MyOnlineEdu.com

A Place for eLearning & Education

MOVE FORWARD
YOUR CAREER

Programming Best Practices

By – Amalendu Kundu

Programming Best Practices

Topics

- Why Follow Standards?
- Coding Techniques
  - Readability
  - Tips & Styles
- Programming Practices
  - Maintainability
  - DRY Principal
- Conclusions and Goals

Why Follow Standard Practice?

Why Follow Standards?

Consider the below scenario

- You are part of a software development team of 5 people.
- All 5 developers are using their own standards while developing code.

After completion of development

- Team Lead
- Project Manager
- Client

Coding Techniques

Coding Technique - Readability

- Consistent Indentation
  - No matter how many *SPACE*'s you use for an indent, use it consistently throughout the source code. *SPACE*'s and *TAB* 's do not mix well!
  - Indent code to better convey the logical structure of your code. Without indenting, code becomes difficult to follow.

```
if ( per >= 50 )
printf ( "Second division" ) ;
else
{
if ( per >= 40 )
printf ( "Third division" ) ;
else
printf ( "Fail" ) ;
}
```

```
if ( per >= 50 )
printf ( "Second division" ) ;
else
  {
if ( per >= 40 )
      printf ( "Third division" ) ;
  else
  printf ( "Fail" ) ;
}
```

```
if ( per >= 50 )
  printf ( "Second division" ) ;
else
{
  if ( per >= 40 )
    printf ( "Third division" ) ;
  else
    printf ( "Fail" ) ;
}
```

Coding Technique – Readability (Contd…)

- Consistent Naming Scheme: two options
  - **camelCase**: First letter of each word is capitalized, except the first word.
    Example: calculateTotal or validateEmailAddress
  - **Underscores**: Underscores between words, like: display_matrix() or v_total_amount.

- Some developers prefer to use underscores for procedural functions, and class names, but use camelCase for class method names

- A name should tell what rather than how, avoid names that expose underlying implementation.

Coding Technique – Readability (Contd…)

- Break large, complex sections of code into smaller, comprehensible modules (subroutine/functions/methods).

- Use a verb-noun method to name routines that perform some operation-on-a-given-object. *Example*: calculateTotalAmount(), get_user_input() etc

Tips & Styles

Coding Technique – Tips & Styles

- Use *SPACE* instead of *TAB*. *TAB*'s appear differently in different IDE.
- Use 3-4 spaces for each indentation.
- Establish a *maximum line length* to avoid having to scroll the window of the text editor.
- Use *SPACE* after each "comma" in lists, such as array values and arguments, also before and after the "equal" of an assignment
  - centigrade = (fahrenheit – 32) / 9 * 5;
  - temp_flag = 10 * pow(x, y);

Coding Technique – Tips & Styles (Contd…)

- Avoid placing more than one statement per line.

- Choose and stick to a style for naming various elements of the code, this is one of the most influential aids to understand the logical flow.

```
function userLogin() {
    if ($isUser) {
        login();
        visit_home_page();
    } else {
        logout();
    }
    finalize();
}
```

```
function userLogin()
{
    if ($isUser)
    {
        login();
        visit_home_page();
    }
    else
    {
        logout();
    }
    finalize();
}
```

Coding Technique – Tips & Styles (Contd…)

- Put comment in your code. Comment helps others to understand what are you trying to do.

- Avoid obvious and unnecessary comments.

```
/* Function Name:  validate_email
   Purpose: To validate an given email address and returns either
            0 or 1
   Parameters: p_email_id is a pointer to the string of email_id
   Return Value: 0 when it is a valid email address.
                 1 when it is an invalid email address
*/
int valididate_email(char *p_email_id)
{
  …
}
```

```
/* Calling the function validate_email to check
   if the entered email_id is valid or not
*/

validate_email(email_id);
```

Coding Technique – Tips & Styles (Contd…)

- Append computation qualifiers (Avg, Sum, Min, Max, Index) to the end of a variable name where appropriate.

- Use customary opposite pairs in variable names, such as min/max, begin/end, and open/close.

- Boolean variable names should contain Is which implies Yes/No or True/False values, such as fileIsFound or isValidEmail or isPrimeNumber.

- Even for a short-lived variable that may appear in only a few lines of code, still use a meaningful name. Use single-letter variable names, such as i, or j, for short-loop indexes or array index only.

