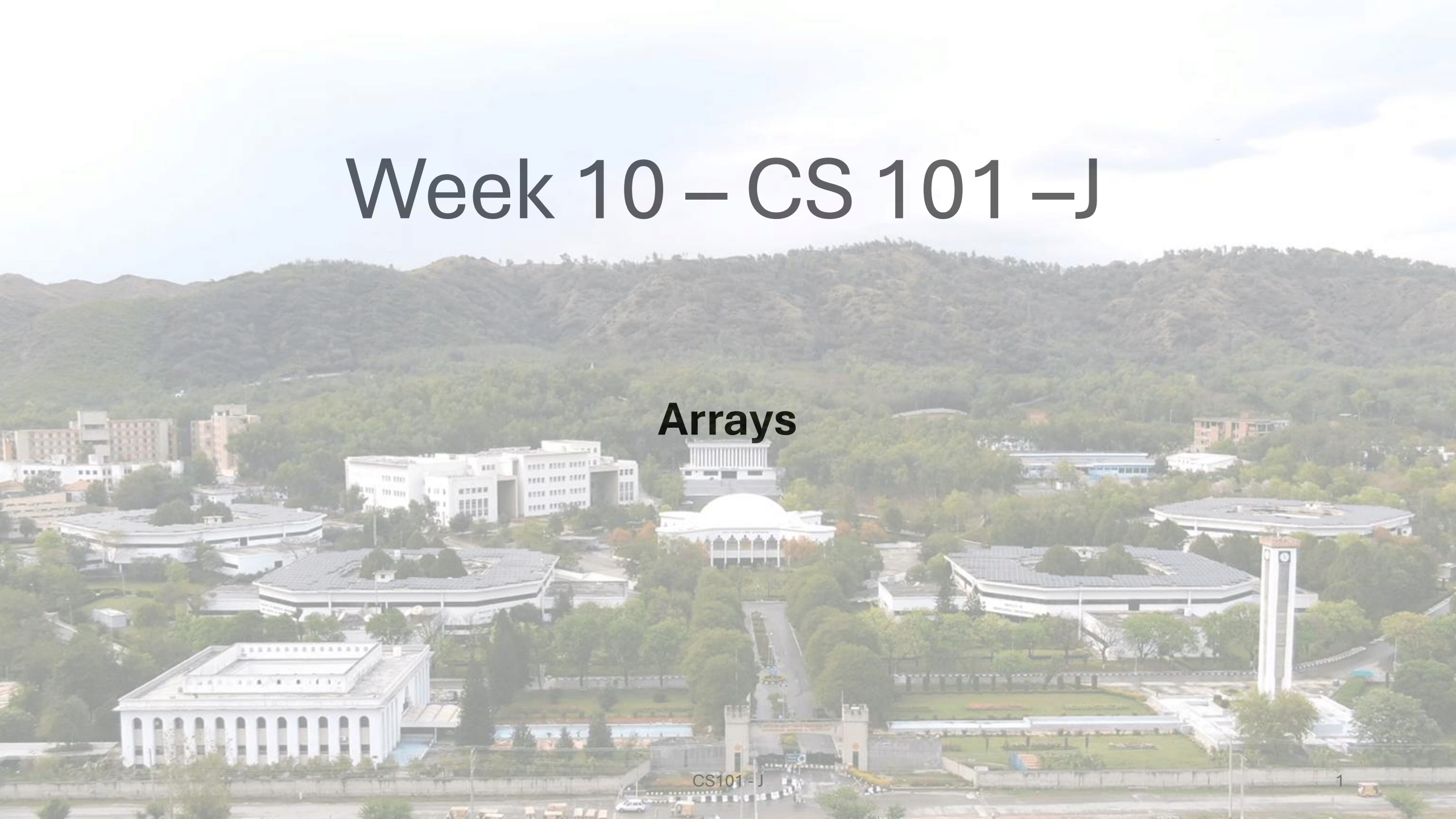
Week 10 – CS 101 –J



Arrays

Recap of Recursion (from week 9)

- Can a function call itself again?
- Yes, it can – and this process is called as recursion
- When using recursion, it is VERY IMPORTANT to write a condition, called as base case – (used to stop infinite calling of function itself)

Recursion

- There are certain problem (engineering and scientific) which requires recursion.
  - E.g. Factorial, Fibonacci Series, Tower of Hanoi, Graph Searching

- Not all problems require recursion concept

Recursion

- Example: factorial

  $n! = n * (n – 1) * (n – 2) * \ldots * 1$

  - Recursive relationship ($n! = n * (n – 1)!$)

    $5! = 5 * 4!$

    $4! = 4 * 3! \ldots$

  - Base case ($1! = 0! = 1$)

