ESc 101: Fundamentals of Computing

Lecture 26

Mar 10, 2010

Outline

1. Execution of a Recursive Function

Recursion

- A function is recursive if it is called inside its own definition.
- Such a definition is a substitute for loop, as in the `determinant` example.
- The execution jumps to the beginning of the function at the recursive call.
- To avoid infinite repetitions, it is necessary that:
    - in every successive call, some parameter value reduces,
    - and for small enough value of that parameter, there is no recursive call in the function.

Recursion

- A function is recursive if it is called inside its own definition.
- Such a definition is a substitute for loop, as in the `determinant` example.
- The execution jumps to the beginning of the function at the recursive call.
- To avoid infinite repetitions, it is necessary that:
  - in every successive call, some parameter value reduces,
  - and for small enough value of that parameter, there is no recursive call in the function.

Recursion

- A function is recursive if it is called inside its own definition.
- Such a definition is a substitute for loop, as in the `determinant` example.
- The execution jumps to the beginning of the function at the recursive call.
- To avoid infinite repetitions, it is necessary that:
  - in every successive call, some parameter value reduces,
  - and for small enough value of that parameter, there is no recursive call in the function.

Recursion

- A function is recursive if it is called inside its own definition.
- Such a definition is a substitute for loop, as in the `determinant` example.
- The execution jumps to the beginning of the function at the recursive call.
- To avoid infinite repetitions, it is necessary that:
  - in every successive call, some parameter value reduces,
  - and for small enough value of that parameter, there is no recursive call in the function.

Recursion

- A function is recursive if it is called inside its own definition.
- Such a definition is a substitute for loop, as in the `determinant` example.
- The execution jumps to the beginning of the function at the recursive call.
- To avoid infinite repetitions, it is necessary that:
  - in every successive call, some parameter value reduces,
  - and for small enough value of that parameter, there is no recursive call in the function.

Recursion

- A function is recursive if it is called inside its own definition.
- Such a definition is a substitute for loop, as in the `determinant` example.
- The execution jumps to the beginning of the function at the recursive call.
- To avoid infinite repetitions, it is necessary that:
  - in every successive call, some parameter value reduces,
  - and for small enough value of that parameter, there is no recursive call in the function.

A Simple Recursive Function

```
// Compare strings s and t
int strcmp_rec(char *s, char *t)
{
    if (*s != *t) // unequal strings
        return (int) (*s - *t);
    if (*s == ’\0’) // end of both s and t
        return 0; // they are equal!
    // s and t agree on first symbol, compare the rest
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n", strcmp_rec("Test", "Test"));
}
```

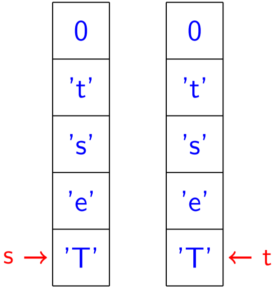
Execution of strcmp_rec()

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == ’\0’)
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

Execution of strcmp_rec()

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == ’\0’)
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

Execution of `strcmp_rec()`



0 | 0

't' | 't'

's' | 's'

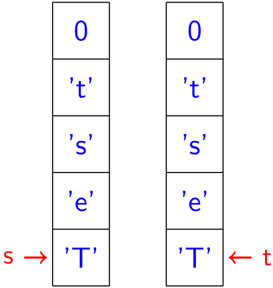
'e' | 'e'

s → 'T' | 'T' ← t

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

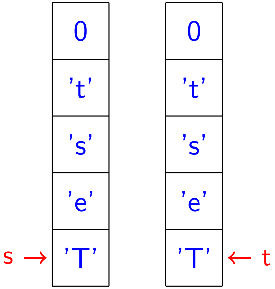
EXECUTION OF strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

Execution of strcmp_rec()



0 0

't' 't'

's' 's'

