Process Activities

Implementation

Introduction

- The implementation phase is the process of converting a system specification into an executable system.
- If an incremental approach is used, it may also involve refinement of the software specification.
- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process.

Coding Style

- They way you write your code.
- It is all about consistency and make your code readable for others.
- Certain rules are follow to make your code style good.

Good Coding Style

Elements of Good Coding Style,

- Naming convention
- File names
- Indentation
- Placement of braces
- Use of whitespace

Naming convention

- Variable vs routine

variable_name, routineName(), RoutineName()

- Class vs object

Widge, my_widget

- Member variables

variable or variable

File name

- Common to use _ or – between words, don’t use space.
- Windows allow space between words but not in Linux or MAC.
- Even extension could need specification.

Linked-list.cpp, Linked_list.cpp, linkedlist.cpp

Linked list.cpp

Indentation

- A timeless battle: tabs versus spaces
- Important is indent to show nesting level and be consistent.

```
class Test
{
  int a;
  void func(int a)
   {
     print(a)
    }
}
```

```
class Test
{
    int a;
      void func(int a)
      {
           print(a);
       }
}
```

```
class Test
 {
int a;
   void func(int a)
  {
  print(a);
   }
   }
```

Placement of Braces

1. K&R

```
void CheckNeg(int x)
{
   if (x<0){
     negative(x)
   } else
     nonnegative(x)
}
```

1. 1TBS

```
void CheckNeg(int x){
   if (x<0){
     negative(x)
   } else {
     nonnegative(x)
   }
}
```

1. Allman

```
void CheckNeg(int x)
{
   if (x<0)
   {
     negative(x)
   }
   else
   {
     nonnegative(x)
   }
}
```

Use of white spaces

- Can make a real difference in understanding,

double *a;

double* a;

double * a;

Example of understanding difficulty,

double* a,b; **VS** **double *a,b;**

Examples

Example 1:

```
X = 3+4 * 7+5;
```

Example 1:

$$X = 3+4 * 7+5;$$

Seems like,

3+4= 7

7+5 = 12

7*12 =84

X=84

Actually,

4*7 = 28

3+28=31

31+5 =36

X=36

Example #2

```
/* 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 ; }
```

Example #2 Revised

```
/* 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 ; }
```

Example #2 Revised Again

```
/* 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 ;
}
```

Example #3

```
FUNCTION comppoly (x)
   float y1 , y2
   float a1 =0.1 , b1 =0.3 , a2 =2.1 , b2 =5.3, c =0.22
   y1 = a1*x + b1
   y2 = a1*x^2 + b2*x + c
   return (y2 >y1)
END FUNCTION
```

Example #3 Self-Documenting

```
FUNCTION ComparePolynomials (x)
  // DECLARE VARIABLES , PARAMETERS
  float y_line , y_quadratic
  float line_param = [0.1 , 0.3]
  float quad_param = [2.1 , 5.3 , 0.22]
  // CALCULATE THE LINE AND QUADRATIC VALUES AT X
  y_line = line_param[0]*x + line_param[1]
  y_quadratic = quad_param[0]*x^2 + quad_param[1]*x + quad_param[2]
  // COMPARE THE FUNCTIONS , RETURNING A LOGICAL
  return ( y_line > y_quadratic )
END FUNCTION
```

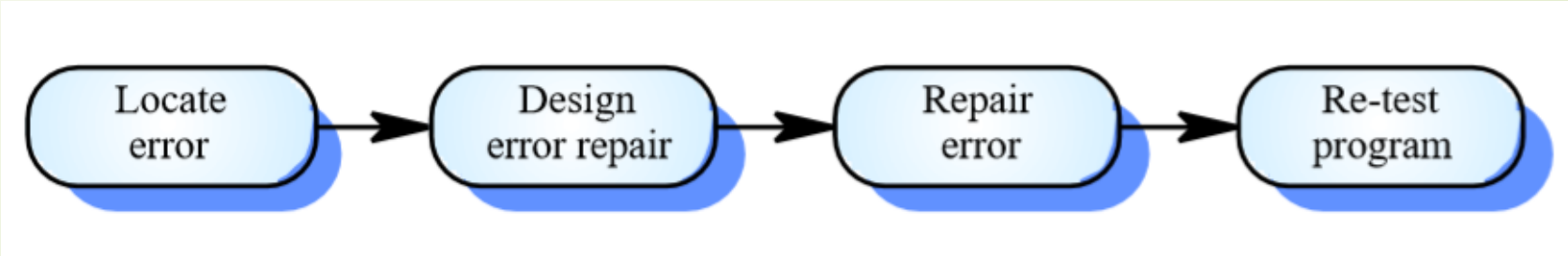
Debugging

- The art of removing defects.

