COMP1911 - Computing 1A

Dr Binghao Li
School of Minerals and Energy Resources Engineering
School of Computer Science and Engineering
binghao.li@unsw.edu.au
June 2022

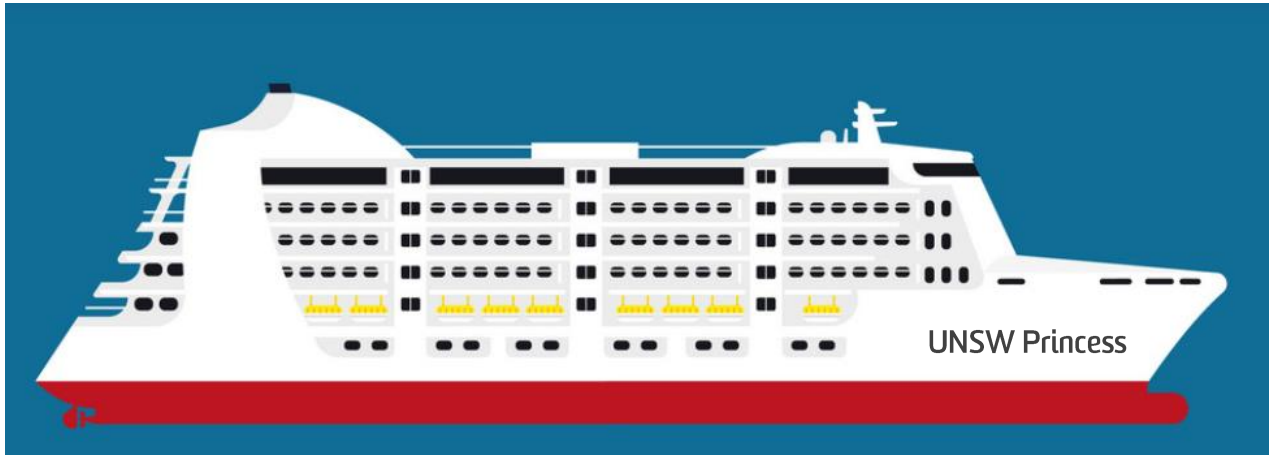
8. Arrays and Memory

In this lecture we will cover:

- What is array
- Memory and address
- Declaring arrays
- Initializing arrays
- Accessing array elements
- Multi-dimensional array
- Passing arrays to functions

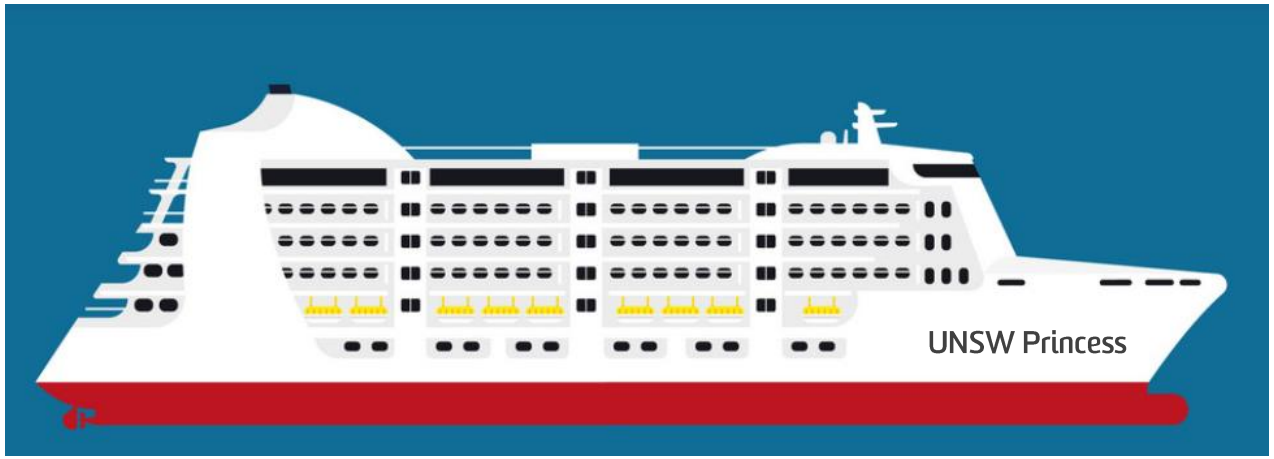
Memory - Cabins

A cruise ship called UNSW Princess, there are 1024 cabins, each cabin is only suitable for a small person. However, if you are big, you can request for several contiguous cabins.
Cabin number start from 0 - 0x000 to 1023 - 0x3FF



