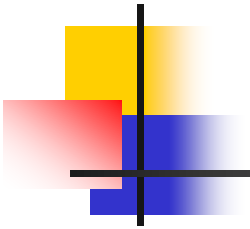
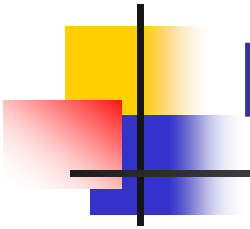
Software Development and Exception Handling



- Software Development
  - Phases of Software Development
    - Specification, Design, Implementation, Analysis, Testing, Maintenance, Obsolescence
- Handling Runtime Errors
  - Exceptions



Phases of Software Development

- Specification of the task
  - Precise description of problem
- Design of a solution
  - Formulate steps to solve problem (algorithm)
    - Break down large problem into smaller sub-problems
    - Identify common tasks
    - Create independent functions
      - Procedural abstraction with preconditions and postconditions

Design of a Solution

- Precondition → statement giving condition required to be true when function is called
  - Function not guaranteed to perform correctly unless precondition is true
  - Programmer who *calls* function is responsible for ensuring precondition is valid
- Postcondition → statement describing what is true when function completes
  - Programmer who *writes* function ensures that postcondition is true at end of function

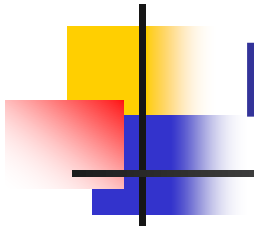
Phases of Software Development

- Pseudocode/flowchart documents design
- Implementation (coding) of the solution
  - Create code to carry out design
  - Function prototype
    - Return type
    - Function name
    - Parameter list
  - Declared constants

```
double calc_sqrt(double x);
// Precondition: x >= 0
// Postcondition: Square root of x is
// returned.
```

```
const int MIN_VAL = 0;
```

Phases of Software Development



- Assert to check precondition
  - #include <cassert>
  - Boolean assertion argument
    - If true, no action
    - If false, display error message and halt program
  - Turn off assertion checks
    - #define NDEBUG
- Program return values
  - Defined in <cstdlib>
  - EXIT_SUCCESS or EXIT_FAILURE

```
assert(x >= MIN_VAL);
```

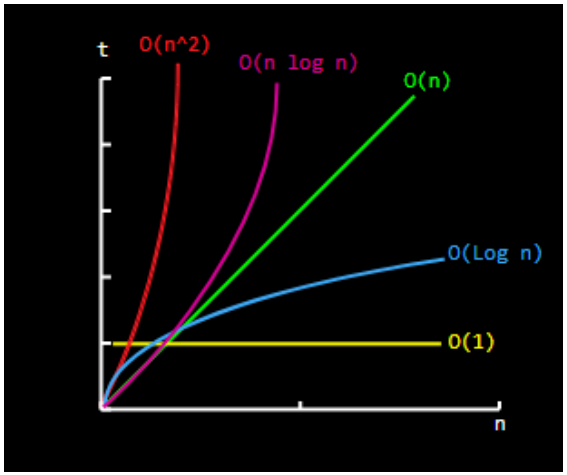
```
return EXIT_SUCCESS;
```

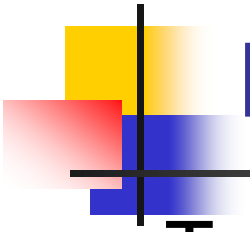
Phases of Software Development

- Analysis of the solution
  - Reason about algorithm's speed as input gets larger
    - Big-O notation

FIGURE 1.3 Number of Operations for Three Methods

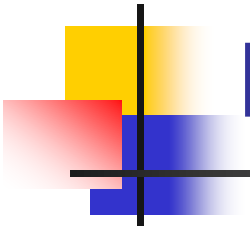
Number of stairs (n)	Logarithmic $O(\log n)$ Method 3, with $\lfloor \log_{10} n \rfloor + 1$ operations	Linear $O(n)$ Method 1, with $3n$ operations	Quadratic $O(n^2)$ Method 2, with $n^2 + 2n$ operations
10	2	30	120
100	3	300	10,200
1000	4	3000	1,002,000
10,000	5	30,000	100,020,000





Phases of Software Development

- Testing and debugging
  - Choosing test data
    - Correct data
    - Incorrect data
      - Boundary value → one step away from different kind of behavior
  - Fully exercise code with profiler
    - Execute each line of code *at least* once
    - Test code that may be skipped altogether
  - Find errors in program using debugger



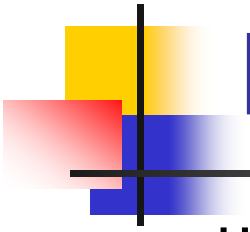
Phases of Software Development

- Maintenance and evolution of the system
  - Bug fixes and/or enhancements
- Obsolescence
  - Software costs exceed benefits
  - Decreased utilization

Handling Runtime Errors

- Good program design anticipates errors and attempts to perform some type of corrective action
  - runtime errors
- Three fundamental ways to handle errors:
  - use exit() function from <cstdlib>

    ```
    void exit(int);
    ```
  - use return value or boolean flag and test
  - use exception handling
    - place code under inspection in **try** block
    - **throw** exception from inside try block (throw point)
    - **catch** block (exception handler) follows **try** block
      - parameter same as type of thrown exception



Handling Runtime Errors

- use exception handling
  - exception moves in reverse order through chain of function calls until an exception handler is found that will catch the exception
  - try-catch blocks can be nested; internal throw with no parameter passes exception to external try-catch block
  - exception can go uncaught
    - no catch blocks with exception parameter of right type
      - can use a catch-all clause where catch parameter is an ellipse (...)
    - exception thrown outside of try block
  - if no exception is thrown, handlers ignored and code executes normally

Handling Runtime Errors

- example

```
double divide(int num, int den)
{
   if(den == 0)
      throw “ERROR: Cannot divide by zero.\n”;
   else
      return static_cast<double>(num)/den;
}
```

← exception is thrown

```
try
{
   quotient = divide(num1, num2);
   cout << “The quotient is “ << quotient << endl;
}
catch (char *exceptionString)
{
   cerr << exceptionString;
}
cout << “End of program.\n”;
```

code that calls function that might throw exception

code that handles exception (repeat as needed)

← program resumes after catch block

Exceptions

- Standard exceptions defined in <exception>
  - Base class exception
    - Derived
    - Logic errors
    - Runtime errors