C programming language

College of Saint Benedict & Saint John's University

Dennis Ritchie in 2011 / CC BY 2.0

Brian Kernighan in 2012 / CC BY 2.0

```
/* file: helloworld.c */

#include <stdio.h>

int main() {
  printf("hello, world\n");
  return 0;
}
```

```
$ gcc -o helloworld helloworld.c
$ ./helloworld
hello, world
```

global variables

```c
// file: figure2-4.c
// Stan Warford
// A nonsense program to illustrate global variables

#include <stdio.h>

char ch;
int j;

int main() {
  scanf("%c %d", &ch, &j);
  j += 5;
  ch++;
  printf("%c\n%d\n", ch, j);
  return 0;
}
```

```
$ gcc -o figure2-4 figure2-4.c
$ ./figure2-4
M 419
N
424
```

3

```
5   #include <stdio.h>
6
7   char ch;
8   int j;
9
10  int main() {  <-----------------
11      scanf("%c %d", &ch, &j);
12      j += 5;
13      ch++;
14      printf("%c\n%d\n", ch, j);
15      return 0;  <----------------
16  }
```

C programs ALWAYS start execution with the **main** function

returning from **main** ends the program

global variables are
declared here —
outside of any function

characters in C are
treated internally
like signed integers

```c
#include <stdio.h>

char ch;
int j;

int main() {
    scanf("%c %d", &ch, &j);
    j += 5;
    ch++;
    printf("%c\n%d\n", ch, j);
    return 0;
}
```

program breakdown

```
5   #include <stdio.h>
6
7   char ch;
8   int j;
9
10  int main() {
11      scanf("%c %d", &ch, &j);
12      j += 5;
13      ch++;
14      printf("%c\n%d\n", ch, j);
15      return 0;
16  }
```

correct headers must be included to access library functions

read data from **stdin** (the terminal)

print data to **stdout** (the terminal)

scanf and **printf** are both library functions declared in **stdio.h**

```
5  #include <stdio.h>
6
7  char ch;
8  int j;
9
10 int main() {
11   scanf("%c %d", &ch, &j); ◄----
12   j += 5;
13   ch++;
14   printf("%c\n%d\n", ch, j);
15   return 0;
16 }
```

& is the address of operator — scanf expects the address of the variables where the data will be stored

```
1  if (<cond>) {
2    /* ... */
3  }
4  else (<cond>) {
5    /* ... */
6  }
7  else {
8    /* ... */
9  }
```

```
1  if (x) {
2    /* ??? */
3  }
4  if (x-y) {
5    /* ??? */
6  }
7  if (x=y) {
8    /* ??? */
9  }
```

- under what conditions will each of the above be executed?

```
switch (<expr>) {
  case <const>:
  /* ... */

  case <const>: /* fall-through */
  /* ... */
  break;

  default:
  /* ... */
  break;
}
```

```
1  for (<init>; <cond>; <incr>) {
2    /* ... */
3  }
4
5  while (<cond>)
6    /* ... */
7  }
8
9  do {
10   /* ... */
11 } while (<cond>);
```
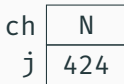
global variables

declared outside of any function and remain in place throughout the execution of the entire program. they are stored at a fixed location in memory.

local variables

declared within a function and come into existence when the function is called and cease to exist when the function terminates. they are stored on the run-time stack.

(a) Fixed location. (b) Run-time stack.

run-time stack a.k.a. "the stack"

run-time stack

stores information about the active functions of a C program, including:

- return value,
- actual parameters,
- return address, and
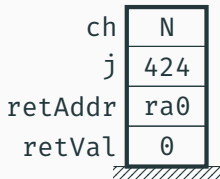- local variables

in that order.