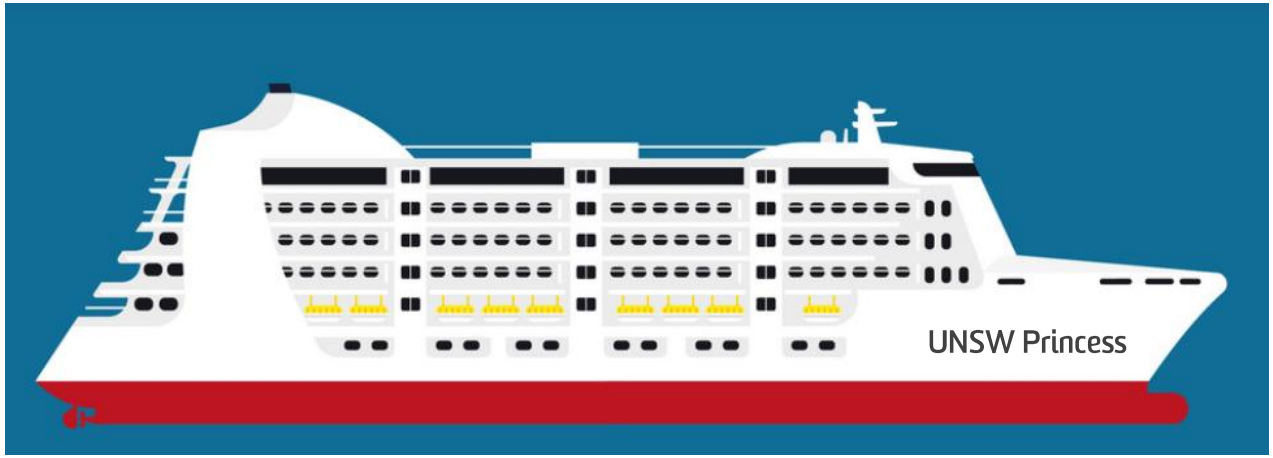
Address, variable and value

- Cabin number;
- I occupy cabins 0 and 1, we name my room as Binghao's Room;
- Dylan occupies cabins 2 and 3, we call his room as Dylan's Room;
- Person insider the room.