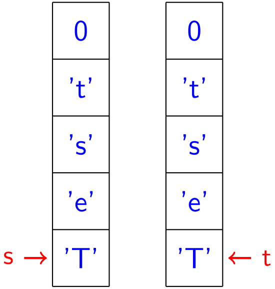
'e' 'e'

s → 'T' 'T' ← t

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

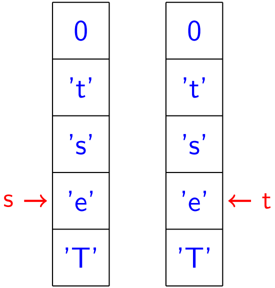
Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

Execution of strcmp_rec()

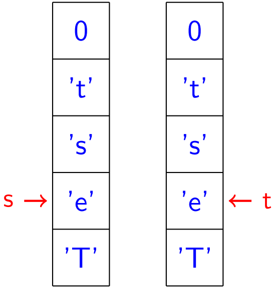


	0	0	
	't'	't'	
	's'	's'	
s →	'e'	'e'	← t
	'T'	'T'	

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == ’\0’)
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

Execution of strcmp_rec()



0 | 't' | 's' | 'e' | 'T'

s → 'e'

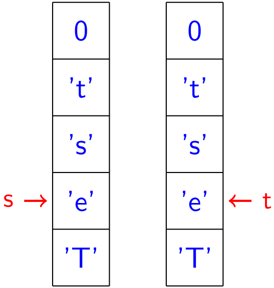
0 | 't' | 's' | 'e' | 'T'

'e' ← t

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

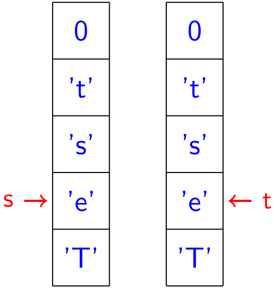
Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

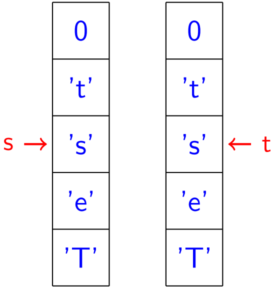
Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

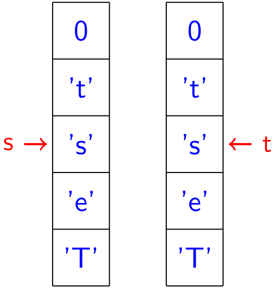
Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

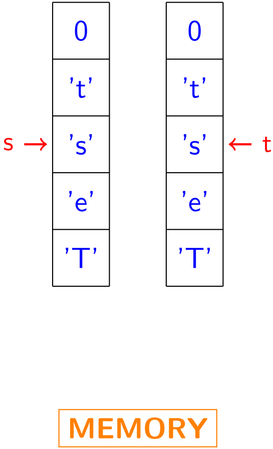
Execution of strcmp_rec()



MEMORY

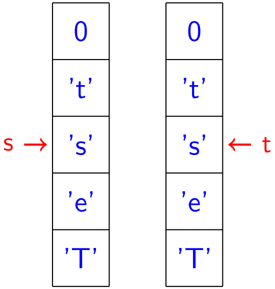
```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

EXECUTION OF strcmp_rec()



```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

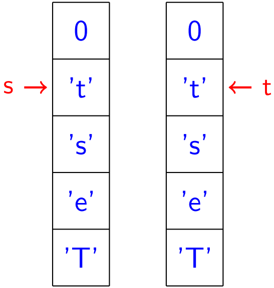
Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == ’\0’)
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

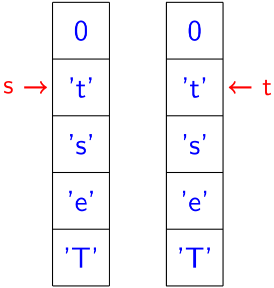
Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == ’\0’)
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