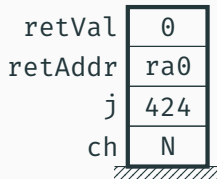## run-time stack a.k.a. "the stack"

run-time stack

stores information about the active functions of a C program, including:

- return value,
- actual parameters,
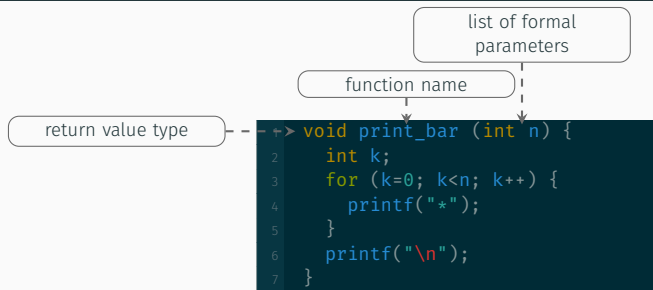- return address, and
- local variables

in that order.

ch	N		retVal	0
j	424		retAddr	ra0
retAddr	ra0		j	424
retVal	0		ch	N

       (a)   (b)

```
    void print_bar (int n) {
      int k;
      for (k=0; k<n; k++) {
        printf("*");
      }
      printf("\n");
    }
```

list of formal parameters

function name

return value type

list of formal
parameters

function name

return value type

```
1  void print_bar (int n) {
2      int k;
3      for (k=0; k<n; k++) {
4          printf("*");
5      }
6      printf("\n");
7  }
```
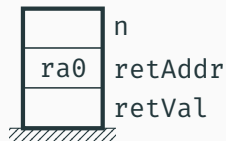
return value type

```
1  int fact(int n) {
2      int f, j;
3      f = 1;
4      for (j=1; j<=n; j++) {
5          f *= j;
6      }
7      return f;
8  }
```

type of **<expr>**
must match return
type of function

```c
int fact(int n) {
  int f, j;
  f = 1;
  for (j=1; j<=n; j++) {
    f *= j;
  }
  return f;
}

int main() {
  int n;
  scanf("%d", &n);
  printf("%d\n", fact(n)); // ra1
  return 0;
}
```

```
        | n
  ra0   | retAddr
        | retVal
```
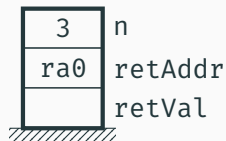
```c
5   int fact(int n) {
6     int f, j;
7     f = 1;
8     for (j=1; j<=n; j++) {
9       f *= j;
10    }
11    return f;
12  }
13
14  int main() {
15    int n;
16    scanf("%d", &n);
17    printf("%d\n", fact(n)); // ra1
18    return 0;
19  }
```
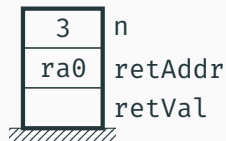
3	n
ra0	retAddr
	retVal

11

```
5   int fact(int n) {
6     int f, j;
7     f = 1;
8     for (j=1; j<=n; j++) {
9       f *= j;
10    }
11    return f;
12  }
13
14  int main() {
15    int n;
16    scanf("%d", &n);
17    printf("%d\n", fact(n)); // ra1
18    return 0;
19  }
```
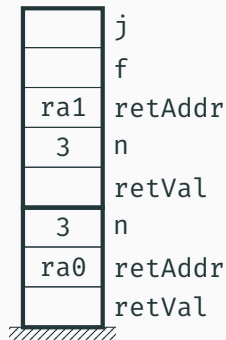
3	n
ra0	retAddr
	retVal

11

```c
5   int fact(int n) {
6     int f, j;
7     f = 1;
8     for (j=1; j<=n; j++) {
9       f *= j;
10    }
11    return f;
12  }
13
14  int main() {
15    int n;
16    scanf("%d", &n);
17    printf("%d\n", fact(n)); // ra1
18    return 0;
19  }
```
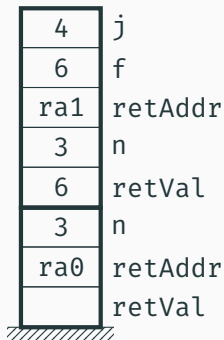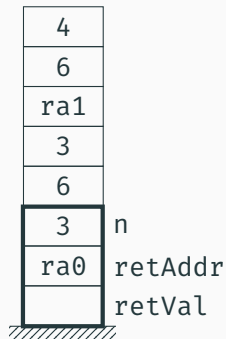
	j
	f
ra1	retAddr
3	n
	retVal
3	n
ra0	retAddr
	retVal

```c
int fact(int n) {
  int f, j;
  f = 1;
  for (j=1; j<=n; j++) {
    f *= j;
  }
  return f;
}

int main() {
  int n;
  scanf("%d", &n);
  printf("%d\n", fact(n)); // ra1
  return 0;
}
```