Coding Technique – Tips & Styles (Contd…)

- For variable names, it is sometimes useful to include notation that indicates the scope of the variable, such as prefixing a
  - g_ for global variables.
  - l_ for local variables.
  - p_ for input parameters
  - o_ for out parameters

- Constants should be all uppercase with underscores between words, such as NUM_DAYS_IN_WEEK. Also, begin groups of enumerated types with a common prefix, such as FONT_ARIAL and FONT_ROMAN.

Coding Technique – Tips & Styles (Contd…)

- Group your code which is meant for certain tasks.

```
//Initialize variables
prin = 1000;
rate = 10;
num_of_years = 5;

//Calculate interests
calculate_simple_int(prin, rate, num_of_years);
calculate_complex_int(prin, rate, num_of_years);
```

- Adding a comment at the beginning of each block of code also emphasizes the visual separation.

Programming Practices

Maintainability

- *DRY* Principal: Don't Repeat Yourself.
- Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.
- Modular Approach -- reusability
- Object Oriented Programming Approach

```
// Include the header, left bar and footer templates
require_once 'template/header_template.php';
require_once 'template/left_sidebar_template.php';
...
//Main page content
…
require_once 'template/footer_template.php';
```

Programming Practices – Maintainability (Contd…)

- Keep scope of variables as small as possible to avoid confusion and ensure maintainability.
- Use variables and routines for one purpose only. Avoid multipurpose routines that perform a variety of unrelated tasks.
- Keep in mind what control flow constructs do, for instance

```
if (marks < 30)
   printf("Fail");
else if (marks >= 30)
   printf("Passed");
```

```
if (marks < 30)
   printf("Fail");
else
   printf("Passed");
```

Programming Practices – Maintainability (Contd…)

- Avoid Deep Nesting.

- Too many levels of nesting can make code harder to read and follow.

```
if ( per >= 60 )
  printf ( "First division ") ;
else
{
  if ( per >= 50 )
    printf ( "Second division" ) ;
  else
  {
    if ( per >= 40 )
      printf ( "Third division" ) ;
    else
      printf ( "Fail" ) ;
  }
}
```

```
if ( per >= 60 )
  printf ( "First division" ) ;

if ( ( per >= 50 ) && ( per < 60 ) )
  printf ( "Second division" ) ;

if ( ( per >= 40 ) && ( per < 50 ) )
  printf ( "Third division" ) ;

if ( per < 40 )
  printf ( "Fail" ) ;
```

Programming Practices – Maintainability (Contd…)

- Don’t assume output formats. Functions should return values in original type, the caller should decide what to do, reformat, sent to standard output, etc. . .

- Try to keep return point from a function as the last statement in the function.

```
char get_marks_grade(int p_marks)
{
  if (p_marks >= 70)
    return ‘A’;
  if(p_marks < 70 && p_marks >= 50)
    return ‘B’;
  if(p_marks < 50 && p_marks >= 30)
    return ‘C’;

  return ‘F’;
}
```

```
char get_marks_grade(int p_marks)
{
  char ret_grade = ‘F’;
  if (p_marks >= 70)
    ret_grade = ‘A’;
  if(p_marks < 70 && p_marks >= 50)
    ret_grade = ‘B’;
  if(p_marks < 50 && p_marks >= 30)
    ret_grade = ‘C’;

  return ret_grade;
}
```

Programming Practices – Maintainability (Contd…)

- Similar to RETURN statement use *BREAK*, *CONTINUE*, *GOTO*, *EXIT* wisely.
- Recover or fail "gracefully". Robust programs should report an error message (and optimally attempt to continue).
- Provide useful error messages. Expanding on the previous point, you should provide a user-friendly error message while simultaneously logging a programmer-friendly message with enough information that support team can investigate the cause of the error.
- Use constant literals instead of numeric or character literals.

```
for (i = 1; i <= 7; i++)
{
  …
}
```

```
#define DAYS_IN_WEEK 7
for (i = 1; i <= DAYS_IN_WEEK; i++)
{
  …
}
```

Conclusion and Goals

Conclusion and Goals

- Make your code highly readable.
- Make you code highly maintainable.
- Make your code modular and follow DRY principal.
- Do code review with peers.
- Programming is not an exact science, but the more you practice, the more you develop skills.
- Using such "cooking recipes" and a bit of common sense with positive attitude should hopefully help you to develop your awareness for good practice.

Thank You!