Lecture 9:

Arrays

Prof. Shervin Shirmohammadi
University of Ottawa

A Problem with Simple Variables…

- Suppose that an algorithm reads 5 integers and displays them in reverse order:

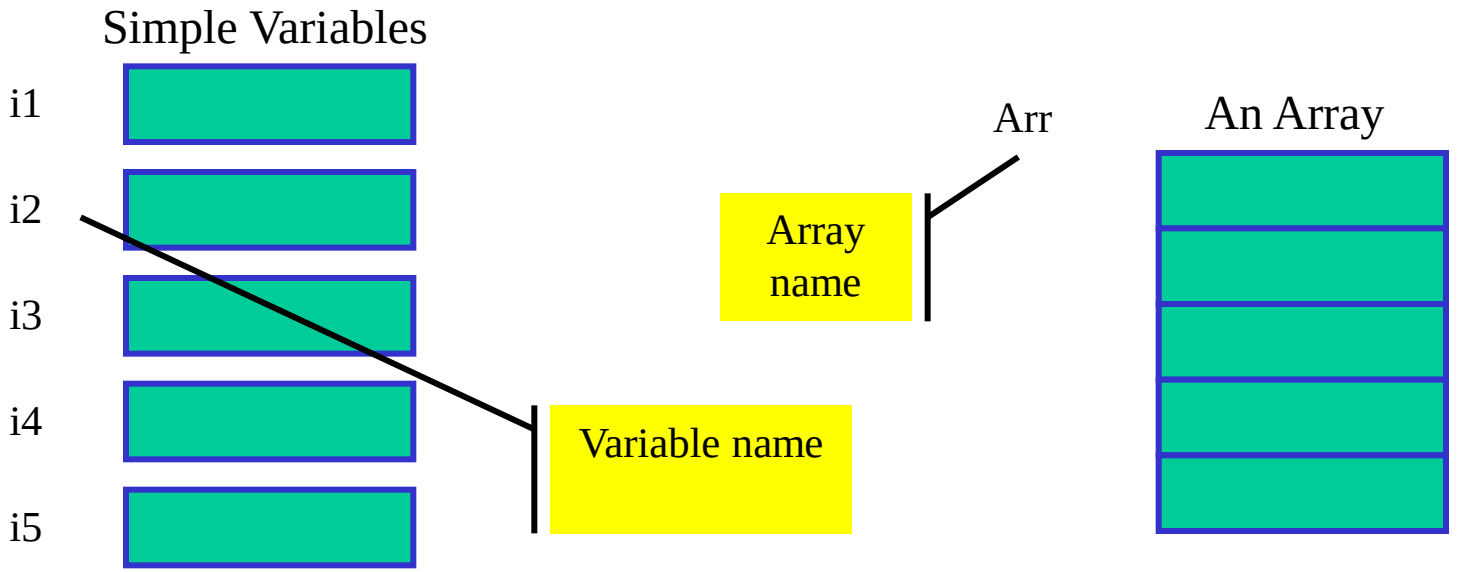
```
Read value into i1
Read value into i2
Read value into i3
Read value into i4
Read value into i5

Print(i5)
Print(i4)
Print(i3)
Print(i2)
Print(i1)
```

- What happens with 1000 integers? N integers? How to declare so many variables?