4	j
6	f
ra1	retAddr
3	n
6	retVal
3	n
ra0	retAddr
	retVal

```c
int fact(int n) {
  int f, j;
  f = 1;
  for (j=1; j<=n; j++) {
    f *= j;
  }
  return f;
}

int main() {
  int n;
  scanf("%d", &n);
  printf("%d\n", fact(n)); // ra1
  return 0;
}
```
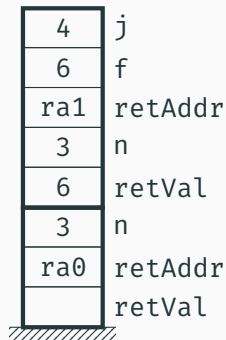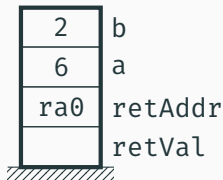
4	
6	
ra1	
3	
6	
3	n
ra0	retAddr
	retVal

```c
int fact(int n) {
  int f, j;
  // f = 1;
  for (j=1; j<=n; j++) {
    f *= j;
  }
  return f;
}

int main() {
  int n;
  scanf("%d", &n);
  printf("%d\n", fact(n));
  scanf("%d", &n);
  printf("%d\n", fact(n)); // ra1
  return 0;
}
```

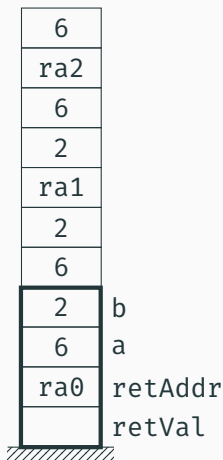|	
4	j
6	f
ra1	retAddr
3	n
6	retVal
3	n
ra0	retAddr
	retVal

functions — call-by-"reference"

```
5   void swap(int r, int s) {
6     int temp;
7     temp = r;
8     r = s;
9     s = temp;
10  }
11
12  void order(int x, int y) {
13    if (x > y) {
14      swap(x, y);
15    } // ra2
16  }
17
18  int main() {
19    int a, b;
20    scanf("%d %d", &a, &b);
21    order(a, b);
22    printf("d %d\n", a, b); // ra1
23    return 0;
24  }
```

2	b
6	a
ra0	retAddr
	retVal

```
5   void swap(int r, int s) {
6     int temp;
7     temp = r;
8     r = s;
9     s = temp;
10  }
11
12  void order(int x, int y) {
13    if (x > y) {
14      swap(x, y);
15    } // ra2
16  }
17
18  int main() {
19    int a, b;
20    scanf("%d %d", &a, &b);
21    order(a, b);
22    printf("d %d\n", a, b); // ra1
23    return 0;
24  }
```
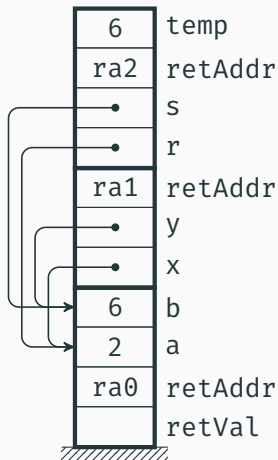
6	
ra2	
6	
2	
ra1	
2	
6	
2	b
6	a
ra0	retAddr
	retVal

12

```c
void swap(int *r, int *s) {
  int temp;
  temp = *r;
  *r = *s;
  *s = temp;
}

void order(int *x, int *y) {
  if (*x > *y) {
    swap(x, y);
  } // ra2
}

int main() {
  int a, b;
  scanf("%d %d", &a, &b);
  order(&a, &b);
  printf("d %d\n", a, b); // ra1
  return 0;
}
```

6	temp
ra2	retAddr
	s
	r
ra1	retAddr
	y
	x
6	b
2	a
ra0	retAddr
	retVal

pointers

- a pointer is a variable whose value is a memory address

```
1 int  i  = 0x1A;
2 int *ip = &i;
```