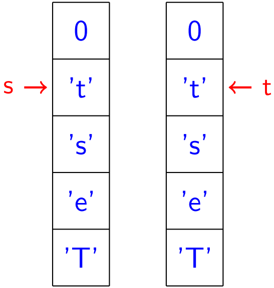
Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

Execution of strcmp_rec()

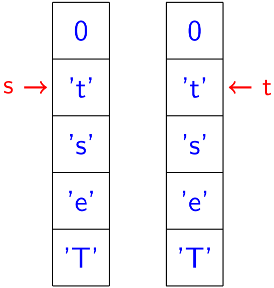


0	0
s → 't'	't' ← t
's'	's'
'e'	'e'
'T'	'T'

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == ’\0’)
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

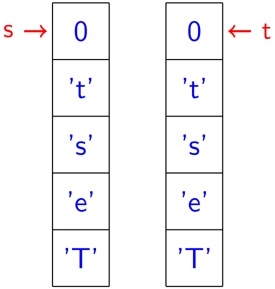
Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

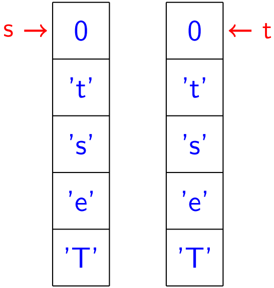
Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

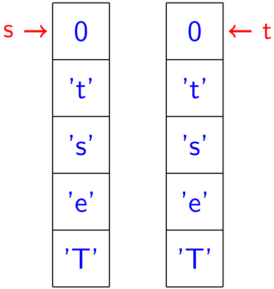
Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

EXECUTION OF strcmp_rec()



s → 0 | 0 ← t

't' | 't'

's' | 's'

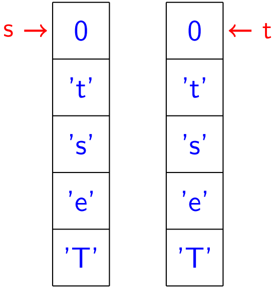
'e' | 'e'

'T' | 'T'

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

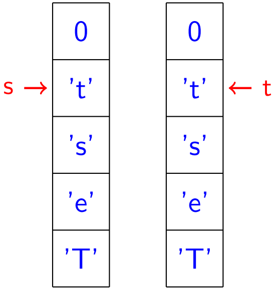
Execution of strcmp_rec()



```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == ’\0’)
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

MEMORY

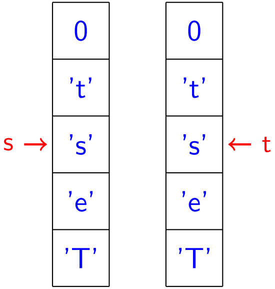
Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == ’\0’)
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

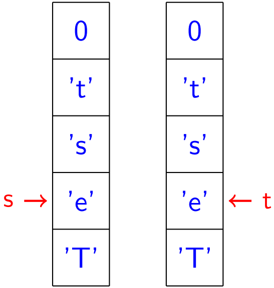
Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

Execution of strcmp_rec()

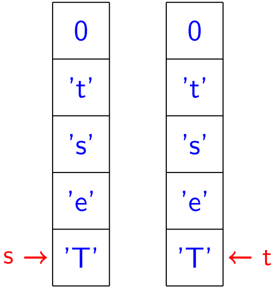


	0	0	
	't'	't'	
	's'	's'	
s →	'e'	'e'	← t
	'T'	'T'	

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

Execution of strcmp_rec()



0	0
't'	't'
's'	's'
'e'	'e'
s → 'T'	'T' ← t

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

Execution of strcmp_rec()

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Test"));
}
```

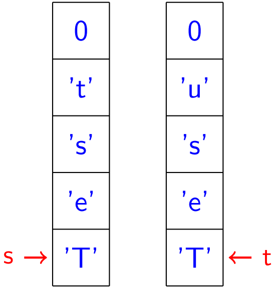
Another Execution of strcmp_rec()

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

Another Execution of strcmp_rec()

