoverOff();
                                      }
                                      else if ( mainColor == BallColor_Blue ) {
                                          RollOut();
                                      }
                                      break;
    default:                          FatalError( "Unrecognized kind of ball." );
                                      break;
}
```


```
switch ( ballColor ) {
    case BallColor_Blue:
        Rollout();
        break;
    case BallColor_Orange:
        SpinOnFinger();
        break;
    case BallColor_FluorescentGreen:
        Spike();
        break;
    case BallColor_White:
        KnockCoverOff();
        break;
    case BallColor_WhiteAndBlue:
        if ( mainColor == BallColor_White ) {
            KnockCoverOff();
        }
        else if ( mainColor == BallColor_Blue ) {
            RollOut();
        }
        break;
    default:
        FatalError( "Unrecognized kind of ball." );
        break;
}
```

```
while(pathName[startPath+position]<>';') and
   ((startPath+position)<length(pathName)) do

while ( pathName[ startPath+position ] <> ';' ) and
   (( startPath + position ) < length( pathName )) do

while ( pathName[ startPath + position ] <> ';' ) and
   ( ( startPath + position ) < length( pathName ) ) do
```

```
ReadEmployeeData(maxEmps,empData,inputFile,empCount,inputError);

GetCensus( inputFile, empCount, empData, maxEmps, inputError );
```

```
1  strcpy( char * t, char * s ) {
2      while ( *++t = *++s )
3          ;
4  }
```

?

```
1  strcpy( char * t, char * s ) {
2      do {
3          ++t;
4          ++s;
5          *t = *s;
6      }
7      while ( *t != '\0' );
8  }
```

```
1   int rowIndex, columnIdx; Color previousColor, currentColor, nextColor; Point  previousTop, previousBottom,
     currentTop, currentBottom, nextTop, nextBottom; Font  previousTypeface, currentTypeface, nextTypeface;
    Color choices[ NUM_COLORS ];
2
3   int rowIndex, columnIdx;
4   Color previousColor, currentColor, nextColor;  Point previousTop, previousBottom, currentTop,
    currentBottom, nextTop,  nextBottom;
5   Font previousTypeface, currentTypeface, nextTypeface;  Color choices[ NUM_COLORS ];
6
7   int rowIndex;
8   int columnIdx;
9   Color previousColor;
10  Color currentColor;
11  Color nextColor;
12  Point previousTop;
13  Point previousBottom;
14  Point currentTop;
15  Point currentBottom;
16  Point nextTop;
17  Point nextBottom;
18  Font previousTypeface;
19  Font currentTypeface;
20  Font nextTypeface;
21  Color choices[ NUM_COLORS ];
```

```
1  bool ReadEmployeeData( int                maxEmployees,
2                         EmployeeList       *employees,
3                         EmployeeFile       *inputFile,
4                         int                *employeeCount,
5                         bool               *isInputError )
6  ...
7  void InsertionSort( SortArray   data,
8                      int         firstElement,
9                      int         lastElement )
```

```
1  public bool ReadEmployeeData(
2      int maxEmployees,
3      EmployeeList *employees,
4      EmployeeFile *inputFile,
5      int *employeeCount,
6      bool *isInputError
7  )
8  ...
```

Raspoređivanje klasnih interfejsa

- Komentar na vrhu koji opisuje kako se klasa koristi
- Konstruktori i destruktori
- Javne metode
- Zaštićene meode
- Privatne metode i članice klase

Raspoređivanje unutar implementacije klase

- Komentar na vrhu koji opisuje sadržinu datoteke u kojoj se klasa nalazi
- Klasne podatke
- Javne metode
- Zaštićene metode
- Privatne metode

```
1   //*************************************************************
2   //*************************************************************
3   // MATHEMATICAL FUNCTIONS
4   //
5   // This class contains the program's mathematical functions.
6   //*************************************************************
7   //*************************************************************
8
9   //*************************************************************
10  // find the arithmetic maximum of arg1 and arg2
11  //*************************************************************
12  int Math::Max( int arg1, int arg2 ) {
13  //*************************************************************
14     if ( arg1 > arg2 ) {
15        return arg1;
16     }
17     else {
18        return arg2;
19     }
20  }
21
22  //*************************************************************
23  // find the arithmetic minimum of arg1 and arg2
24  //*************************************************************
25  int Math::Min( int arg1, int arg2 ) {
26  //*************************************************************
27     if ( arg1 < arg2 ) {
28        return arg1;
29     }
30     else {
31        return arg2;
32     }
```

```
1  //*************************************************************************
2  // MATHEMATICAL FUNCTIONS
3  //
4  // This class contains the program's mathematical functions.
5  //*************************************************************************
6
7  //-------------------------------------------------------------------------
8  // find the arithmetic maximum of arg1 and arg2
9  //-------------------------------------------------------------------------
10 int Math::Max( int arg1, int arg2 ) {
11    if ( arg1 > arg2 ) {
12       return arg1;
13    }
14    else {
15       return arg2;
16    }
17 }
18
19 //-------------------------------------------------------------------------
20 // find the arithmetic minimum of arg1 and arg2
21 //-------------------------------------------------------------------------
22 int Math::Min( int arg1, int arg2 ) {
23    if ( arg1 < arg2 ) {
24       return arg1;
25    }
26    else {
27       return arg2;
28    }
29 }
```

Pitanja?

Osnovna teorema formatiranja

"Dobar vizualni raspored pokazuje logičku strukturu programa."

Hvala na pažnji