Data Structures and Algorithms

Lecture 1: C

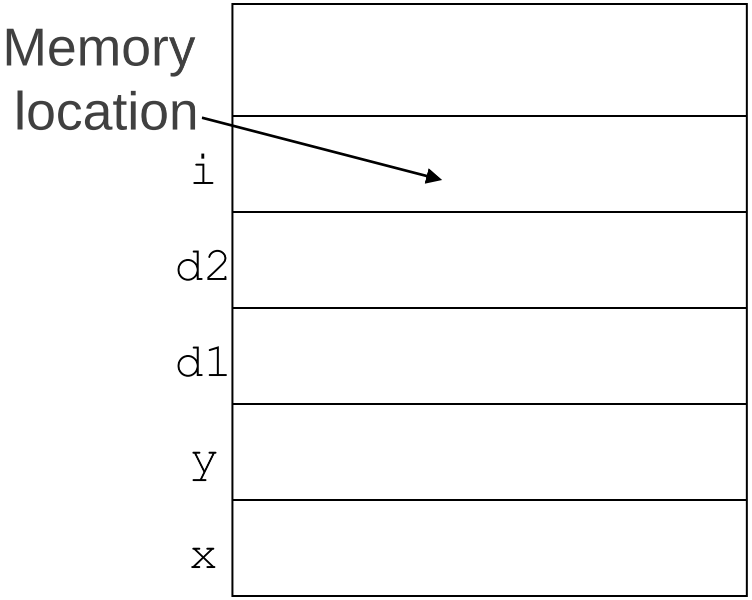
Department of Computer Science & Technology
United International College

Outline

- Functions and Memory
- Pointers
- Recursion

Functions & Memory

- Every function needs a place to store its local variables. Collectively, this storage is called the *stack*
- Each storage space has a numerical address
- Instead of using raw addresses, we use variables to attach a name to an address
- All of the data/variables for a particular function call are located in a *stack frame*

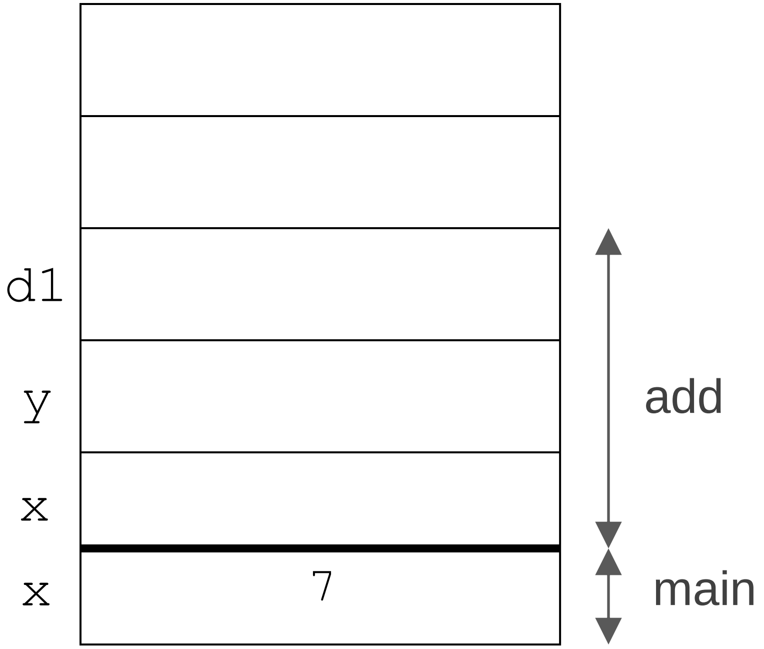


```
void aFunc(int x, int y)
{
   double d1, d2;
   int i;
}
```