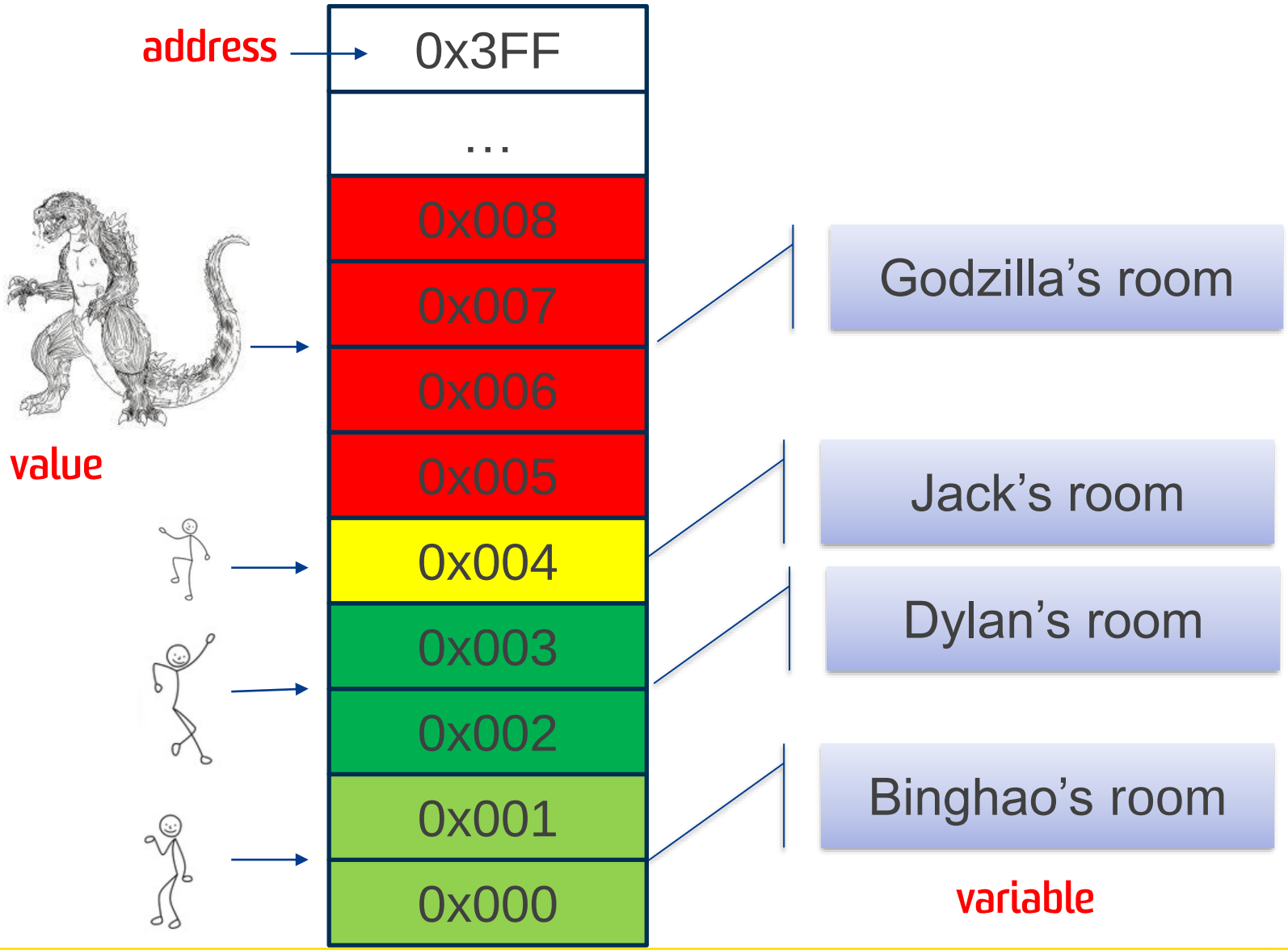


Address, variable and value

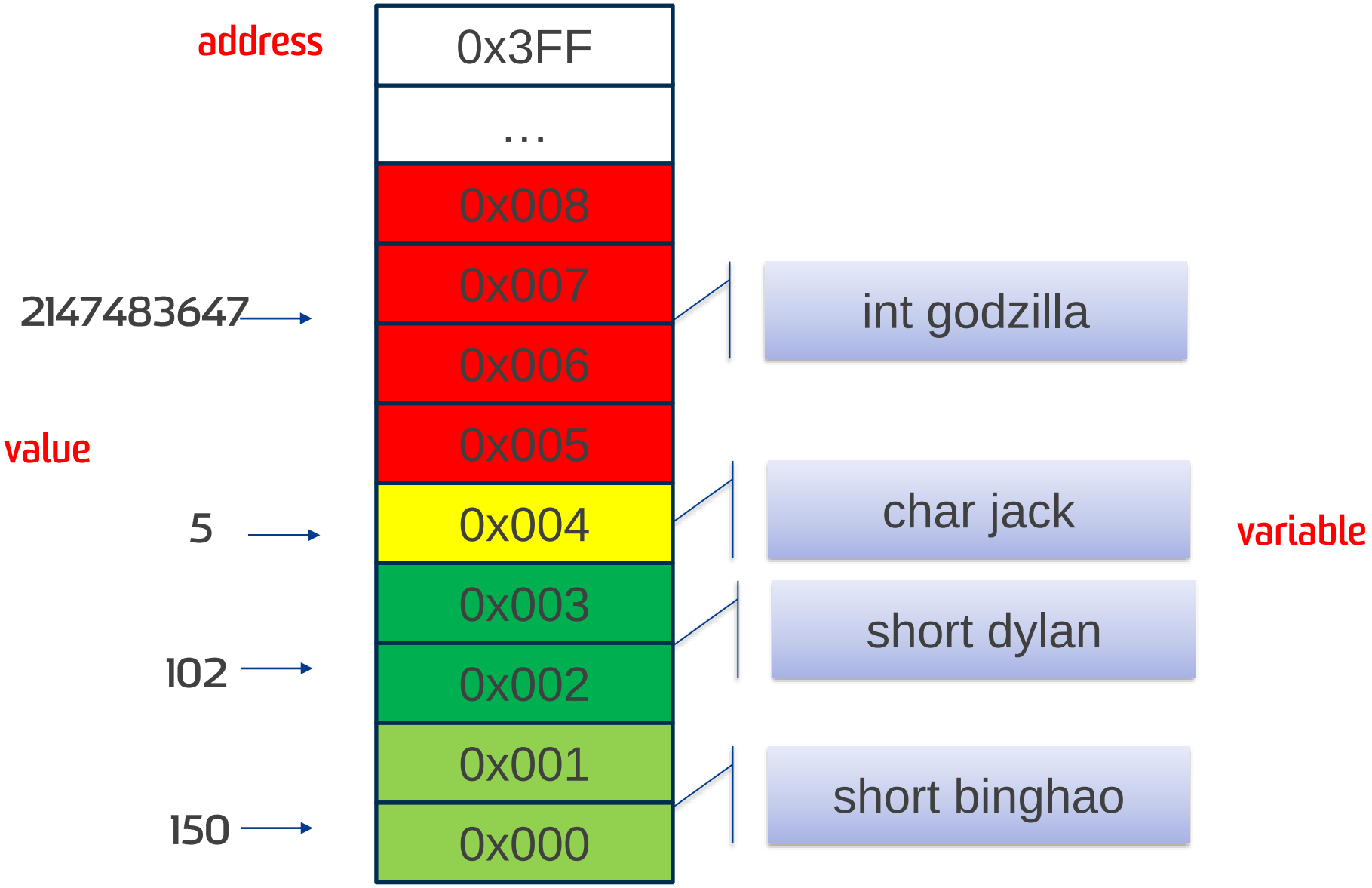
- Cabin number – address
- I occupy cabins 0 and 1, I name my room as Binghao's Room – variable name;
- Dylan occupies cabins 2 and 3, we call his room as Dylan's Room – variable name;
- Person insider the room – Value



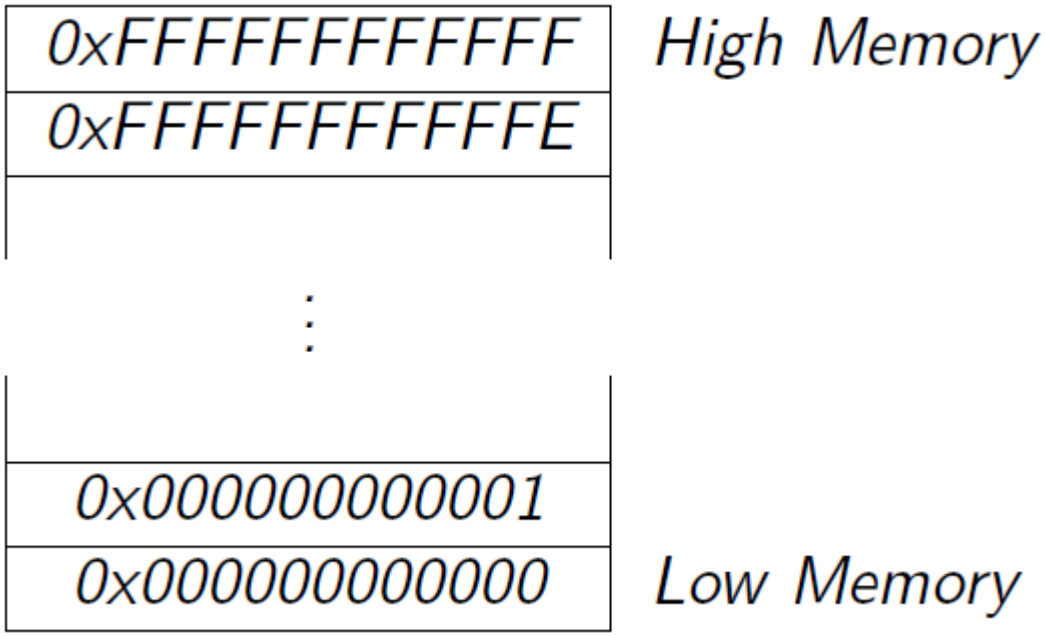
UNSW Princes



Computer memory



Memory Organisation



281,474,976,710,656

Memory from 0x000000000000 - 0xFFFFFFFFFFFF, all for variables, how many double type of variables can be stored?