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

Another Execution of strcmp_rec()



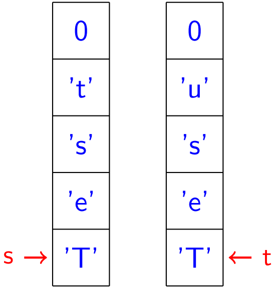
0
't'
's'
'e'
s → 'T'

0
'u'
's'
'e'
'T' ← t

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

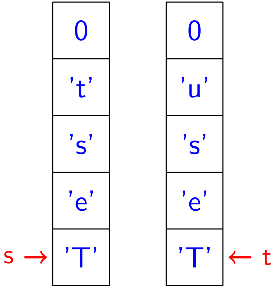
Another Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == ’\0’)
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

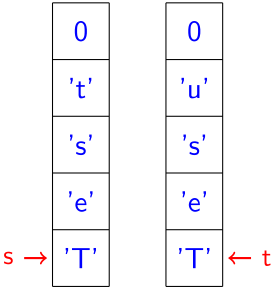
Another Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

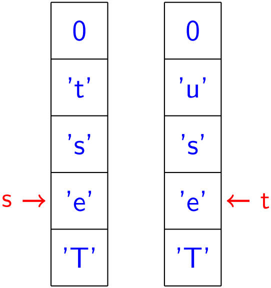
Another Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

Another Execution of strcmp_rec()



0 0

't' 'u'

's' 's'

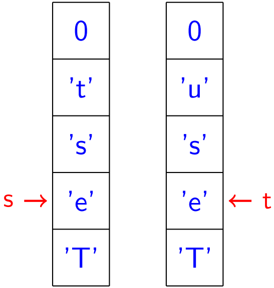
s → 'e' 'e' ← t

'T' 'T'

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
  printf("%d\n",
   strcmp_rec("Test", "Tesu"));
}
```

Another Execution of strcmp_rec()

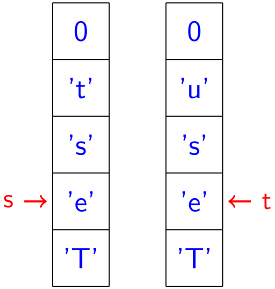


	0	0	
	't'	'u'	
	's'	's'	
s →	'e'	'e'	← t
	'T'	'T'	

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

Another Execution of strcmp_rec()

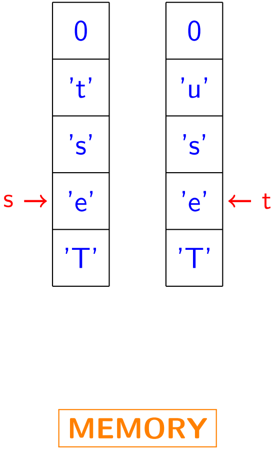


0 | 0
't' | 'u'
's' | 's'
s → 'e' | 'e' ← t
'T' | 'T'

MEMORY

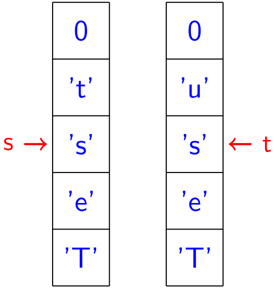
```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

Another Execution of strcmp_rec()



```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

ANOTHER EXECUTION OF strcmp_rec()



0 | 'u' | 's' | 'e' | 'T'

0 | 't' | 's' | 'e' | 'T'

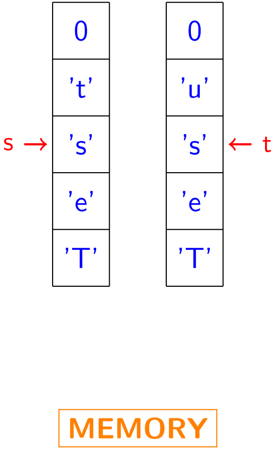
s → 's'

's' ← t

MEMORY

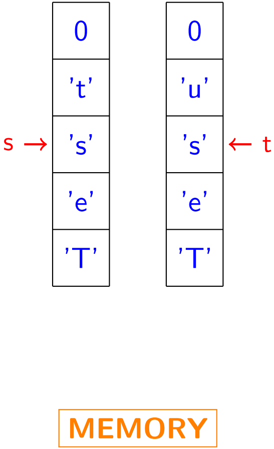
```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

