

# 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>);
```

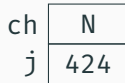
# memory model — part i

## 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.

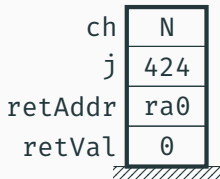
# run-time stack a.k.a. “the stack”

## run-time stack

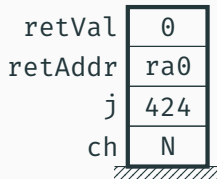
stores information about the active functions of a C program, including:

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

in that order.

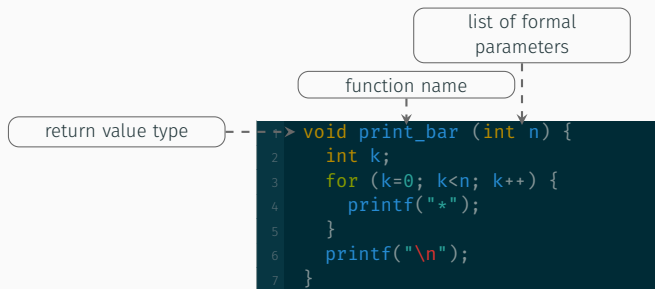


(a)



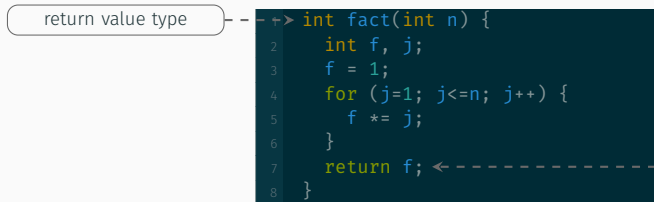
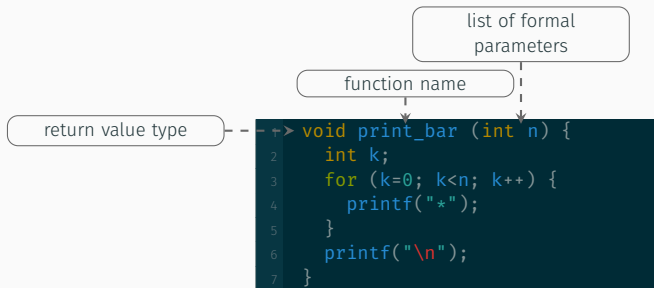
(b)

# functions





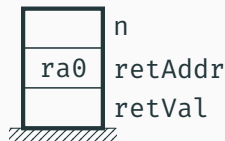
# functions



type of <expr>  
must match return  
type of function

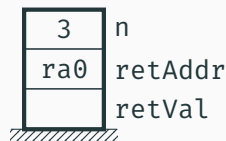
## functions — call-by-value

```
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 }
```



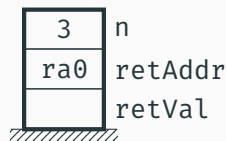
## functions — call-by-value

```
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 }
```



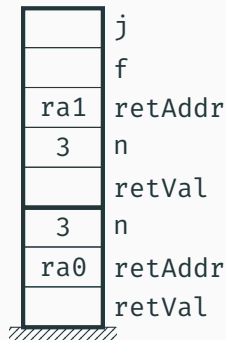
## functions — call-by-value

```
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 }
```



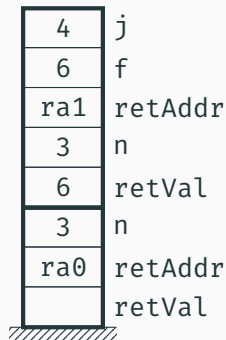
## functions — call-by-value

```
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 }
```



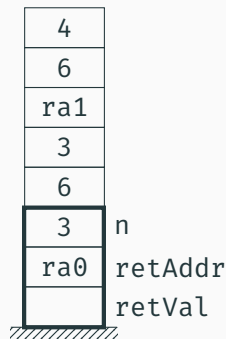
## functions — call-by-value

```
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 }
```



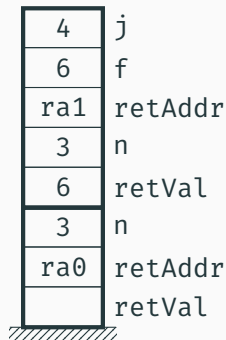
## functions — call-by-value

```
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

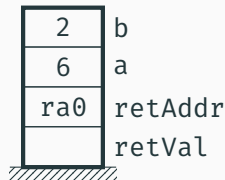
```
5  int fact(int n) {  
6      int f, j;  
7      // f = 1;  
8      for (j=1; j<=n; j++) {  
9          f *= j;  
10     }  
11     return f;  
12 }  
13  
14 int main() {  
15     int n;  
16     scanf("%d", &n);  
17     printf("%d\n", fact(n));  
18     scanf("%d", &n);  
19     printf("%d\n", fact(n)); // ra1  
20     return 0;  
21 }
```





## functions — call-by-“reference”

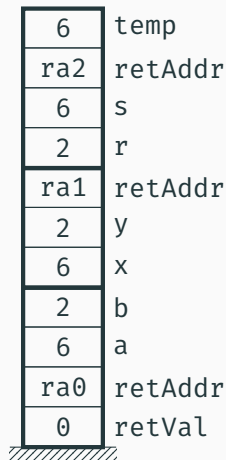
```
5 void swap(int r, int s) {  
6     int temp;  
7     temp = r;  
8     r = s;  
9     s = temp;  
10 }  
11  
12 void order(int x, int y) {  
13     if (x > y) {  
14         swap(x, y);  
15     } // ra2  
16 }  
17  
18 int main() {  
19     int a, b;  
20     scanf("%d %d", &a, &b);  
21    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



## functions — call-by-“reference”

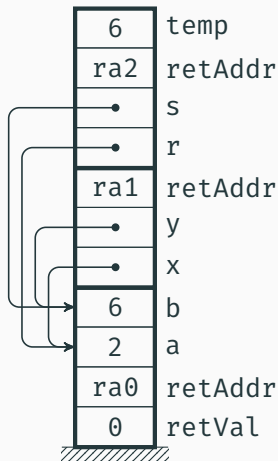
```
5 void swap(int r, int s) {  
6     int temp;  
7     temp = r;  
8     r = s;  
9     s = temp;  
10 }  
11  
12 void order(int x, int y) {  
13     if (x > y) {  
14         swap(x, y);  
15     } // ra2  
16 }  
17  
18 int main() {  
19     int a, b;  
20     scanf("%d %d", &a, &b);  
21    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



# 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   

00	00	00	1A
----	----	----	----

 } i

0x????????   

00	00	12	A0
----	----	----	----

 } ip

## pointers cont.

```
1 printf("0x%X\n", i);      /* 0x1A */
2 printf("0x%#X\n", &i);    /* 0x12A0 */
3 printf("0x%#X\n", ip);    /* 0x12A0 */
4 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.

```
1 printf("0x%X\n", i);           /* 0x1A */
2 printf("0x%X\n", *ip);         /* 0x1A */
3 printf("0x%X\n", ip[0]);       /* 0x1A */
4 printf("0x%X\n", *(ip+0));     /* 0x1A */
5 printf("0x%X\n", &i);          /* 0x12A0 */
6 printf("0x%X\n", ip);          /* 0x12A0 */
7 printf("0x%X\n", &ip);        /* 0x??????? */
```



## pointers cont.

```
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???????? */
```

# comparison

Java	C
object-oriented	procedural
interpreted	compiled
<b>String</b>	<b>char</b> array
condition ( <b>boolean</b> )	condition ( <b>int</b> )
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