Two forms:

- Print statements
- Debugging tools

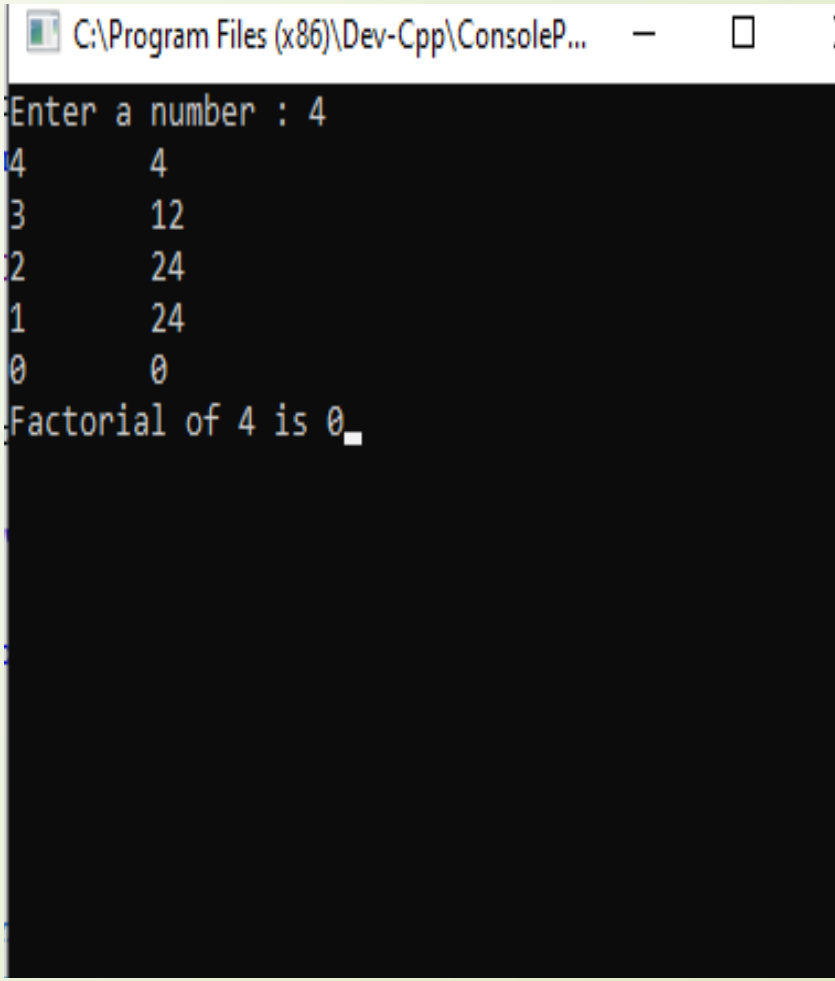
- Error Messages: Its difficult to read error messages prompt by compiler.