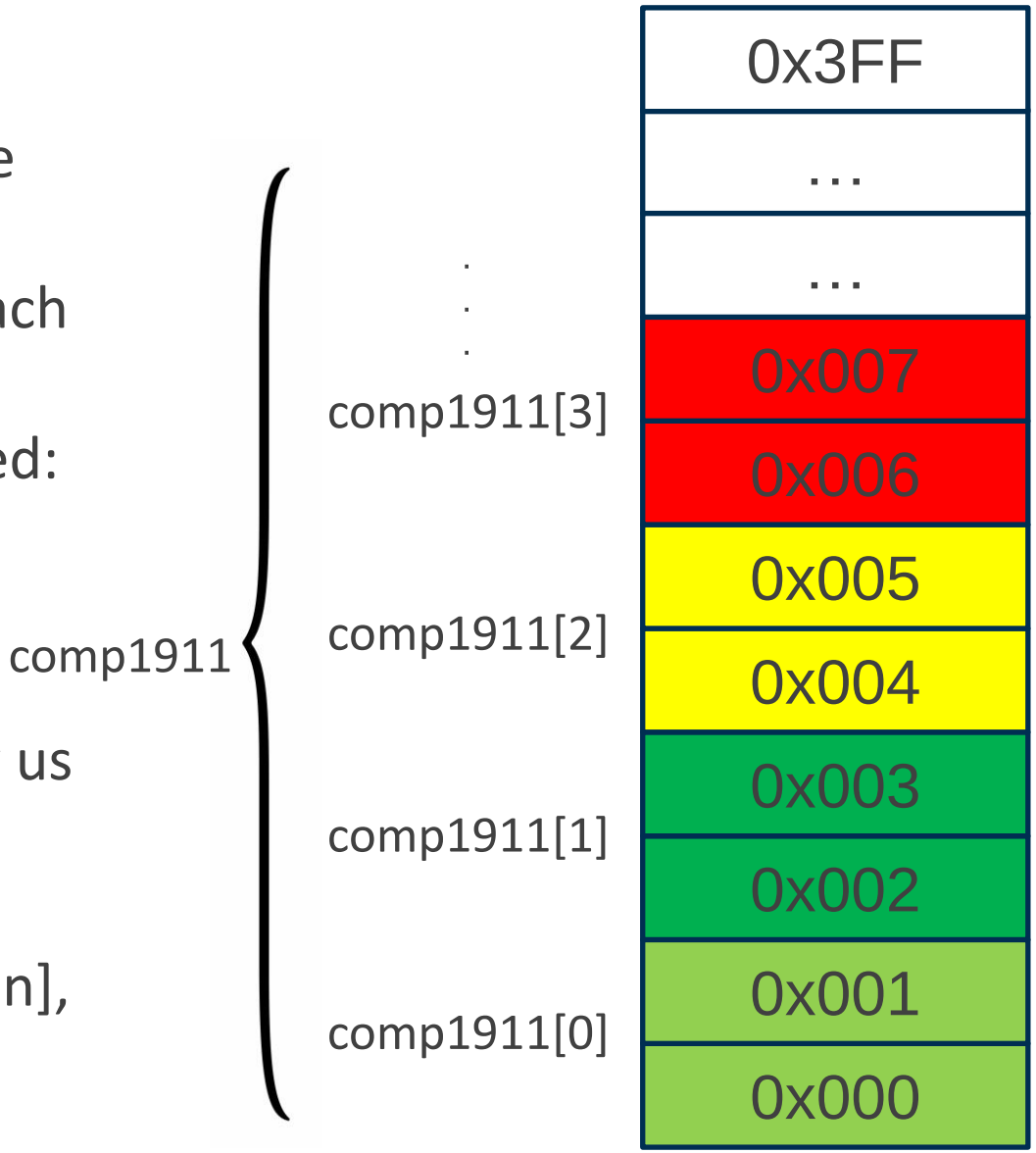
Total Results: 0

Arrays – A group of people with similar features

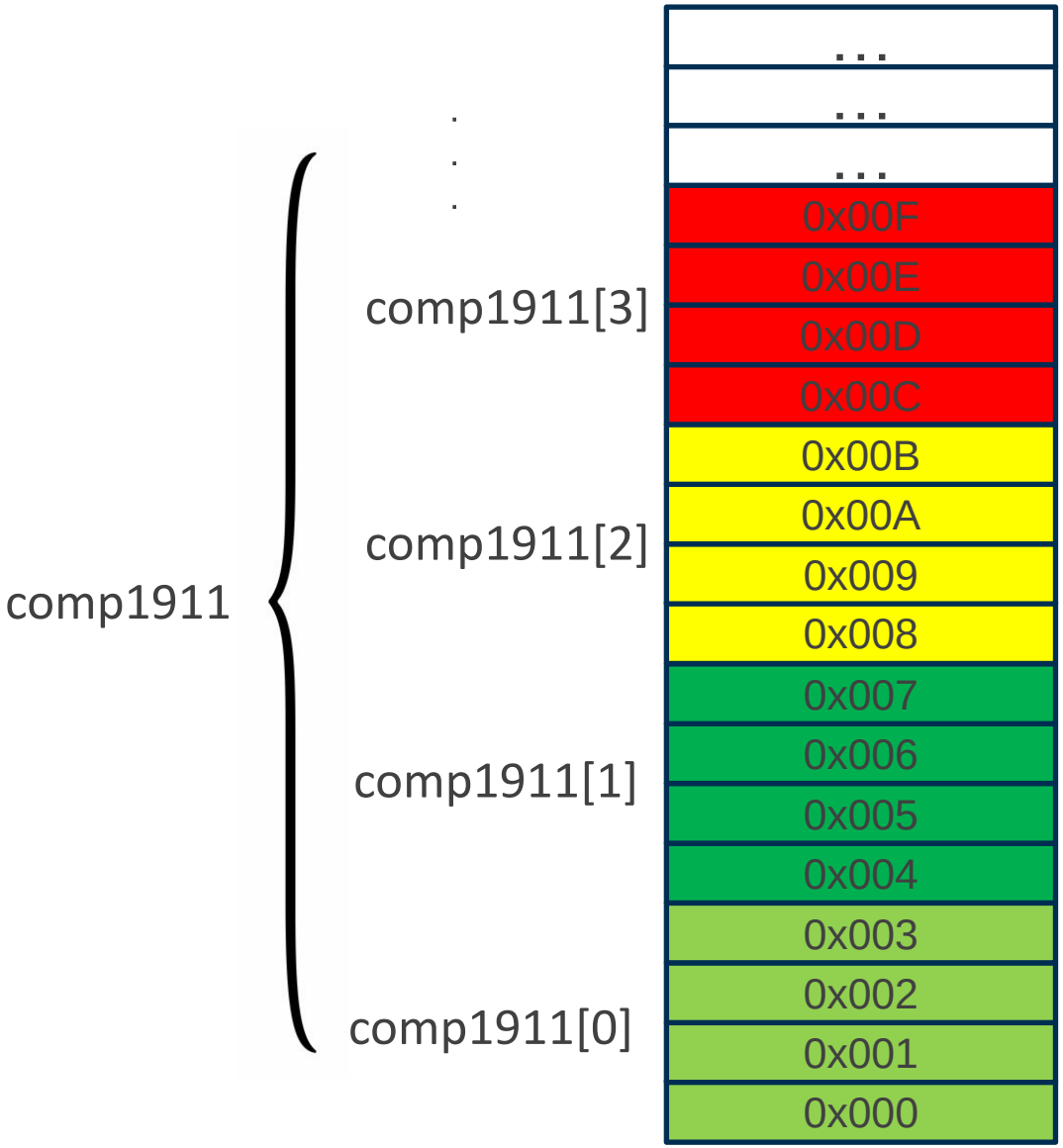
Say COMP1911 booked 190 rooms for all students. We are all medium size, we use two cabins to create a room for each students.
We call the rooms we occupied: comp1911[0], comp1911[1], comp1911[2] etc.

We call all rooms occupied by us – comp1911

Each room will be comp1911[n], n from 0 to 189



Arrays – A group of data with similar features



Arrays

How to store a list of items that have the same data type?
Suppose I need to compute statistics on class marks?

```
int markStudent0, markStudent1, markStudent2, ...;
markStudent0 = 73;
markStudent1 = 42;
markStudent2 = 99;
...
```

- cumbersome, need hundreds of individual variables
- can't write while loop which executes for each student
- becomes unfeasible if dealing with a lot of values

Arrays

Solution use an array

```
int markStudent[3]; // # of items, not final index
markStudent[0] = 73;
markStudent[1] = 42;
markStudent[2] = 99;
...
```

Arrays

```
// Declare an array with 10 elements
// and initialises all elements to 0.
int myArray[10] = {0};
```

	myArray
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Arrays

```
// Declare an array with 10 elements
// and initialises all elements to 0.
int myArray[10] = {0};

// Put some values into the array.