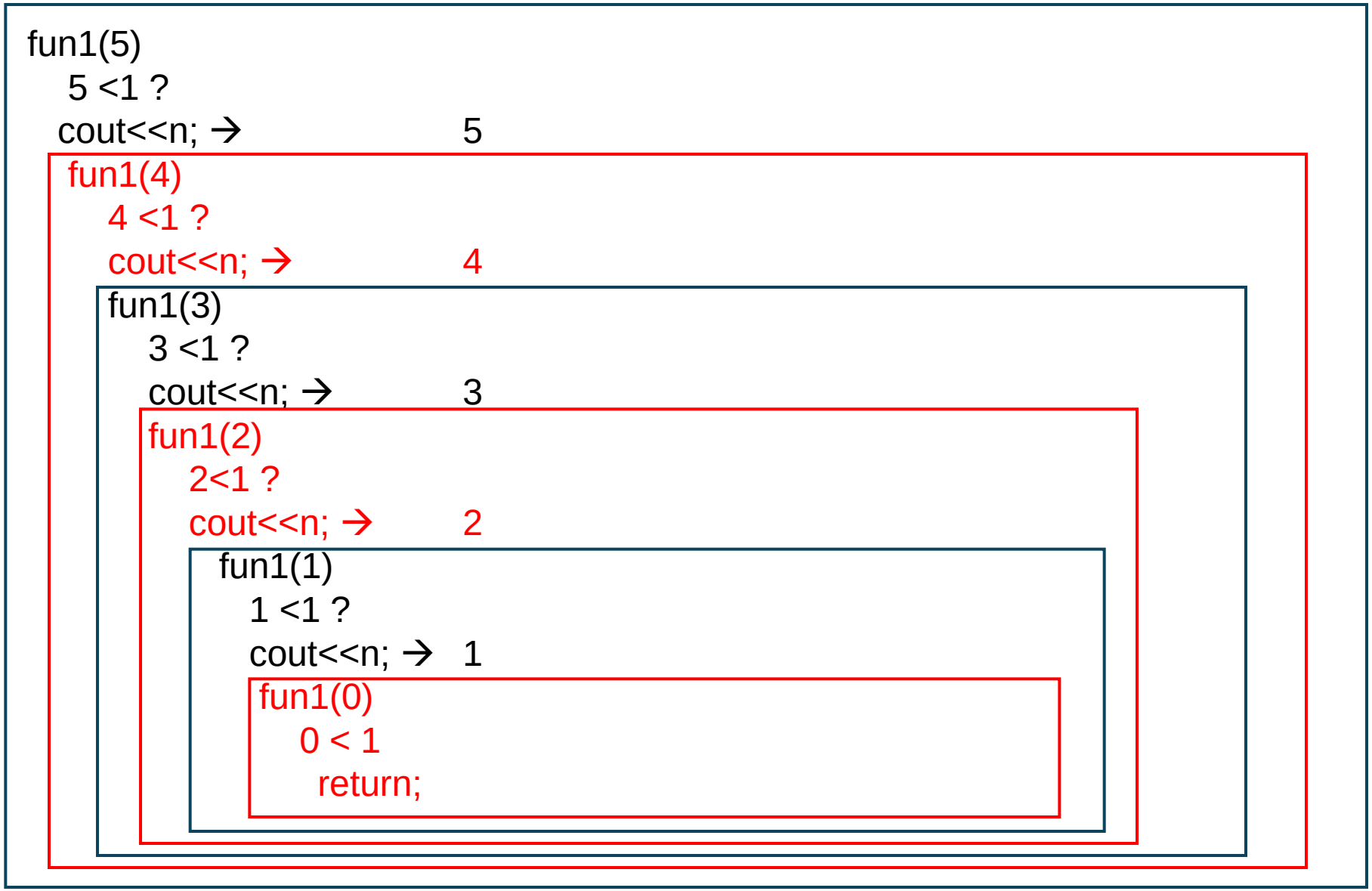
Recursion

```
void fun1(int n)
  {
        if (n<1)
              return;
           cout<<n;
        fun1(n-1);
}
```

What would be the output of fun1(5)?

Recursion unrolled

What would be the output of fun1(5)?



Recursion

- Understanding the flow of statements in a recursion call is very important.