- &i evaluates to the address where the variable i is stored in memory
- i is an int, so ip is a *pointer* to an int

0x000012A0 $\boxed{00\,|\,00\,|\,00\,|\,1A}$ $\Big\}$i

0x???????? $\boxed{00\,|\,00\,|\,12\,|\,A0}$ $\Big\}$ip

13

```
printf("0x%X\n", i);    /* 0x1A */
printf("0x%#X\n", &i);  /* 0x12A0 */
printf("0x%#X\n", ip);  /* 0x12A0 */
printf("0x%#X\n", &ip); /* 0x???????? */
```

pointer dereference

- *ptr will
    1. treat the value of ptr as a memory address
    2. get the bytes of data located at that memory address
    3. interpret those bytes according to the type of pointer that ptr is

```
1 printf("0x%X\n", *ip);    /* 0x1A */
```

pointer dereference

- `*ptr` will
    1. treat the value of `ptr` as a memory address
    2. get the bytes of data located at that memory address
    3. interpret those bytes according to the type of pointer that `ptr` is

```
1  printf("0x%X\n", *ip);    /* 0x1A */
```

- `ip[X] = *(ip + X)`

```
1  printf("0x%X\n", ip[0]); /* 0x1A */
```

pointers cont.

```
printf("0x%X\n", i);        /* 0x1A */
printf("0x%X\n", *ip);      /* 0x1A */
printf("0x%X\n", ip[0]);    /* 0x1A */
printf("0x%X\n", *(ip+0));  /* 0x1A */
printf("0x%#X\n", &i);      /* 0x12A0 */
printf("0x%#X\n", ip);      /* 0x12A0 */
printf("0x%#X\n", &ip);     /* 0x???????? */
```

```
1  char *cp = "hello, world";
```

- cp is a *pointer* to a **char**

```
0x00004C80  | h | e | l | l | o | , |   | w | o | r | l | d | \0 |
```

```
0x????????  | 00 | 00 | 4C | 80 |
```

```
1  printf("%c\n", *cp);       /* h */
2  printf("%c\n", cp[0]);     /* h */
3  printf("%c\n", cp[4]);     /* o */
4  printf("%c\n", *(cp+4));   /* o */
5  printf("%s\n", cp);        /* hello, world */
6  printf("%s\n", cp+7);      /* world */
7  printf("0x%#X\n", cp);     /* 0x4C80 */
8  printf("0x%#X\n", &cp);    /* 0x???????? */
```
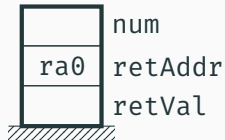
```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra1
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```
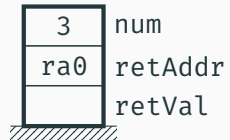
```
$ ./figure2-22
```



```
        ┌──────┐ num
        │      │
        ├──────┤
   ra0  │      │ retAddr
        ├──────┤
        │      │ retVal
        └──────┘
       ////////
```

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