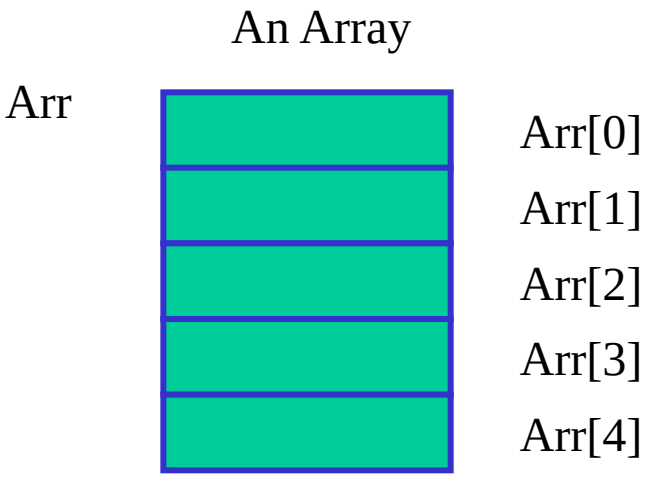
Arrays

- "Simple" variables contain one value.
- An array has many variables inside it, each able to contain a different value.
- An array is essentially a collection of variables of the same type.



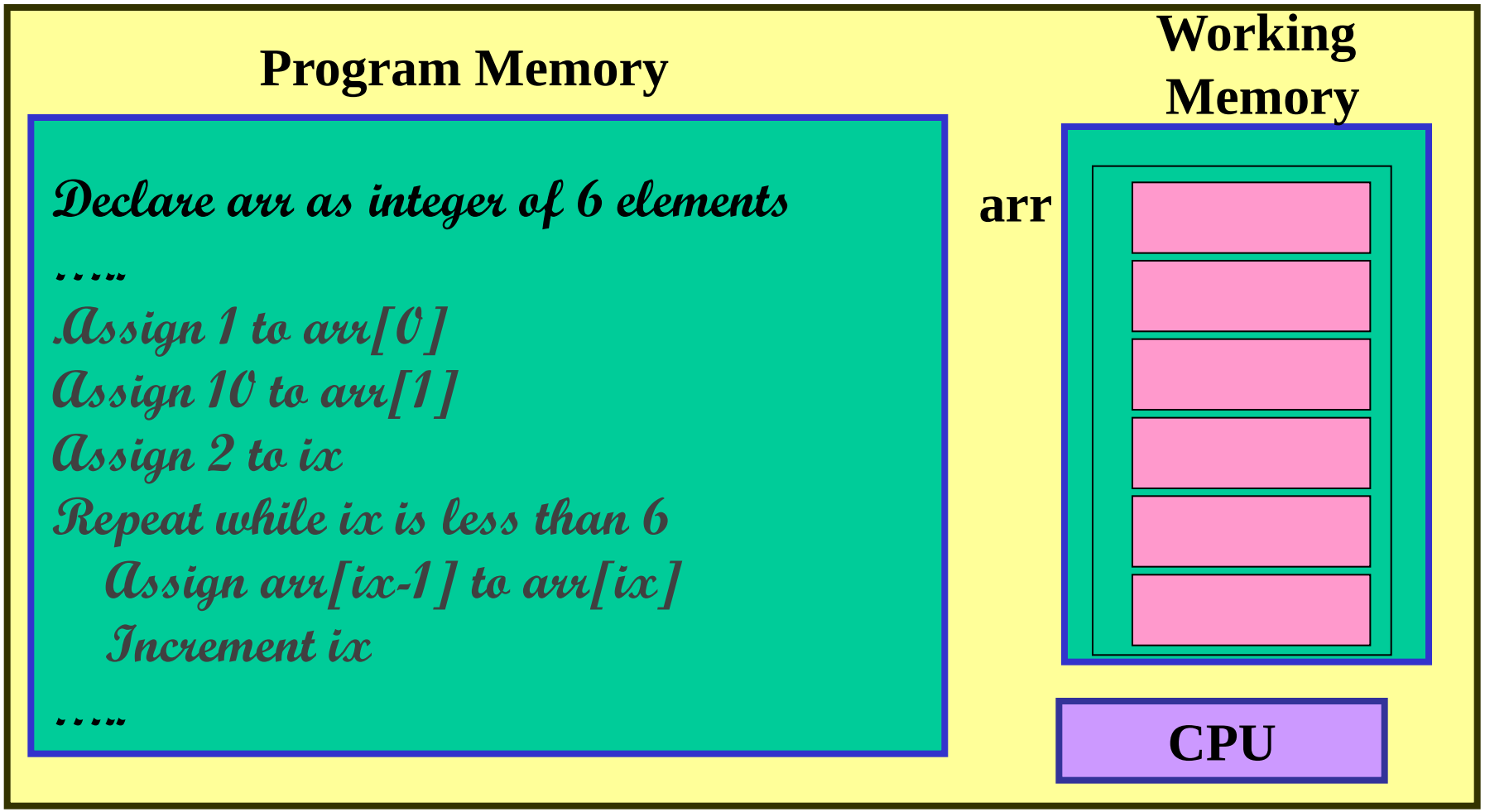
Arrays (continued)