Another Execution of strcmp_rec()



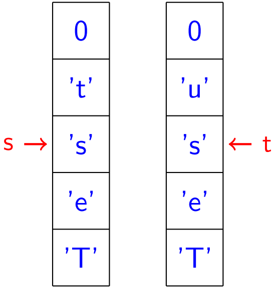
```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

Another Execution of strcmp_rec()



```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

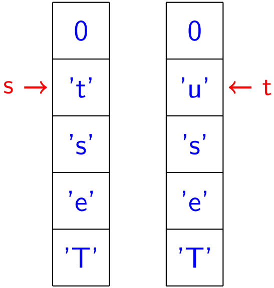
Another Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == ’\0’)
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

Another Execution of strcmp_rec()



0 | 0

s → 't' | 'u' ← t

's' | 's'

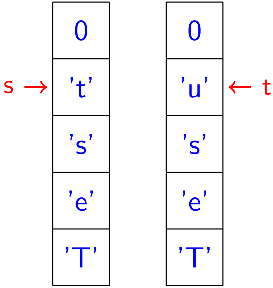
'e' | 'e'

'T' | 'T'

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

Another Execution of strcmp_rec()

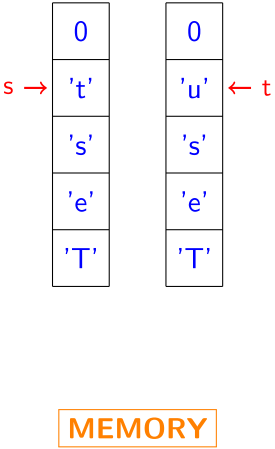


s	t
0	0
s → 't'	'u' ← t
's'	's'
'e'	'e'
'T'	'T'

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == ’\0’)
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

Another Execution of strcmp_rec()



0 0

s → 't' 'u' ← t

's' 's'

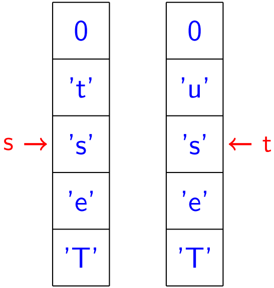
'e' 'e'

'T' 'T'

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

Another Execution of strcmp_rec()

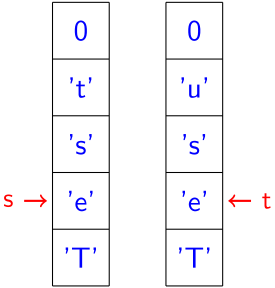


0 | 0
't' | 'u'
s → 's' | 's' ← t
'e' | 'e'
'T' | 'T'

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

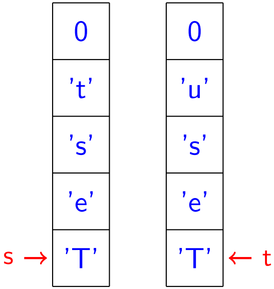
Another Execution of strcmp_rec()



MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

ANOTHER EXECUTION OF strcmp_rec()



0 0

't' 'u'

's' 's'

'e' 'e'

s → 'T' 'T' ← t

MEMORY

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == '\0')
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```

Another Execution of strcmp_rec()

```
int strcmp_rec(char *s, char *t)
{
    if (*s != *t)
        return (int) (*s - *t);
    if (*s == ’\0’)
        return 0;
    return strcmp_rec(s+1, t+1);
}

int main()
{
   printf("%d\n",
     strcmp_rec("Test", "Tesu"));
}
```