- Let us re-visit the code again and write cout after the function calls itself

Recursion with Example fun2

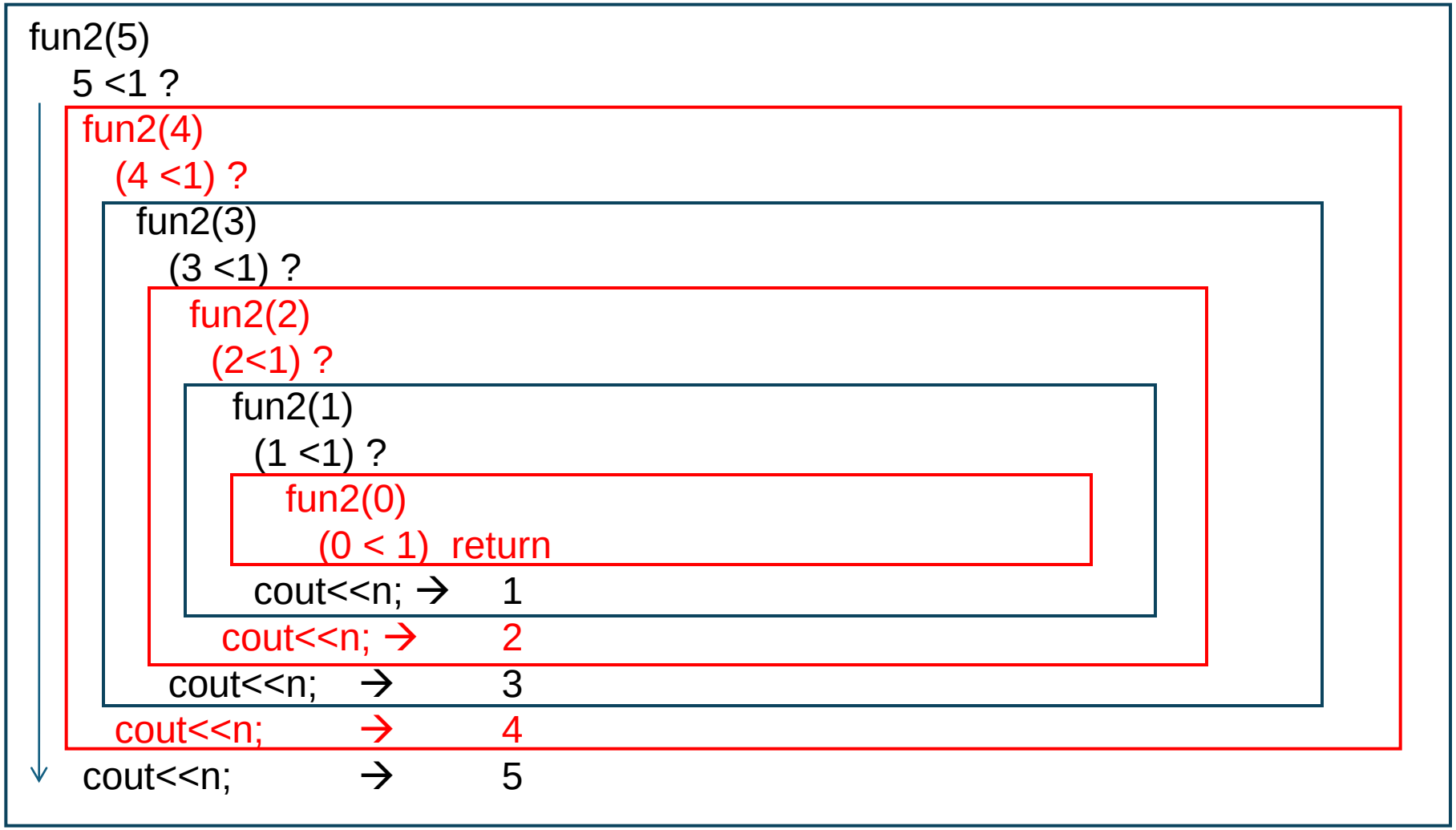
```
void fun2(int n)
        {
        if (n<1)
                return;
        fun2(n-1);
         cout<<n;
      }
```

What would be the output of fun2(5)?

Recursion unrolled

What would be the output of fun2(5)?

```
void fun2(int n){
    if (n<1) return;
    fun2(n-1);
    cout<<n;
}
```



Arrays

- **Array is a collection of elements with**
  - **same name**
  - **same data type**

- **In memory, array occupies consecutive memory locations**

- **For example: There are 66 students in CS101 Section A and I want to store their marks as integer data type.**
  - **One possible approach:**

```
int MarksStu1, MarksStu2, MarksStu3, …. MarksStu66;
```

  - **Another solution:** `int MarksStu[66];`

Arrays

- Declaring arrays simplifies our declaration of variables
  - At a later stage, we may change the data type from int to float

- We use index or subscript to identify each element of array

- The number of elements in an array is called length of array (or sometimes size of array)

Uses of Arrays

- Array simplifies declaring a lot variables, inputting values and displaying the output values (large data stored with one name)
- The values stored in arrays can be sorted
- Searching on arrays can be easily applied

Types of Arrays

- One dimensional
- Two dimensional
- Multi-dimensional Array

Declaring Arrays

- **To declare an array, we need to specify:**
  - **Data type of array**
  - **Name of array**
  - **Number of elements**
  - **Examples**

```
int c[ 10 ];
float hi[ 3284 ];
```

