```c
// Authored By: 1MS19IS051
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>

// Function to add elements to arrrays into another
double *vectAdd(double *c, double *a, double *b, int n){
    #pragma omp parallel for
    for(int i = 0; i < n; i++){
        c[i] = a[i] + b[i];
    }
    // Return the updated final array
    return c;
}
```

```c
// Authored By: 1MS19IS051
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>
#include "vadd.c"

int main(){
    int n = 10;
    // Create three arrays
    double a[20], b[20], c[20];

    for (int i = 0; i < n; i++){
        // Assign random values
        a[i] = rand() % n;
        b[i] = rand() % n;
    }

    for (int i = 0; i < n; i++){
        // Print the summations
        printf("%f\n", vectAdd(c,a,b,n)[i]);
    }
    return 0;
}
```

```python
# Authored By: 1MS19IS051
#! /usr/bin/env python
from PyQt5.QtWidgets import QApplication
from PyQt5.QtPrintSupport import QPrinter
from PyQt5.QtGui import QTextDocument
import argparse
import logging
import os
import re
import sys

try:
    import pygments
    from pygments import lexers, formatters, styles
except ImportError as ex:
    logging.warning('\nCould not import the required "pygments" \
        module:\n{}'.format(ex))
    sys.exit(1)

__version__ = '1.1.0'


def logger(func):
    def log_wrap(self, ifile=None, ofile=None, size="A4"):
        logging.getLogger().name = "code2pdf> "
        logging.getLogger().setLevel(logging.INFO)
        func(self, ifile, ofile, size)
    return log_wrap


class Code2pdf:

    """
            Convert a source file into a pdf with syntax highlighting.
    """
    @logger
    def __init__(self, ifile=None, ofile=None, size="A4"):
        self.size = size
```

```
        if not ifile:
            raise Exception("input file is required")
        self.input_file = ifile
        self.pdf_file = ofile or "{}.pdf".format(ifile.split('.')[0])

    def highlight_file(self, linenos=True, style='default'):
        """ Highlight the input file, and return HTML as a string. """
        try:
            lexer = lexers.get_lexer_for_filename(self.input_file)
        except pygments.util.ClassNotFound:
            # Try guessing the lexer (file type) later.
            lexer = None

        try:
            formatter = formatters.HtmlFormatter(
                linenos=linenos,
                style=style,
                full=True)
        except pygments.util.ClassNotFound:
            logging.error("\nInvalid style name: {}\nExpecting one of:\n \
                {}".format(style, "\n    ".join(sorted(styles.STYLE_MAP))))
            sys.exit(1)

        try:
            with open(self.input_file, "r") as f:
                content = f.read()
                try:
                    lexer = lexer or lexers.guess_lexer(content)
                except pygments.util.ClassNotFound:
                    # No lexer could be guessed.
                    lexer = lexers.get_lexer_by_name("text")
        except EnvironmentError as exread:
            fmt = "\nUnable to read file: {}\n{}"
            logging.error(fmt.format(self.input_file, exread))
            sys.exit(2)

        return pygments.highlight(content, lexer, formatter)

    def init_print(self, linenos=True, style="default"):
        app = QApplication([])  # noqa
        doc = QTextDocument()
        doc_html = self.highlight_file(linenos=linenos, style=style)
        doc_html = re.sub(re.compile(r'<http://pygments.org>'), '', doc_html)
        doc.setHtml(doc_html)
        printer = QPrinter()
        printer.setOutputFileName(self.pdf_file)
        printer.setOutputFormat(QPrinter.PdfFormat)
        page_size_dict = {"a2": QPrinter.A2, "a3": QPrinter.A3, "a4": QPrinter.A4, "letter": QPrinter.Letter}
        printer.setPageSize(page_size_dict.get(self.size.lower(), QPrinter.A4))
        printer.setPageMargins(15, 15, 15, 15, QPrinter.Millimeter)
        doc.print_(printer)
        logging.info("PDF created at %s" % (self.pdf_file))


def get_output_file(inputname, outputname=None):
    """ If the output name is set, then return it.
        Otherwise, build an output name using the current directory,
        replacing the input name's extension.
    """
    if outputname:
        return outputname

    inputbase = os.path.split(inputname)[-1]
    outputbase = "{}.pdf".format(os.path.splitext(inputbase)[0])
    return os.path.join(os.getcwd(), outputbase)


def parse_arg():
    parser = argparse.ArgumentParser(
        description=(
            "Convert given source code into .pdf with syntax highlighting"),
        epilog="Author:tushar.rishav@gmail.com"
    )
    parser.add_argument(
        "filename",
        help="absolute path of the python file",
        type=str)
    parser.add_argument(
        "-l",
        "--linenos",
        help="include line numbers.",
        action="store_true")
    parser.add_argument(
        "outputfile",
        help="absolute path of the output pdf file",
        nargs="?",
        type=str)
    parser.add_argument(
        "-s",
```

```
        "--size",
        help="PDF size. A2,A3,A4,A5,letter etc",
        type=str,
        default="A3")
    parser.add_argument(
        "-S",
        "--style",
        help="the style name for highlighting.",
        type=str,
        default="default",
        metavar="NAME")
    parser.add_argument(
        "-v",
        "--version",
        action="version",
        version="%(prog)s v. {}".format(__version__))
    return parser.parse_args()


def main():
    args = parse_arg()
    pdf_file = get_output_file(args.filename, args.outputfile)
    pdf = Code2pdf(args.filename, pdf_file, args.size)
    pdf.init_print(linenos=args.linenos, style=args.style)
    return 0

if __name__ == "__main__":
    sys.exit(main())

```