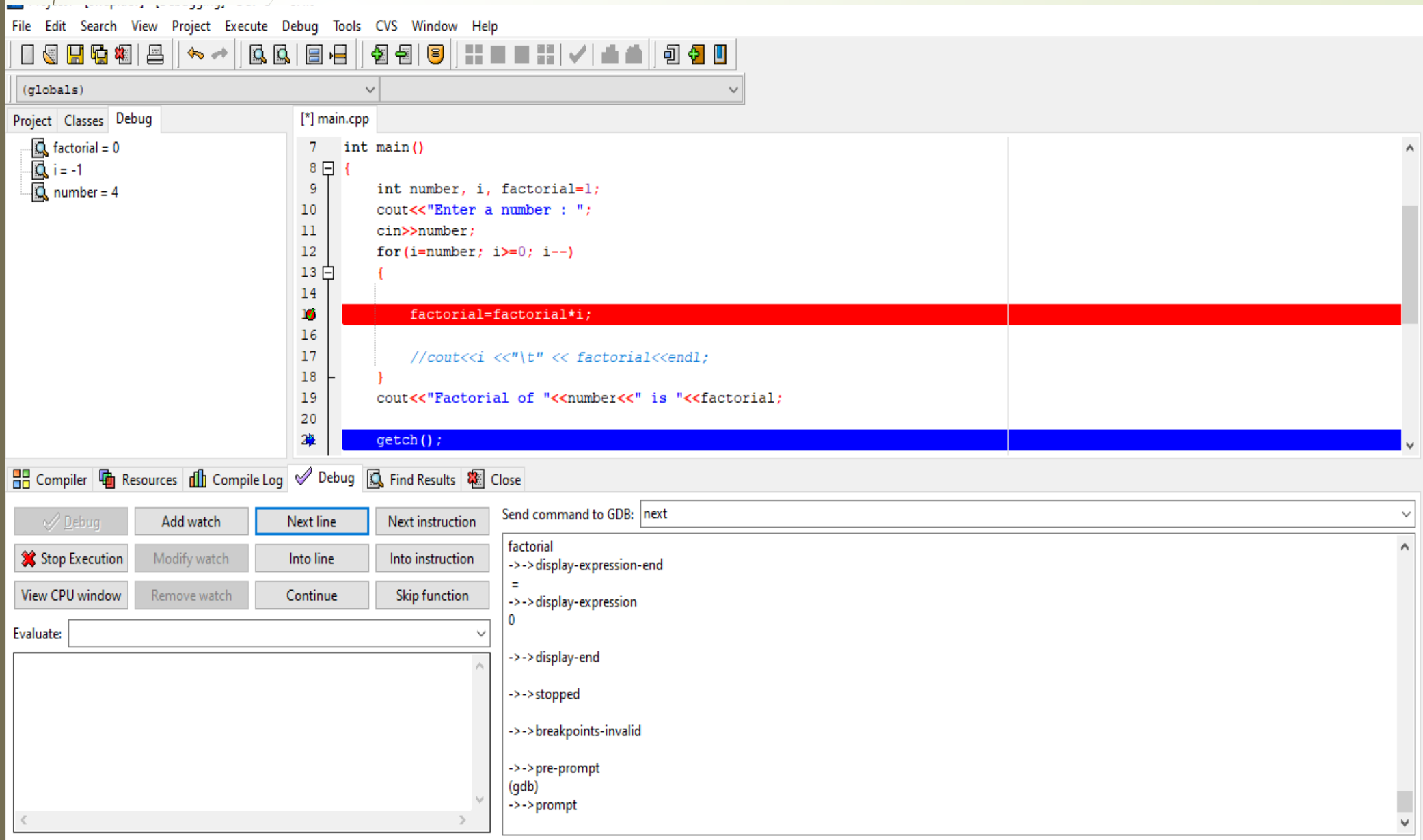
Example (Print Statement)

main.cpp

```
5   #include<conio.h>
6   using namespace std;
7   int main()
8   {
9       int number, i, factorial=1;
10      cout<<"Enter a number : ";
11      cin>>number;
12      for(i=number; i>=0; i--)
13      {
14
15          factorial=factorial*i;
16
17          cout<<i <<"\t" << factorial<<endl;
18      }
19      cout<<"Factorial of "<<number<<" is "<<factorial;
```



Example Debugging



In-code Documentation

- Comment
- Self-Documenting Code

Example 1

```
1 current = 1;
2 previous = 0;
3 sum = 1;
4 for ( int i = 0; i < num; i++ ) {
5    System.out.println( "Sum = " + sum );
6    sum = current + previous;
7    previous = current;
8    current = sum;
9 }
```

Example 1 (Comment)

```
1  // write out the sums 1..n for all n from 1 to num
2  current = 1;
3  previous = 0;
4  sum = 1;
5  for ( int i = 0; i < num; i++ ) {
6      System.out.println( "Sum = " + sum );
7      sum = current + previous;
8      previous = current;
9      current = sum;
10 }
```

Example 2

```
1 x = b;
2 for ( int i = 2; i <= num; i++ ) {
3    x = x * b;
4 }
5 System.out.println( "Result = " + x );
```