- If array Arr has 5 positions, we refer to them using the integers 0-4, called indices or subscripts.
  - e.g. Arr[2] is the THIRD position with index 2.
  - Note that Arr[2] is equivalent to a variable name and can be used anywhere a variable name is used, e.g. in expressions.



A one dimensional array

- An array is a collection of memory locations where multiple values of the same type can be stored



Example

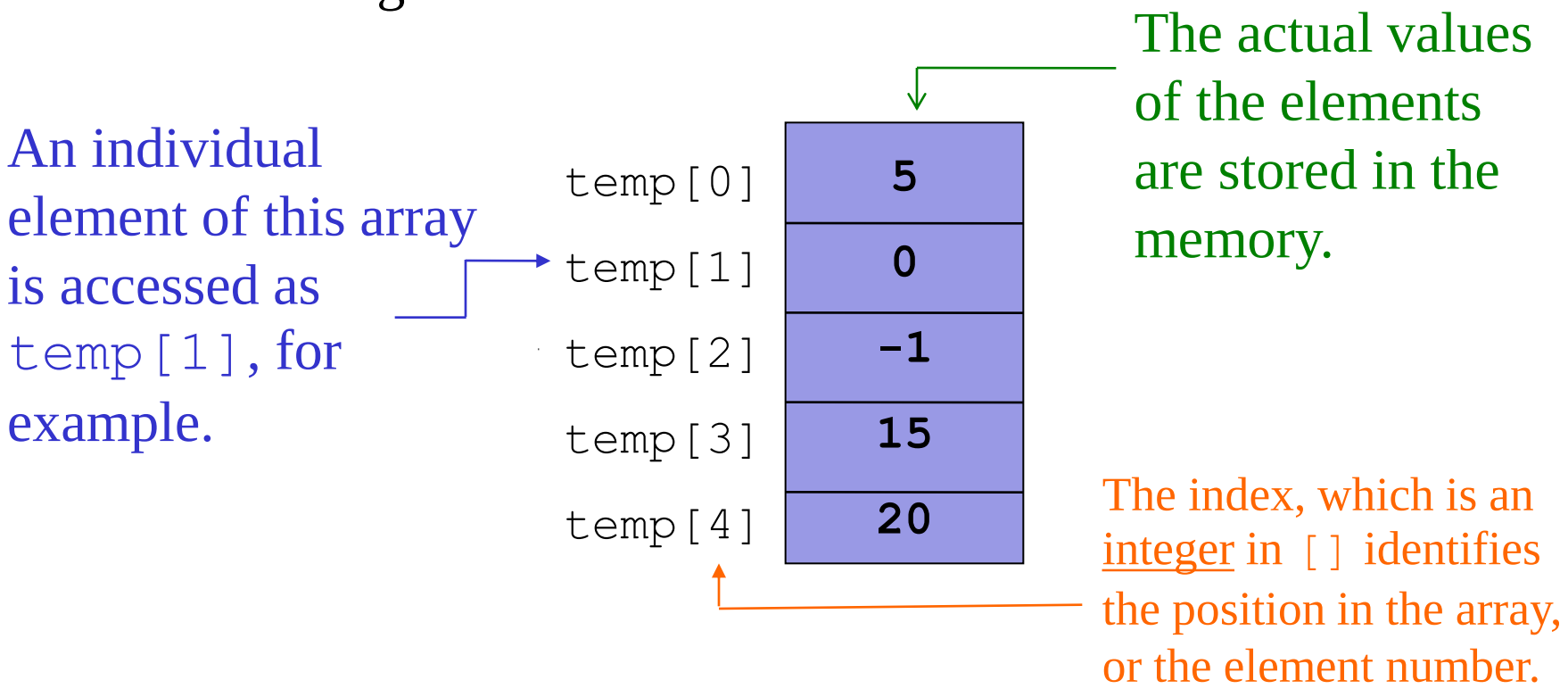
- Declare an array to store the final exam mark (in percentage, like 75, 90, etc.) in a class of 200 students, and assign the value of 80 to all of them. Then, print the entire class mark.

