2 A Simple Function [5 points]

The program prints: 2 246 6 8 10

3 Fix the Function [1 point/fix, so 2 points for first 2]

3.1

Either declare a function prototype for printNum before main, or move the definition of printNum to before main.

3.2

Either add an int argument called number to printNum (preferable because it avoids use of global variables), or move the int number declaration to a global variable.

3.3

Make num a pass-by-reference parameter (i.e. add a & before its name).

3.4

Add a return statement to difference returning diff (or just scrap diff altogether and make the function return abs(x-y));.

3.5

Add a third argument to sum.

3.6

Add a * to make line 7 say int *xPtr = arr, *yPtr = ....
4 Sums

In this problem and all others, half a point should be deducted for not using a `const` argument where it would have been appropriate.

4.1 [4 points]

```c
int sum(const int x, const int y) {
    return x + y;
}

double sum(const double x, const double y) {
    return x + y;
}
```

4.2 [1 point]

Mixing and matching an `int` with a `double` makes it ambiguous which one you want to call. The compiler could either cast 1 to a `double` and call the `double` version of `sum`, or it could cast 10.0 to an `int` and call the `int` version.

4.3 [2+2 points]

```c
int sum(const int x, const int y, const int z) {
    return x + y + z;
}

int sum(const int a, const int b, const int c, const int d) {
    return a + b + c + d;
}
```

4.4 [5 + 1 points]

```c
int sum(const int a, const int b, const int c = 0, const int d = 0) {
    return a + b + c + d;
}
```

If the given definitions were included together, the compiler would give a compile error, since it cannot disambiguate between a call to the 3-argument function and a call to the 4-argument one with a default parameter.
4.5 [5 points]

```c
int sum(const int numbers[], const int numbersLen) {
    int sum = 0;
    for (int i = 0; i < numbersLen; ++i) {
        sum += numbers[i];
    }
    return sum;
}
```

4.6 [8 points]

```c
int sum(const int numbers[], const int numbersLen) {
    return numbersLen == 0 ? 0 : numbers[0] + sum(numbers + 1, numbersLen - 1);
}
```

5  Calculating π

5.1 [3 points]

```c
double x = rand() / (double)RAND_MAX, y = rand() / (double)RAND_MAX;
```

5.2 [6 points]

```c
int dartsInCircle = 0;
for (int i = 0; i < n; ++i) {
    double x = rand() / (double)RAND_MAX, y = rand() / (double)RAND_MAX;
    if (sqrt(x*x + y*y) < 1) {
        ++dartsInCircle;
    }
}
```

5.3 [6 points]

```c
double computePi(const int n) {
    srand( time(0) );
    int dartsInCircle = 0;
```
for(int i = 0; i < n; ++i) {
    double x = rand() / (double)RAND_MAX, y = rand() / (double)RAND_MAX;
    if( sqrt(x*x + y*y) < 1 ) {
        ++dartsInCircle;
    }
}

// r^2 is 1, and a/4 = dartsInCircle/n, yielding this for pi:
return dartsInCircle / static_cast<double>(n) * 4;

6. Array Operations

6.1  [4 points]

void printArray(const int arr[], const int len) {
    for(int i = 0; i < len; ++i) {
        cout << arr[i];
        if(i < len-1) {
            cout << ", ";
        }
    }
}

6.2  [4 points]

void reverse(int numbers[], const int numbersLen) {
    for(int i = 0; i < numbersLen / 2; ++i) {
        int tmp = numbers[i];
        int indexFromEnd = numbersLen - i - 1;
        numbers[i] = numbers[indexFromEnd];
        numbers[indexFromEnd] = tmp;
    }
}

6.3  [6 points]

void transpose(const int input[][LENGTH], int output[][WIDTH]) {
    for(int i = 0; i < WIDTH; ++i) {
        for(int j = 0; j < LENGTH; ++j) {
output[j][i] = input[i][j];
}

6.4 [2 points]
A pointer to the first element in the array would be returned, but the array would have gone out of scope, making the pointer invalid.

6.5 [3 points]

void reverse(int numbers[], const int numbersLen) {
    for(int i = 0; i < numbersLen / 2; ++i) {
        int tmp = *(numbers + i);
        int indexFromEnd = numbersLen - i - 1;
        *(numbers + i) = *(numbers + indexFromEnd);
        *(numbers + indexFromEnd) = tmp;
    }
}

7 Pointers and Strings

7.1 [5 points]

int stringLength(const char *str) {
    int length = 0;
    while(*(str + length) != '\0') {
        ++length;
    }
    return length;
}

7.2 [3 points]

void swap(int &x, int &y) {
    int tmp = x;
    x = y;
    y = tmp;
}
7.3 [4 points]

```c
void swap(int *x, int *y) {
    int tmp = *x;
    *x = *y;
    *y = tmp;
}
```

---

7.4 [5 points]

```c
void swap(int **x, int **y) {
    int *tmp = *x;
    *x = *y;
    *y = tmp;
}
```

---

7.5 [8 points]

1. char *nthCharPtr = &oddOrEven[5];
2. nthCharPtr -= 2; or nthCharPtr = oddOrEven + 3;
3. cout << *nthCharPtr;
4. char **pointerPtr = &nthCharPtr;
5. cout << *pointerPtr;
6. cout << **pointerPtr;
7. nthCharPtr++; to point to the next character in oddOrEven (i.e. one character past the location it currently points to)
8. cout << nthCharPtr - oddOrEven;