myArray[0] = 3;
```

	myArray
0	3
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Arrays

```
// Declare an array with 10 elements
// and initialises all elements to 0.
int myArray[10] = {0};

// Put some values into the array.
myArray[0] = 3;
myArray[5] = 17;
```

	myArray
0	3
1	0
2	0
3	0
4	0
5	17
6	0
7	0
8	0
9	0

Arrays

```
// Declare an array with 10 elements
// and initialises all elements to 0.
int myArray[10] = {0};

// Put some values into the array.
myArray[0] = 3;
myArray[5] = 17;
myArray[10] = 42; // <-- Error
```

	myArray
0	3
1	0
2	0
3	0
4	0
5	17
6	0
7	0
8	0
9	0

Array Indices

Note that arrays are indexed from 0 to *size* - 1. Attempting to access an invalid array index is a run-time error!

Note: when the program is compiled without **error** and gives **error** in the **running time** is known as **Run Time Error.**

Arrays

Other ways to define arrays:

```
int myArray[10];
int myArray[] = {3,12,9,12,8,17,33,22,43,10};
int myArray[10] = {3};
```

Each definition creates a int array with 10 elements. What are the differences?

What are the differences?

```
int myArray[10];
int myArray[] =
{3,2,9,2,8,7,3,2,4,0} ;
int myArray[10] = {3};
```

Arrays

Other ways to define arrays:

```
int myArray[10];
int myArray[] = {3,12,9,12,8,17,33,22,43,10};
int myArray[10] = {3};
```

Each definition creates a int array with 10 elements. What are the differences?