```
$ ./figure2-22
Enter a small integer: 3
```

3	num
ra0	retAddr
	retVal

recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```
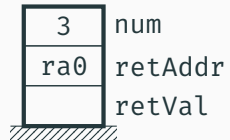
```
$ ./figure2-22
Enter a small integer: 3
```

3	num
ra0	retAddr
	retVal

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```
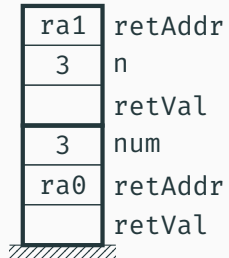
```
$ ./figure2-22
Enter a small integer: 3
```


ra1	retAddr
3	n
	retVal
3	num
ra0	retAddr
	retVal

recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```
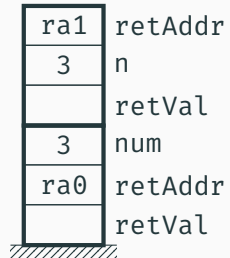
```
$ ./figure2-22
Enter a small integer: 3
```

ra1	retAddr
3	n
	retVal
3	num
ra0	retAddr
	retVal

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```
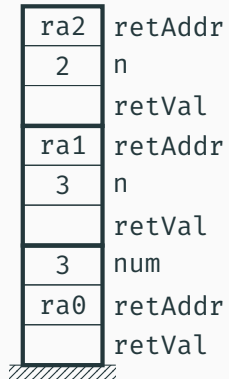
```
$ ./figure2-22
Enter a small integer: 3
```

ra2	retAddr
2	n
	retVal
ra1	retAddr
3	n
	retVal
3	num
ra0	retAddr
	retVal

18

recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```
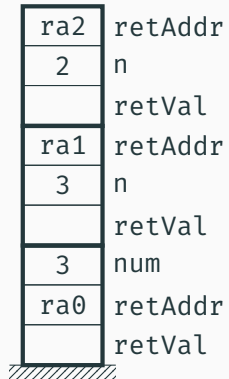
```
$ ./figure2-22
Enter a small integer: 3
```

ra2	retAddr
2	n
	retVal
ra1	retAddr
3	n
	retVal
3	num
ra0	retAddr
	retVal

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```
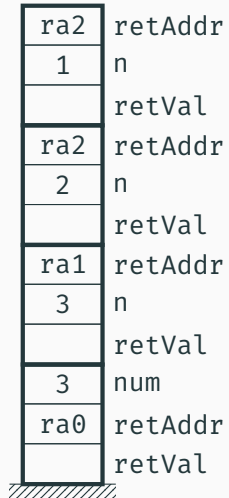
```
$ ./figure2-22
Enter a small integer: 3
```


ra2	retAddr
1	n
	retVal
ra2	retAddr
2	n
	retVal
ra1	retAddr
3	n
	retVal
3	num
ra0	retAddr
	retVal

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

```
$ ./figure2-22
Enter a small integer: 3
```

ra2	retAddr
1	n
1	retVal
ra2	retAddr
2	n
	retVal
ra1	retAddr
3	n
	retVal
3	num
ra0	retAddr
	retVal

```
1   #include <stdio.h>
2
3   int fact(int n) {
4     if (n <= 1) {
5       return 1;
6     }
7     else {
8       return n * fact(n - 1); // ra2
9     }
10  }
11
12  int main() {
13    int num;
14    printf("Enter a small integer: ");
15    scanf("%d", &num);
16    printf("Its factorial is: ");
17    printf("%d\n", fact(num)); // ra1
18    return 0;
19  }
```
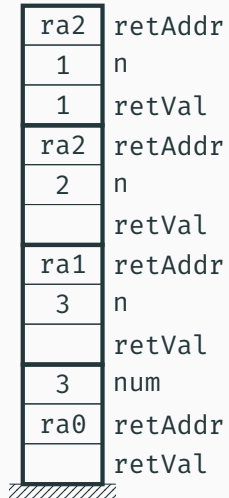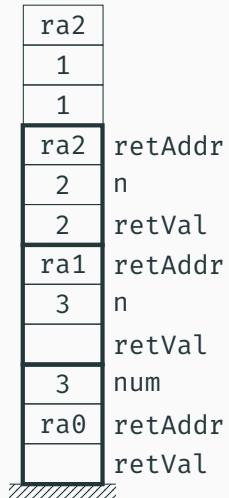
```
$ ./figure2-22
Enter a small integer: 3
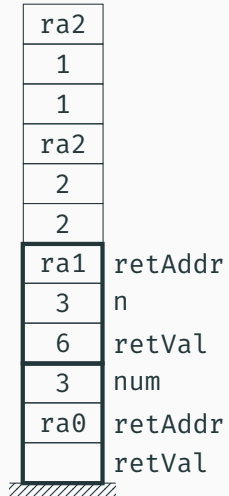```

| | |
|---|---|
| ra2 | |
| 1 | |
| 1 | |
| ra2 | retAddr |
| 2 | n |
| 2 | retVal |
| ra1 | retAddr |
| 3 | n |
| | retVal |
| 3 | num |
| ra0 | retAddr |
| | retVal |

