I/O WITH MULTIPLE VARIABLES

//Inputting 5 numbers and averaging them
int main() {
    int x1, x2, x3, x4, x5;
    cout << "Enter 5 numbers: ";
    cin >> x1 >> x2 >> x3 >> x4 >> x5;

    double avg = (x1 + x2 + x3 + x4 + x5) / 5.0;
    cout << "Average: " << avg << endl;

    return 0;
}

Now how about a program to store and calculate the average of 30 exam grades? Should we create 30 variables? Of course not! Luckily, C++ has an easy way to manage many variables.
INTRODUCING ARRAYS

Arrays enable us to work with any number of variables! Arrays have the following properties:

- **Data-type** – Can be `int`, `char`, `double`, `string`, `bool`, etc
- **Name** – Arrays have names just like variables do!
- **Size** – How many array “boxes” would you like?

```c
int grades[10];
```

This is an `int` array of size 10!
This is as good as making 10 individual integer variables!
Each “box” of any array is a variable with the data-type of the array! In the example above, each box of the array is an integer variable! You can access an array box/element as follows: `array_name[array_index]`
We usually use for-loops to traverse (travel across) arrays! Again, remember that array indexes start from 0 and end at array_size-1:

Our “usual” loop
for (int i = 0; i < n; ++i) { //”n” is the size of the array
    ...do something with array element at index i...
}

You may also do the following, though the loop above is preferred:
for (int i = 0; i <= n-1; ++i) {
    ...do something with array element at index i...
}
int array[3] = {4, 7, -1}; // initialization

// print individually
cout << array[0]; // 4
cout << array[1]; // 7
cout << array[2]; // -1

// print with loop
for (int i = 0; i < 3; ++i)
    cout << array[i] << endl;

// assign individually
array[0] = 5; // 5
array[1] = array[0]; // 5

// assign with loop
for (int i = 0; i < 3; ++i)
    array[i] = i;
//Inputting 30 grades and averaging them
int main() {
    int grades[30]; //making room in memory for 30 integers

    for (int i = 0; i < 30; ++i) {
        cout << "Enter grade #" << i+1 << ": ";
        cin >> grades[i]; //yes, this is possible!
    } //remember that each element of grades is an int variable!

    double avg = 0;
    for (int i = 0; i < 30; ++i)
        avg += grades[i];

    cout << "Class average: " << avg/30 << endl;
    return 0;
}
DON'T DO THIS

```cpp
int n;
cout << "Enter a size: ";
cin >> n;

int array[n]; //BIG NO-NO!!
```

Chances are your compiler may allow this, but what you see above is illegal by C++ standards! Don’t do it. In this course, we’ll stick with arrays with “set” sizes (i.e. sizes known while writing the program):

```cpp
int array[1000];
for (int i = 0; i < 100; ++i) {
    cin >> array[i];
} //you need not use every element of the array you declared!
```
Arrays and Functions

Typical array print function:

```cpp
void print(int array[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << array[i] << " ";
    }
    cout << endl;
}
```

Optional: Put array size in parameter

```cpp
void print(int array[4], int size) {
    for (int i = 0; i < size; ++i) {
        cout << array[i] << " ";
    }
    cout << endl;
}
```

In `main()`:

```cpp
int main() {
    int a[4] = {1, 4, 0, 8};
    //pass the NAME of the array
    //to the function!
    print(a, 4);
    return 0;
}
```

Output

```
1 4 0 8
```
Arrays are always “passed by reference”.

```cpp
void add5(int array[], int size) {
    for (int i = 0; i < size; ++i)
        array[i] += 5;
}

void print(int array[], int size) {
    for (int i = 0; i < size; ++i) {
        cout << array[i] << " ";
    }
    cout << endl;
}
```

In `main()`:

```cpp
int main() {
    int a[4] = {1, 4, 0, 8};
    cout << "Now: ";
    print(a, 4);
    add5(a, 4);
    cout << "Later: ";
    print(a, 4);
    return 0;
}
```

Output
Now: 1 4 0 8
Later: 6 9 5 13