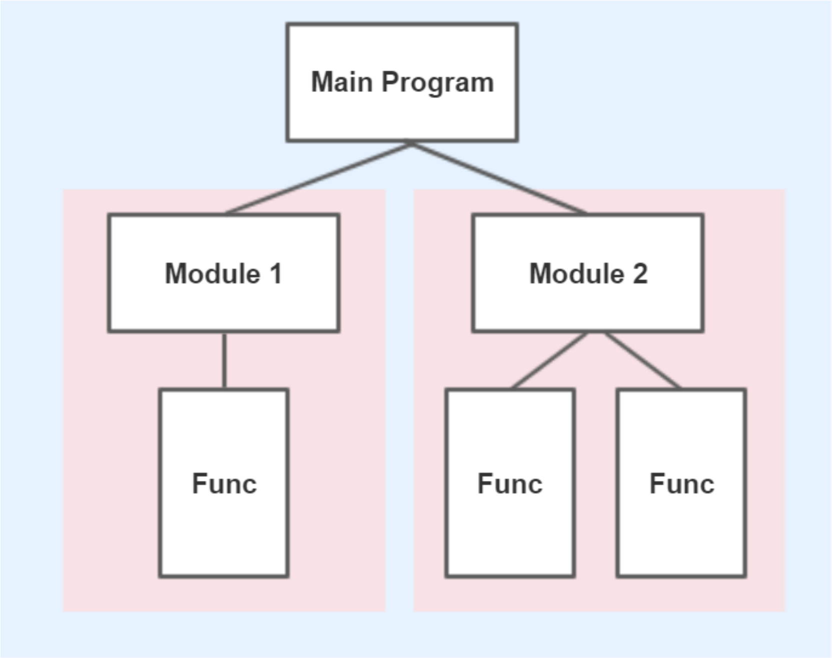
Functions & Memory (cont)

- When a function is called, a new stack frame is created
- Parameters and return values are passed *by copy* (ie, they're copied into and out of the stack frame)
- When a function finishes, its stack frame is reclaimed

```
void add(int x, int y) {
  double d1 = x + y;
}
int main() {
      int x = 7;
      add(1, 2);
      add(2, 3);
      return 0;
}
```



Programming Paradigm: Modular Concept

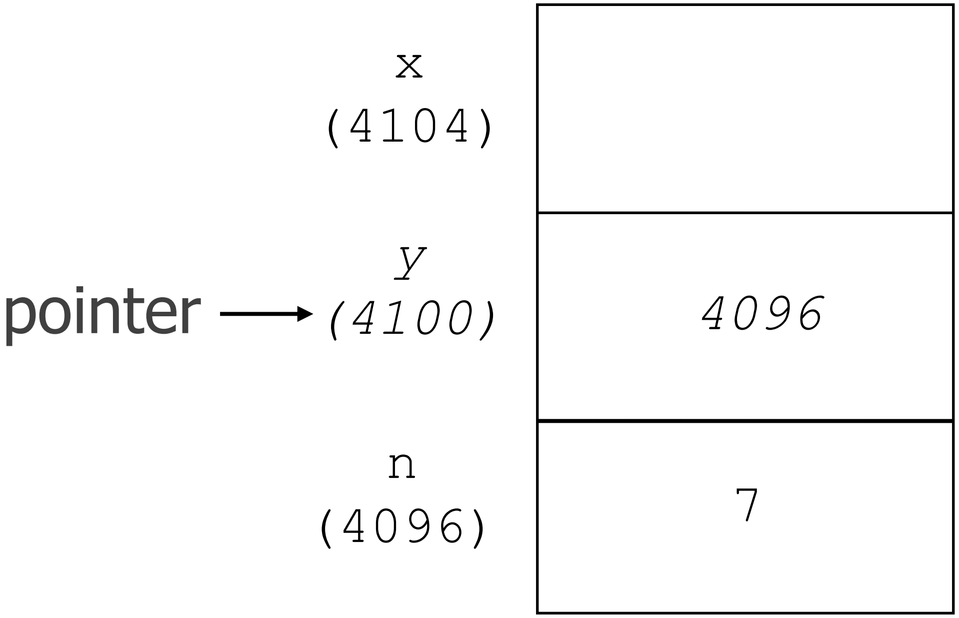


- The main program coordinates calls to procedures in separate modules and hands over appropriate data as parameters

Pointers

- A *pointer* is a variable which contains the address of another variable
- Accessing the data at the contained address is called “dereferencing a pointer” or “following a pointer”

```
int main(){
        int n = 7;
        int *y = &n;
        int x;
        return 0;
}
```



A Demonstration of Pointers

```
#include <stdio.h>
int main(){
 int* pc;
 int c;
 c=22;
 printf("Address of c:%u\n",&c);
 printf("Value of c:%d\n\n",c);

 pc=&c;
 printf("Address of pc:%u\n",pc);
 printf("Content of pc:%d\n\n",*pc);

 *pc=2;
 printf("Address of c:%u\n",&c);
 printf("Value of c:%d\n\n",c);
 return 0;}
```

The Output

```c
#include <stdio.h>
int main(){
 int* pc;
 int c;
 c=22;
 printf("Address of c:%u\n",&c);
 printf("Value of c:%d\n\n",c);

 pc=&c;
 printf("Address of pc:%u\n",pc);
 printf("Content of pc:%d\n\n",*pc);

 *pc=2;
 printf("Address of c:%u\n",&c);
 printf("Value of c:%d\n\n",c);
 return 0;}
```