The one dimensional array

- A one dimensional array consists of an amount of contiguous memory elements that can contain data of the same type.
  - The number of elements is fixed.
- The name of an array must be created in the same way as a variable name and using the same set of characters available for a variable name.
  - The name of the array corresponds to the address of the first data element in the array
- An element of an array is accessible by citing the name of the array followed by the element number or index in brackets `[ ]`.
  - The name of the array with an index is equivalent to a variable name and can be used anywhere a variable name is used (e.g. expressions)
  - Note that expressions can be used as an index
  - Index values start at 0

One Dimensional Arrays in C

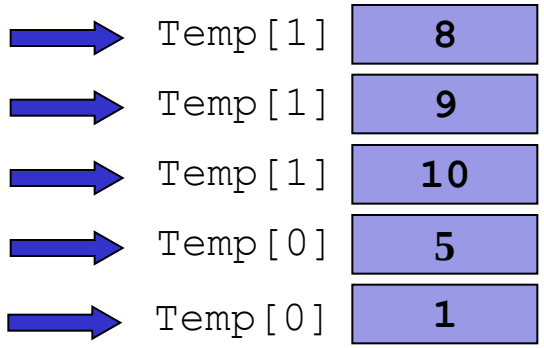
- In C, an array consists of a name and indices used to access its elements.
- In the memory, a 1-D array of dimension 5, called `temp` and containing `int`'s looks like:



- **In C, the first element in an array is always element 0.**
  - ➔ An array of 5 elements is indexed from 0 to 4,
  - ➔ an array of N elements is accessed using indices 0 though N-1.
  - ➔ **Careful**: the i^{th} element is at position i-1 in the array; e.g, the 4^{th} element is in position 3.
- The position of the element or the index must be an integer or an expression that evaluates to an integer.
- An element of an array is used in the same manner as a variable.
  - It can be used to the left of an assignment operator to store a value in the element.
  - It can be used in an arithmetic or logical expression.

Assume that `temp` is an array of `int`'s

E.g.:
```
temp[1] = 5 + 3;
temp[1] = temp[2-1] + 1;
temp[1]++;
temp[0] = temp[1] / 2;
temp[0] = (temp[0] <= temp[1]);
```



- The elements of an array can also be used as an argument in a `printf` or a `scanf`:

E.g.:
```
printf("%d\n", (temp[0]+temp[1]));
scanf("%d", &temp[1]);
```

Definition of One Dimensional Arrays

- An array is defined in a declaration; the declaration specifies:
  - the name of the array,
  - the type of data stored in the array, and
  - the size of the array (number of elements).

  E.g.:
  ```
  int temp[5];
  float x[5], y[20];
  ```
  - The first declaration defines the array `temp` which contains 5 data elements of type `int`.
  - The second declaration defines the arrays `x` and `y`, which contain 5 and 20 elements, respectively, of type `float`.
  - The memory required to store these arrays is reserved by the compiler.