- **Declaring multiple arrays of same type**
  - **Similar format as other variables**
  - **Example**

```
int b[ 100 ], x[ 27 ];
```

Initializing Arrays

- After declaring arrays, we need to initialize arrays

- This can be done either using cin command or by equal to operator

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

- If not enough initializers, rightmost elements become 0
- If too many initializers, a syntax error is generated

```
int n[ 5 ] = { 0 }
```

Sets all the elements to `0`

- If size omitted, the initializers determine it

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

  - 5 initializers, therefore `n` is a 5 element array

Initializing Arrays

- **If the size of array is omitted, the initializers determine it**

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

- **5 initializers, therefore `n` is a 5 element array of type integer**

Example 1

```
#include <iostream>
using namespace std;

int main()
{
        int marks[5];
        int i;
        for (i=0;i<5; i++)
        {
                cout<<"Please enter the value of marks for student "<<i<<" ";
                cin>>marks[i];
        }

        for(i=0;i<5; i++)
        {
                cout<<"The marks of student "<<i<<" are "<<marks[i]<<endl;
        }
        return 0;
}
```

Example 2

```
#include <iostream>
using namespace std;

int main()
{
        int marks[5];
        int i;
        for (i=0;i<5; i++)
        {
                cout<<"Please enter the value of marks for student "<<i+1<<" ";
                cin>>marks[i];
        }

        for(i=0;i<5; i++)
        {
                cout<<"The marks of student "<< i+1 <<" are "<<marks[i]<<endl;
        }
        return 0;
}
```

Strings as Arrays

- C++ has char data-type but not string data type
- char data type can only store one alphabet like S or q or R
- What to do if I want to store a word like apple, orange, GIKI in C++
- A string is an array of character elements
  - But wait, an array has fixed size (or length) and it must be specified in the beginning / initialization

Strings as Arrays

- All strings end with **`null`** (**`'\0'`**)
- Examples:

```
char string1[] = "hello";
char string1[] = { 'h', 'e', 'l', 'l', 'o',
                   '\0' };
```

- Subscripting is the same as for a normal array

`string1[ 0 ]` is **`'h'`**

`string1[ 2 ]` is **`'l'`**

- Input from keyboard

```
char string2[ 10 ];
cin >> string2;
```

- Takes user input
- Side effect: if too much text entered, data written beyond array (and not stored)

Example 3

```
#include <iostream>
using namespace std;

int main()
{
        char string1[]="GIKI";
        char string2[20];
        cout<<"Enter the value of  string2 ";
        cin>>string2;

        cout<<"The value of string1 is "<<string1<<endl;
        cout<<"The value of string2 is "<<string2<<endl;
        cout<<"Now displaying the value of string2 again "<<endl;
        for (int i=0; string2[i]!='\0'; i++)
                cout<<string2[i];

        cout<<endl<<" Finished Printing";

        return 0;
}
```

Passing Arrays to Functions

When passing arrays to functions, there could be 2 possibilities

1. Passing the whole array to function

(by reference – which is default case)

2. Passing an element of array

Note: Unlike other variables, which are passed to function by value (default), arrays are always passed by reference

Passing Arrays to Functions

1. Passing whole array to a function:

   Specify the name without any

   brackets

- To pass array **`myArray`** declared as

```
int myArray[ 24 ];
```

  to function **`myFunction`**, a function call would resemble

```
myFunction( myArray, 24 );
```

- Array size is usually passed to function

Passing Arrays to Functions

2. Individual array elements passed by call-by-value
   - pass subscripted name (i.e., **`myArray[ 3 ]`**) to function

Example 4

```
#include <iostream>
using namespace std;

void modifyArray(int [], int );
void modifyElement( int );



int main()
{
        int i, marks[5] = { 10, 11, 12, 13, 14 };
        for (i=0;i<5; i++)
                cout<<"The value of marks for student "<<i+1<<" is "<<marks[i]<<endl;

        cout<<"Now calling modifyArray function"<<endl;
        modifyArray(marks,5);
        cout<<"The values after modification Array function are "<<endl;

        for (i=0;i<5; i++)
                cout<<"The value of marks for student "<<i+1<<" is "<<marks[i]<<endl;

        cout<<"Now calling modifyElement function"<<endl;
        modifyElement(marks[3]);
```

Example 4

```
                for(i=0;i<5; i++)
                {
                                        cout<<"The marks of student "<<i<<" are "<<marks[i]<<endl;
                }
                return 0;
}

void modifyArray(int b[], int x)
{
                int j;
                for (j=0; j<x; j++)
                                b[j]=b[j]*2;
}


void modifyElement( int c)
{
                c=c*10;
}
```