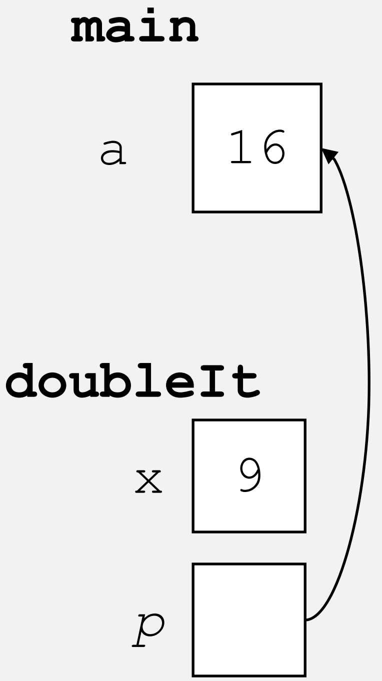
```
Address of c: 2686784
Value of c: 22

Address of pc: 2686784
Content of pc: 22

Address of c: 2686784
Value of c: 2
```

Pointers as Parameters

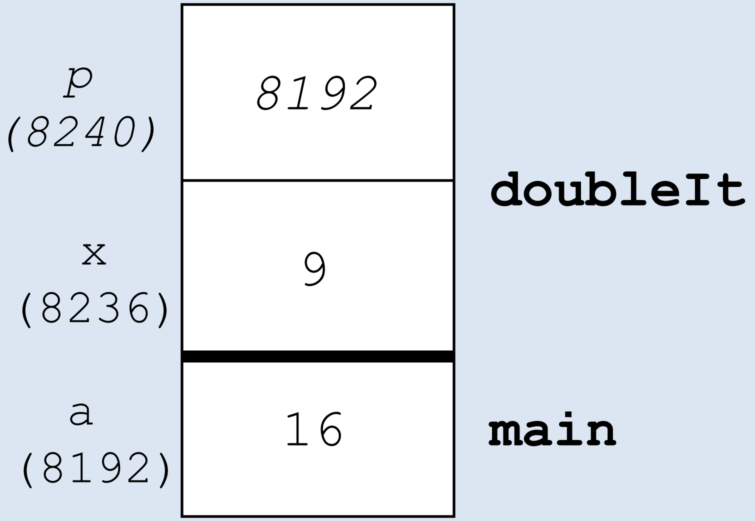
The code

```
void doubleIt(int x,
              int * p)
{
     *p = 2 * x;
}

int main()
{
     int a = 16;
     doubleIt(9, &a);
     return 0;
}
```

Box diagram



Memory Layout



a **gets** 18

The only way a function can access another function’s local variables.