Array Size

C arrays are often fixed at compile time. During run-time once they are created they cannot be resized.

C does not store information about the size of arrays, so it is the responsibility of the programmer to manage this information.

Reading Arrays

Scanf can't read an entire array. This will read only 1 number:

```
#define ARRAY_SIZE 42
...
int array[ARRAY_SIZE];
scanf("%d", &array);
```

Instead you must read the elements one by one:

```
i = 0;
while (i < SIZE) {
    scanf("%d", &array[i]);
    i = i + 1;
}
```

Printing Arrays

printf can't print an entire array. This won't compile:

```
#define ARRAY_SIZE 42
...
int array[ARRAY_SIZE];
printf("%d", array);
```

Instead must print the elements one by one:

```
i = 0;
while (i < ARRAY_SIZE) {
    printf("%d\n", array[i]);
    i = i + 1;
}
```

Copying Arrays

Suppose we have the following:

```
int array1[5] = {1, 2, 3, 4, 5};
int array2[5];
```

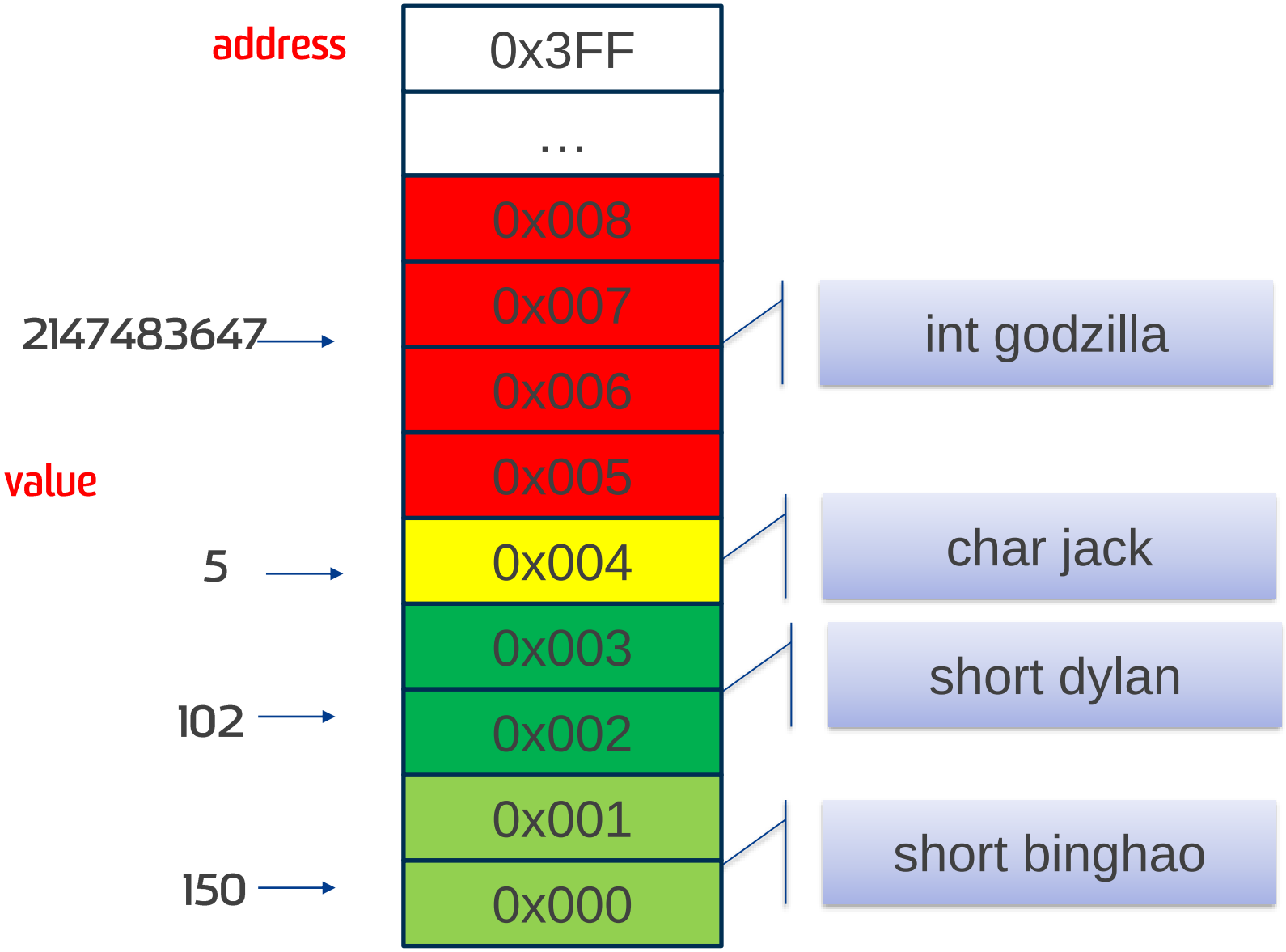
Array assignment not allowed in C. This won't compile:

```
array2 = array1;
```