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```
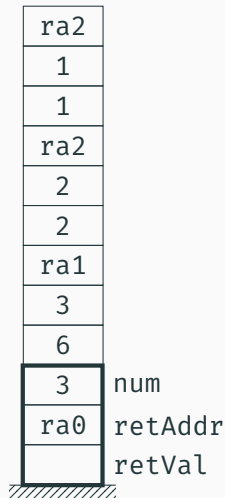
```
$ ./figure2-22
Enter a small integer: 3
```


ra2	
1	
1	
ra2	
2	
2	
ra1	retAddr
3	n
6	retVal
3	num
ra0	retAddr
	retVal

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

```
$ ./figure2-22
Enter a small integer: 3
Its factorial is: 6
```

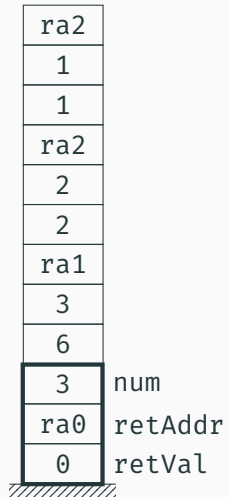| ra2 | |
1	
1	
ra2	
2	
2	
ra1	
3	
6	
3	num
ra0	retAddr
	retVal

18

recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

```
$ ./figure2-22
Enter a small integer: 3
Its factorial is: 6
```

ra2	
1	
1	
ra2	
2	
2	
ra1	
3	
6	
3	num
ra0	retAddr
0	retVal

18

recursion

```c
#include <stdio.h>

int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  else {
    return n * fact(n - 1); // ra2
  }
}

int main() {
  int num;
  printf("Enter a small integer: ");
  scanf("%d", &num);
  printf("Its factorial is: ");
  printf("%d\n", fact(num)); // ra1
  return 0;
}
```

```
$ ./figure2-22
...
$
```

- designate a block of memory to store value(s) of a particular data type

```
int * ip = malloc(100*sizeof(int));
```



```
ip [  •  ]———————→ [      ]
```

```
0x000063DA   A7 38 DC 91 0F F3 21 1E 76 4B AA 01  · · ·

0x????????   00 00 63 DA
```

- designate a block of memory to store value(s) of a particular data type

```
int * ip = malloc(100*sizeof(int));
```



```
ip
```

```
0x000063DA   A7 38 DC 91 0F F3 21 1E 76 4B AA 01 ···

0x????????   00 00 63 DA
```

- release a block of memory back to system to be used elsewhere

```
free(ip);
```

19

```
ip[0] = 0x7; /* *ip = 0x7; */
```

0x000063DA | 00 | 00 | 00 | 07 | 0F | F3 | 21 | 1E | 76 | 4B | AA | 01 | ···

0x???????? | 00 | 00 | 63 | DA |

```
ip[0] = 0x7; /* *ip = 0x7; */
```

0x000063DA | 00 | 00 | 00 | 07 | 0F | F3 | 21 | 1E | 76 | 4B | AA | 01 | $\cdots$

0x???????? | 00 | 00 | 63 | DA

```
ip[1] = 0xA; /* *(ip + 1) = 0xA; */
```

0x000063DA | 00 | 00 | 00 | 07 | 00 | 00 | 00 | 0A | 76 | 4B | AA | 01 | $\cdots$

```c
#include <stdio.h>

int main() {
  struct {
    char first;
    char last;
    int age;
    char gender;
  } bill;

  scanf("%c%c%d%c", &bill.first, &bill.last, &bill.age,
    &bill.gender);
  printf("Initials: %c%c\n", bill.first, bill.last);
  printf("Age: %d\n", bill.age);
  printf("Gender: ");
  if (bill.gender == 'f') {
    printf("fe");
  }
  printf("male\n");
  return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

struct { char *name; } bill;

char * find(char * str, char c) {
  for (; *str != c; str++);
  return str;
}

int main() {
  char *first, *last;

  first = malloc(100);
  scanf("%s%d", first);
  last = find(first, '-') + 1;
  last[-1] = '\0';
  printf("Initials: %c%c\n", first[0], *last);
  free(first);
  bill.name = malloc(20);
  printf("Full name: %s\n", bill.name);
  free(bill.name);

  return 0;
}
```

22

comparison

Java	C
object-oriented	procedural
interpreted	compiled
`String`	`char` array
condition (`boolean`)	condition (`int`)
garbage-collected	no memory management
references	pointers
exceptions	error codes