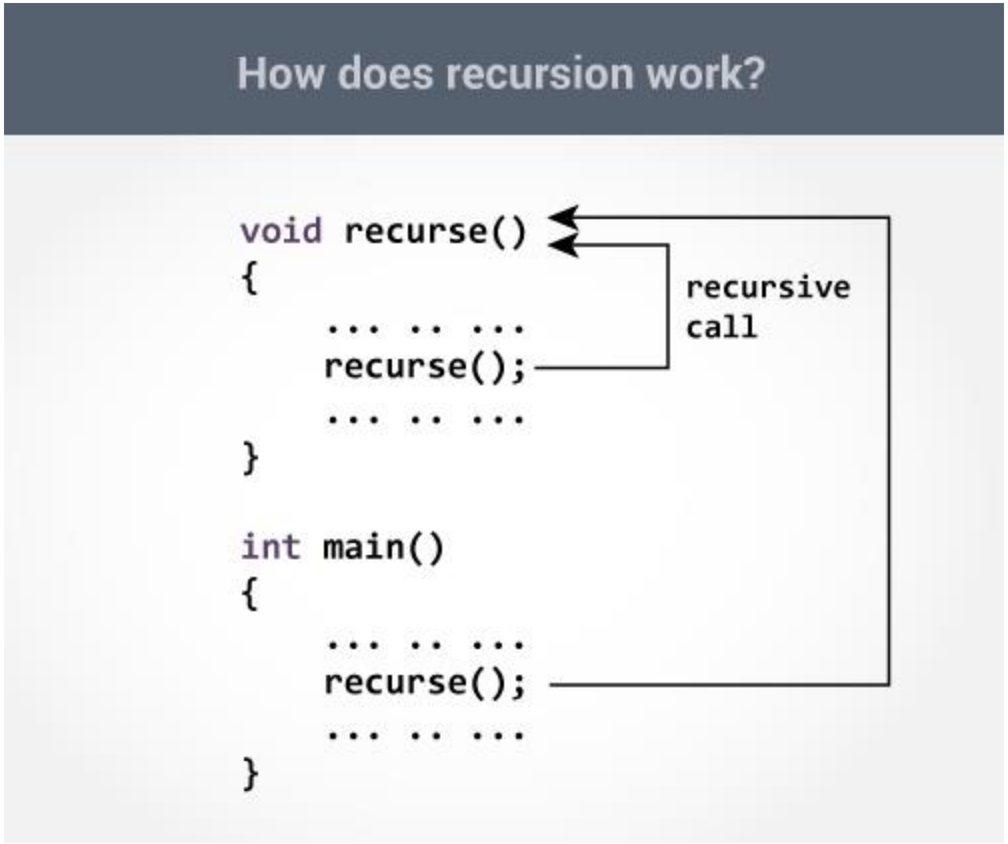
POINTERS AS PARAMETERS

Recursion

How Does Recursion Work?

- A function that calls itself is known as a recursive function.

```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```



How does recursion work?

```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```

Recursion

- The recursion continues until some condition (termination condition) is met.
- Always write the termination condition and make sure that the condition is reachable.
- Otherwise the recursion WILL NOT STOP!

Will this recursion stop?

```
#include <stdio.h>

int recurse(int i)
{
    return recurse i-1;
}

int main()
{
    int i;
    scanf("%d", &i);
    recurse(i);
    return 0;
}
```

Will this recursion stop?

```
#include <stdio.h>

int recurse(int i)
{
    if(i==0)
       return 0;
    return recurse i-1;
}

int main()
{
    int i;
    scanf("%d", &i);
    recurse(i);
    return 0;
}
```

Will this recursion stop?

```
#include <stdio.h>

int recurse(int i)
{
    if(i<=0)
       return 0;
    return recurse i-1;
}

int main()
{
    int i;
    scanf("%d", &i);
    recurse(i);
    return 0;
}
```

Recursion Example: Sum of Natural Numbers

- Sum(n) = 0 + 1 + 2 + … + (n-1) + n, for all n>=0
- Recursion build-up:
  - Base, if n=0: Sum(0)=0
  - Step, if n>0, Sum(n) = n + Sum(n-1)
- Any case will collapse to the base case step by step.