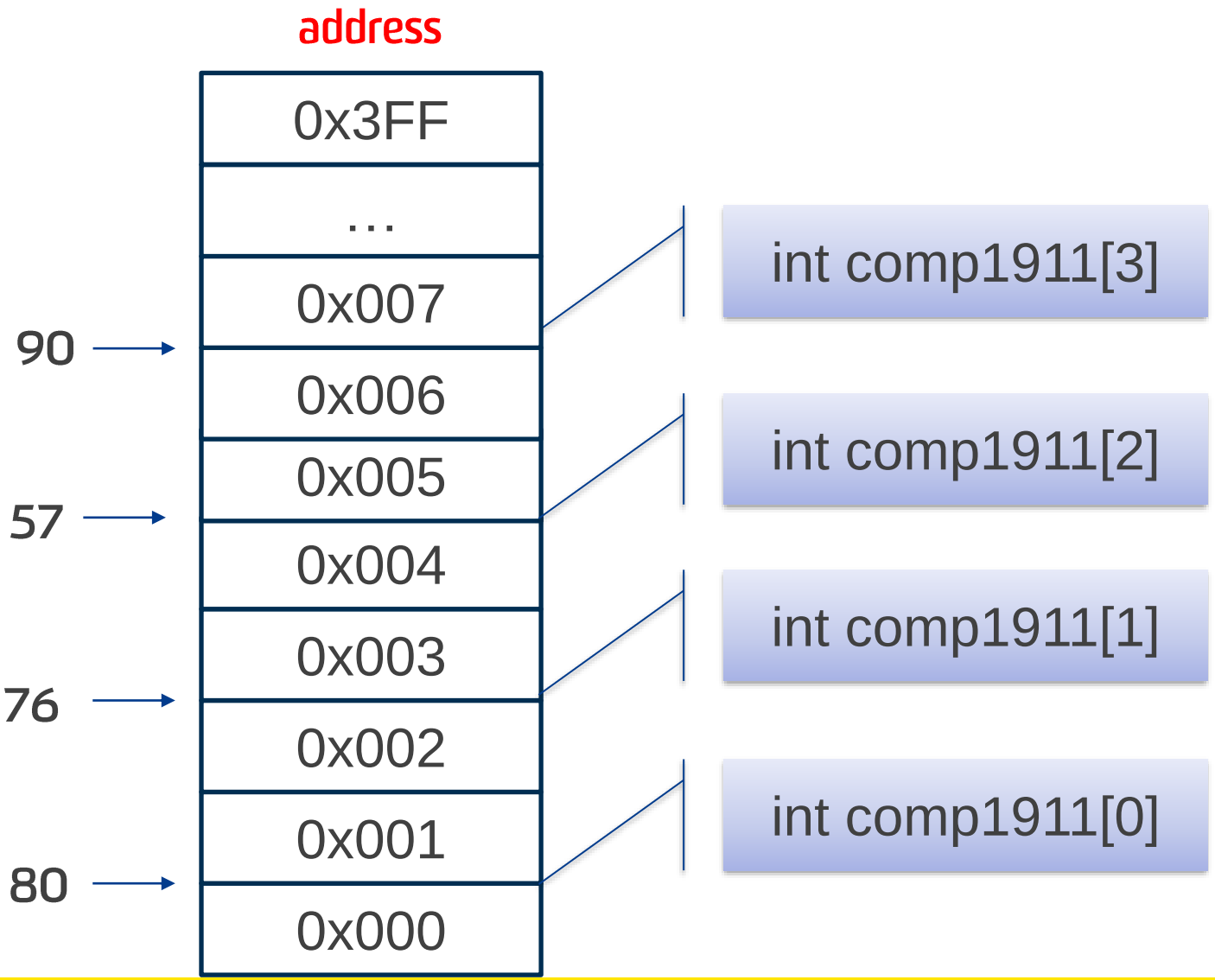
Instead must copy the elements one by one:

```
i = 0;
while (i < 5) {
    array2[i] = array1[i];
    i = i + 1;
}
```

How to store an array



How to store an array



Argument Passing: Array Variables

```c
int main(void) {
    int nums[5] = {0};

    f(nums,5);                  // pass num as argument
    printf("%d\n", nums[0]); // what is printed?
}


void f(int nums[], int size) {
    nums[0] = 42;               // modify argument
}
```

printf() → what is printed?

What is printed?

Total Results: 0

Argument Passing: Array Variables

Array arguments are passed by reference

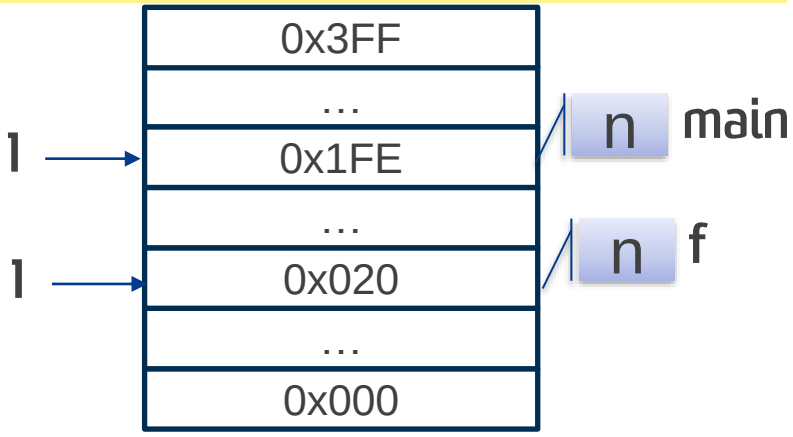
- The array is not copied so changes to array elements visible outside function
- Full explanation will have to wait until we cover pointers

```
int main(void) {
      int nums[5] = {0};

      f(nums,5);                    // pass num as argument
      printf("%d\n", nums[0]); // what is printed?
}


void f(int nums[], int size) {
      nums[0] = 42;                 // modify argument
}
```