Example 2 (Self Documenting Code)

```
1 product = base;
2 for ( int i = 2; i <= num; i++ ) {
3     product = product * base;
4 }
5 System.out.println( "Product = " + product );
```

Example 3: Bad comments

```
1 // set product to "base"
2 product = base;
3 // loop from 2 to "num"
4 for ( int i = 2; i <= num; i++ ) {
5     // multiply "base" by "product"
6     product = product * base;
7 }
8 System.out.println( "Product = " + product );
```

Example 4

```
1 r = num / 2;
2 while ( abs( r - (num/r) ) > TOLERANCE ) {
3     r = 0.5 * ( r + (num/r) );
4 }
5 System.out.println( "r = " + r );
```

Example 4: One line Comment

```
1 // compute the square root of Num using the
2 // Newton-Raphson approximation
3 r = num / 2;
4 while ( abs( r - (num/r) ) > TOLERANCE ) {
5     r = 0.5 * ( r + (num/r) );
6 }
7 System.out.println( "r = " + r );
```

Example 5

```
1  for ( i = 2; i <= num; i++ ) {
2    meetsCriteria [ i ] = true ;
3  }
4  for ( i = 2; i <= num / 2; i++ ) {
5    j = i + i;
6    while ( j <= num ) {
7      meetsCriteria [ j ] = false ;
8      j = j + i;
9    }
10 }
11 for ( i = 1; i <= num; i++ ) {
12   if ( meetsCriteria [ i ] ) {
13     System .out. println ( i + "  meets   criteria ." );
14   }
15 }
```

Example 5 : Self Documenting Code

```
1  for ( primeCandidate = 2; primeCandidate <= num; primeCandidate++)
2    isPrime [ primeCandidate ] = true ;
3  }
4  for ( int factor = 2; factor < ( num / 2 ); factor ++ ) {
5    int factorableNumber = factor + factor ;
6    while ( factorableNumber <= num ) {
7      isPrime [ factorableNumber ] = false ;
8      factorableNumber = factorableNumber + factor ;
9    }
10 }
11 for ( primeCandidate = 1; primeCandidate <= num; primeCandidate++)
12   if ( isPrime [ primeCandidate ] ) {
13     System .out. println ( primeCandidate + ’’ is prime .’’ );
```

Example 6: In line comment

```
1 For rateIdx = 1 to rateCount                    ' Compute discounted rates
2     LookupRegularRate( rateIdx, regularRate )
3     rate( rateIdx ) = regularRate * discount( rateIdx )
4 Next
5
6
7 int boundary = 0;          // upper index of sorted part of array
8 String insertVal = BLANK; // data elmt to insert in sorted part of array
9 int insertPos = 0;         // position to insert elmt in sorted part of array
```

Documenting in code

Documenting is key to understanding, now and lat

Self-documenting code get updated dynamically

Comments provide context that code cannot

Both forms are likely necessary

Software Industry – Code Implementation Best Practices

- Commenting & Documentation properly with in the code.
- Consistent Indentation styles are not always completely distinct from one another. Sometimes, they mix different rules. For example, in PEAR Coding Standards.
- Avoid Obvious Comments means When the text is that obvious, it's really not productive to repeat it within comments.
- Always following the Code Grouping certain tasks require a few lines of code then It is a good idea to keep these tasks within separate blocks of code, with some spaces between them.
- Always used Consistent Naming Scheme and it should be word boundaries either Pascal Case or Camel Case.
- Always follow the DRY Principle that is Don't Repeat Yourself, also known as DIE: Duplication is Evil

Software Industry – Code Implementation Best Practices

- Always Avoid Deep Nesting within your code because too many levels of nesting can make code harder to read and follow.
- It is good practice that follow the Limit Line Length concept to avoid writing horizontally long lines of code.
- File and Folder Organization is the best approaches is to either use a framework or manage project folder structure.
- Always capitalize SQL Special Words and functions capitalize them to distinguish them from your table and column names.
- Always use Separation of Code and Data because things have changed over the years and websites became more and more dynamic and functional.
- Continuously do Code Refactoring so that we can make changes to the code without changing any of its functionality therefore it is a " clean up" process for the sake of improving readability and quality