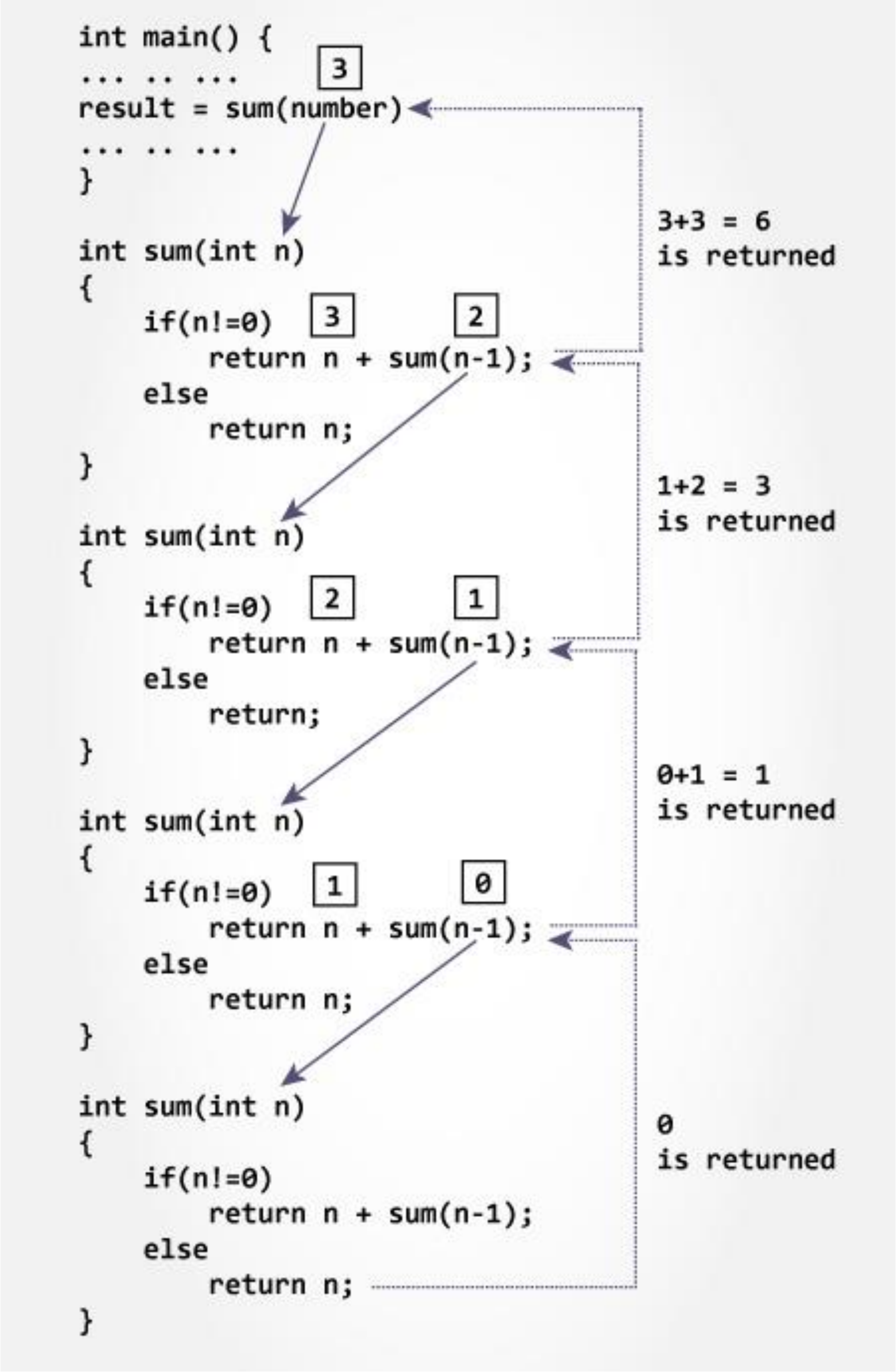
Recursion Example

```c
//Sum of Natural Numbers Using Recursion

#include <stdio.h>
int sum(int n);

int main()
{
    int number, result;

    printf("Enter a positive integer: ");
    scanf("%d", &number);

    result = sum(number);
    printf("sum=%d", result);
}

int sum(int num)
{
    if (num==0)
        return num;
    return num + sum(num-1);
    // sum() function calls itself
}
```

Output

```
Enter a positive integer:
3
6
```

Calling Stack



```
int main() {
... .. ...
result = sum(number)
... .. ...
}

int sum(int n)
{
    if(n!=0)
        return n + sum(n-1);
    else
        return n;
}

int sum(int n)
{
    if(n!=0)
        return n + sum(n-1);
    else
        return;
}

int sum(int n)
{
    if(n!=0)
        return n + sum(n-1);
    else
        return n;
}

int sum(int n)
{
    if(n!=0)
        return n + sum(n-1);
    else
        return n;
}
```

What is the problem of the example?

Task 1

- Read in a positive number and compute its factorial using recursion.
- Note that your program should contain
  - a main function, which does IO
  - and a recursive function, long int factR(int n), which computes the factorial

Task 1

- You may build you recursion as follows.
  - Base, if n=1: factR(1) = 1
  - Step, if n>1: factR(n) = n * factR(n-1)

Task 2

- Read in and reverse a string using recursion.
- Note that your program should contain
  - a main function, which does IO
  - and a recursive function as follows, which reverses the string
- *Note: You may assume that the string size is less than 50.*

```
void reverseR(int length, char *str)
{
    // length: the number of chars in *str
    // The chars in *str are reversed when
    // this fuction completes

}
```

Task 2

- You may build your recursion as follows.
  - Base, if the string size is 1 or 0: nothing is done.
  - Step, if the string size is at least 2: swap the first and last chars and reverse the remainder using recursion.

Task 3

- Read in and compute the greatest common divisor (GCD) of two natural numbers using recursion.
- GCD(x, y) is the greatest natural number which divides both x and y
  - GCD(6, 7) = 1
  - GCD(6, 9) = 3
  - GCD(6, 0) = 6
- Note that your program should contain
  - a main function, which does IO
  - and a recursive function, int GCD(int x, int y), which computes the GCD of x and y.

Task 3

- You can build your recursion as follows. Given x>=y,
  - Base, if y=0: GCD(x, 0) = x
  - Step, if y>0: GCD(x, y) = GCD(y, x % y)
- For example,
  - GCD(9, 6) = GCD(6, 3) = GCD(3, 0) = 3

Submission

- Save your .cpp files as t1.cpp, t2.cpp and t3.cpp, compress them into #####.zip and submit the zip file to iSpace.
- Note: ##### is your student ID.