Initialization of One Dimensional Arrays

- An array can be initialized using a looping structure.

  E.g.:

```
int temp[5], ctr;
for(ctr=0; ctr<=4; ctr++)
{
   temp[ctr]=0;
   x[ctr]=(float)ctr/2;
}
```

Ctr	temp[ctr]
0	0
1	0
2	0
3	0
4	0

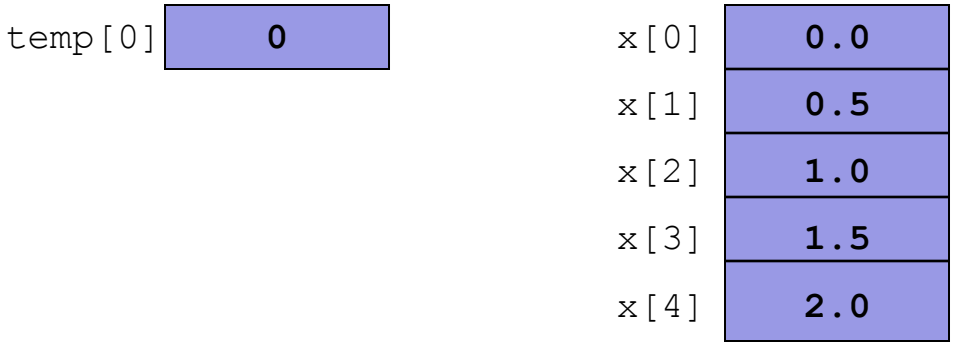
- We can also initialize an array in the declaration in which it is defined.

  E.g.: `float x[5]={0.0, 0.5, 1.0, 1.5, 2.0};`

x[0]	0.0
x[1]	0.5
x[2]	1.0
x[3]	1.5
x[4]	2.0

Initialization of One Dimensional Arrays

- The size of an array can also be defined via the initialization list.
  - If the array size is not stated but the declaration contains an initialization list, then the array size is taken as being equal to the number of items in the initialization list.
    - E.g.: `int temp[]={0};`
      - `float x[]={0.0, 0.5, 1.0, 1.5, 2.0};`
    - After compilation the arrays `temp` and `x` contain the following values:



    - Thus `temp` is an array of 1 `int` and `x` is an array of 5 `float`'s.
  - It is usually not good practice to omit the size of arrays when declaring them!

Symbolic Constants with Arrays

- The use of descriptive symbolic constants in the definition of arrays and in their manipulation is **strongly** encouraged since they:
  - facilitate modifications to the program if the array size must change, and they
  - eliminate "magic" numbers from the program, thus rendering it more readable.

E.g.:

```
#define SIZE 5
      ⋮
int temp[SIZE ], ctr;
for(ctr=0; ctr < SIZE; ctr++)
   temp[ctr]=0;
```

  - In the above example, if the number of temperatures to be stored must be increased from 5 to 10 then only one line of the program would need to be modified:

```
#define NBR_TEMPS 10
```

  - In a large program containing many arrays and loops to manipulate them, the use of symbolic constants greatly enhances the maintainability of the code.

Running Over Array Limits

- There is no mechanism in C to prevent a program from running over the upper and lower limits of an array during execution!
  - Reading or accessing an element outside of the array limits will:
    - generally introduce errors in the results obtained from the program, but
    - usually will not cause the computer to "crash" or to become "hung-up".
  - Assigning a value to an element outside of the array limits will:
    - generally introduce errors into the results obtained from the program, and
    - often will cause the computer to "crash" or to become "hung-up".
- The most common cause for running over the array limits is an error in the looping structure(s) used to traverse the array. **Always verify that**:
  - the index never becomes negative in value, and
  - that the index always remain less than or equal to the size of the array minus 1.
- Be consistent when writing loops that traverse an array. For example, use a symbolic constant and a logical expression similar to:
    - `for(ctr=0; ctr < SIZE; ctr++)`