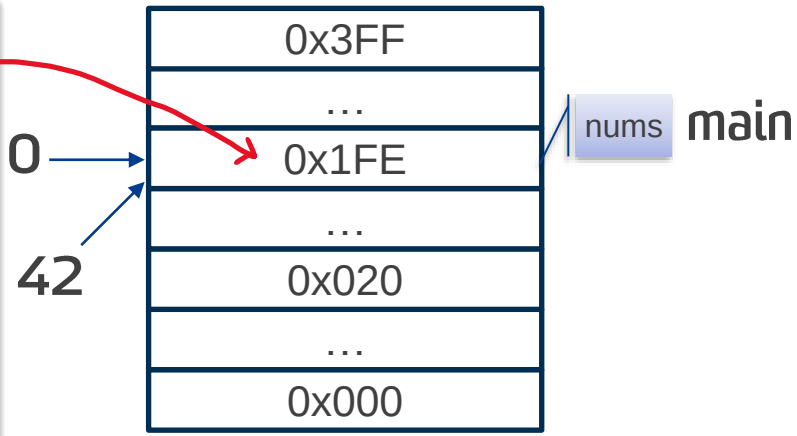
printf() → 42, why?

Argument Passing: variable vs array

```c
int main (void) {
    int n = 1;
    n = f(n) + 10;
    printf("%d\n", n); // what is printed?
}

int f(int n) {
    return (n + 5);
}
```




```c
int main(void) {
    int nums[5] = {0};

    f(nums,5);                  // pass num as argument
    printf("%d\n", nums[0]); // what is printed?
}


void f(int nums[], int size) {
    nums[0] = 42;               // modify argument
}
```



Arrays as Function Arguments

Examples of how the prototypes can be declared:

```
void f0(double ff[SIZE]);
void f1(double ff[]);
```

Notice that the size may be left unspecified.

Consequence: it is up to the programmer to manage the number of elements in its array argument.

Options:

- by using a size constant
- by specifying an additional array size argument

Arrays as Function Arguments

Consider the following:

```c
#define SIZE 10

int sum1(int nums[]);
int sum2(int nums[], int size);

int main(void) {
  int nums[10] = {1, 2, 3};

  sum1(nums);
  sum2(nums, 3);

  return 0;
}
```

The function sum1 uses SIZE to iterate through its array argument, while sum2 uses the supplied size argument. Why is sum2 better?

Beware: Don't Try to Return an Array

It might be tempting to try returning an array from a function:

```
int[] foo(void) {
  int nums[] = {1,2,3};
  return nums;
}
```

This looks good but fails spectacularly! We will cover this next week.

It is possible to return dynamically allocated arrays, which we will learn later in the course.

Beware: Don't Try to Return an Array

Instead of returning an array you can pass in an array, fill it with values. This works because arrays are passed by reference.

```
int main(void){
    int numbers[SIZE];
    //pass the array and the size of the array into foo
    //foo fills it with values
    foo(numbers,SIZE);
    //Use the numbers array which has values
    //foo filled it with
    //etc
}
void foo(int nums[], int size) {
     int i = 0;
     while(i < size){
          nums[i] = i+1;
     }
}
```

Arrays of Arrays

- C supports arrays of arrays.
- This is called a Two-dimensional array
- Useful for multi-dimensional data.

```
int matrix[3][3] = { {1, 2, 3},
                     {4, 5, 6},
                     {7, 8, 9} };

printf("%d\n", matrix[1][1]);
```

What is the output?

Total Results: 0

Read a Two-dimensional Array

```
#define SIZE 42
...
int matrix[SIZE][SIZE];
int i, j;

i = 0
while (i < SIZE) {
  j = 0;
  while (j < SIZE) {
    scanf("%d", &matrix[i][j]);
    j = j + 1;
  }
  i = i + 1;
}
```

Print a Two-dimensional Array

```
...

while (i < SIZE) {
  j = 0;
  while (j < SIZE) {
    print("%d", matrix[i][j]);
    j = j + 1;
  }
  printf("\n");
  i = i + 1;
}
```

How to Store a Two-dimensional Array

int arr[2][3]

2d Arrays as Function Arguments

Recall that arrays are passed to functions by reference.

Examples of how the prototypes can be declared:

```
void f0(double ff[SIZE]);
void f1(double ff[]);
void f2(double ff[][SIZE]);
void f3(double ff[][SIZE1][SIZE2]);
```

Notice that the size of the first (dominant) dimension may be left unspecified.

3d Arrays

int c[2][3][4]

		Columns			
c[0] Array	Rows	c[0][0]	c[0][1]	c[0][2]	c[0][3]
		c[1][0]	c[1][1]	c[1][2]	c[1][3]
		c[2][0]	c[2][1]	c[2][2]	c[2][3]
		Columns			
c[1] Array	Rows	c[0][0]	c[0][1]	c[0][2]	c[0][3]
		c[1][0]	c[1][1]	c[1][2]	c[1][3]
		c[2][0]	c[2][1]	c[2][2]	c[2][3]

Linux and I/O Redirection

Sometimes we want to capture our program output in a file.

Sometimes we don't want to type in the same input again and again when testing our programs.

Linux can do this with file redirection.

- ./prog > sampleoutput
- ./prog < testinput
- ./prog < testinput > sampleoutput

C Arrays

- C array is a collection of variables called **array elements**.
- All array elements must be the same type.
- Array elements don't have a name
- Array elements accessed by a number called the array index.
- Valid array indices for array with *n* elements are *0 .. n - 1*
- Array can have millions/billions of elements.
- Array elements must be initialized.
- Can't assign scanf/printf whole arrays.
- Can assign scanf/printf array elements.

Questions