

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

hello, world

```
1 /* file: helloworld.c */  
2  
3 #include <stdio.h>  
4  
5 int main() {  
6     printf("hello, world\n");  
7     return 0;  
8 }
```

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

global variables

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

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

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 }
```

C programs ALWAYS
start execution with
the `main` function

returning from `main`
ends the program

program breakdown

global variables are
declared here —
outside of any function

characters in C are
treated internally
like signed integers

```
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 }
```

program breakdown

read data from
stdin (the terminal)

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

```
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

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

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 }
```

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

conditions

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

conditions

```
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

```
1  switch (<expr>) {  
2      case <const>:  
3          /* ... */  
4  
5      case <const>: /* fall-through */  
6          /* ... */  
7      break;  
8  
9      default:  
10         /* ... */  
11         break;  
12 }
```

loops

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

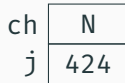
memory model — part i

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.

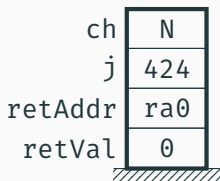
run-time stack a.k.a. “the stack”

run-time stack

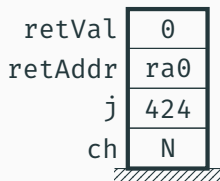
stores information about the active functions of a C program, including:

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

in that order.

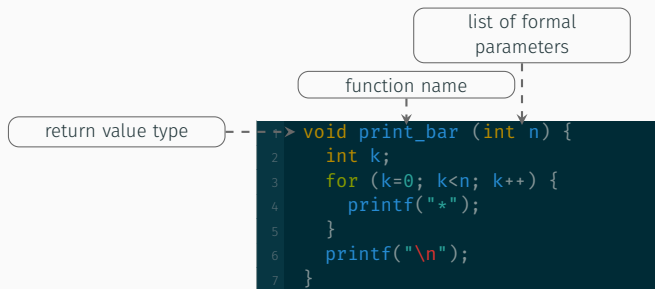


(a)



(b)

functions



functions

return value type

function name

list of formal parameters

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

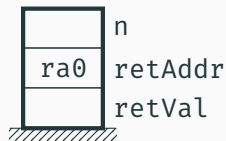
return value type

```
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

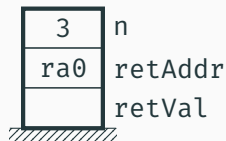
functions — call-by-value

```
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 }
```



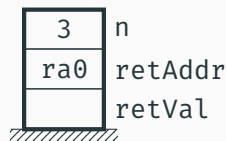
functions — call-by-value

```
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 }
```



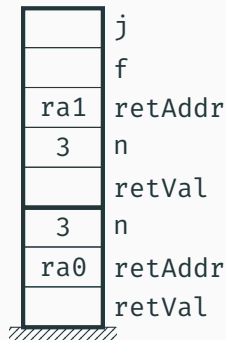
functions — call-by-value

```
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 }
```



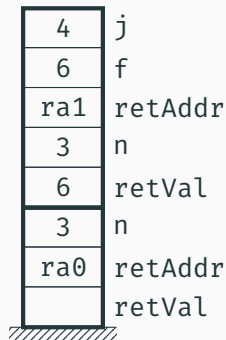
functions — call-by-value

```
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 }
```



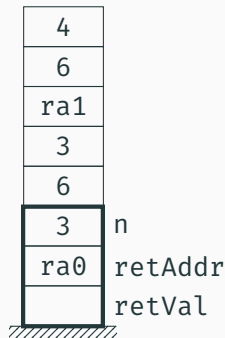
functions — call-by-value

```
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 }
```



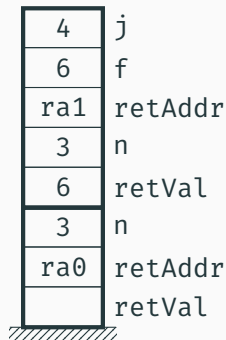
functions — call-by-value

```
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 }
```



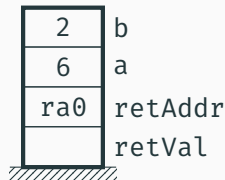
functions — call-by-value

```
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));  
18     scanf("%d", &n);  
19     printf("%d\n", fact(n)); // ra1  
20     return 0;  
21 }
```



functions — call-by-“reference”

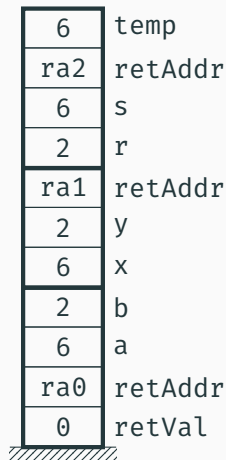
```
5 void swap(int r, int s) {  
6     int temp;  
7     temp = r;  
8     r = s;  
9     s = temp;  
10 }  
11  
12 void order(int x, int y) {  
13     if (x > y) {  
14         swap(x, y);  
15     } // ra2  
16 }  
17  
18 int main() {  
19     int a, b;  
20     scanf("%d %d", &a, &b);  
21    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



functions — call-by-“reference”

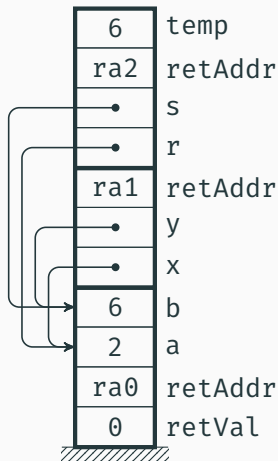
```
5 void swap(int r, int s) {  
6     int temp;  
7     temp = r;  
8     r = s;  
9     s = temp;  
10 }  
11  
12 void order(int x, int y) {  
13     if (x > y) {  
14         swap(x, y);  
15     } // ra2  
16 }  
17  
18 int main() {  
19     int a, b;  
20     scanf("%d %d", &a, &b);  
21    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

6	temp
ra2	retAddr
6	s
2	r
ra1	retAddr
2	y
6	x
2	b
6	a
ra0	retAddr
0	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 }
```



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



except where otherwise noted, this worked is licensed under creative commons attribution-sharealike 4.0 international license