

C programming language

Jeremy Iverson

College of Saint Benedict & Saint John's University

- Dennis Ritchie and Brian Kernighan creators of C circa 1972



Dennis Ritchie in 2011 / CC BY 2.0



Brian Kernighan in 2012 / CC BY 2.0

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

- in Java, everything is a method that is called on an object
- in C, everything is a function
- in Java, source code is compiled to byte code, which is then interpreted by Java VM
- in C, source code is compiled into binary machine code
- in Java, String is a class
- in C, a string is just an array of **char** values which ends with the **char** `'\0'`
- in Java, the Java VM takes care of deallocating memory used
- in C, any memory you allocate, you must also deallocate

hello, world

```
1 #include <stdio.h>
2
3 int main() {
4     printf("hello, world\n");
5     return 0;
6 }
```

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

- The tradition of using the phrase "Hello, world!" as a test message was influenced by an example program in the seminal book *The C Programming Language*

conditions

- under what conditions will each of the following be execute?

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

- $x \neq 0$

- $x \neq y$

- $y \neq 0$

add evens

- create program called `add_even.c` that adds all the even numbers between 1 and 100 and prints the sum

```
1 #include <stdio.h>
2
3 int main(int argc, char * argv[]) {
4     printf("(%d) %s:%s\n", argc, argv[0],
5         argv[1]);
6     return 0;
7 }
```

- modify `add_even.c` to get maximum value from the command-line instead of hard-coded as 100

printf / scanf

- `printf()` interprets variables and prints character representations to standard out (usually the terminal)
- `scanf()` scans characters from standard in (usually the terminal) and interprets them for storage in variables

```
1 #include <stdio.h>
2
3 int main() {
4     int i;
5     scanf("%d", &i);
6     return 0;
7 }
```

- `scanf` requires you to pass the address of the variable, so that its value can be changed

- modify `helloworld.c` to ask user for an input and then print it back
- change its name to `echo.c`

```
$ ./echo  
Enter a string to echo: hello, world  
hello,
```

- why did it print `hello,` instead of `hello, world`?

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

- so how can we use the pointer, `ip`, to access the value of `i`?

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 */
```

- the C compiler is "smart enough" to "know" that `+ X` really means add `X * sizeof(*ip)` to `ip`

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 */
```

- the C compiler is "smart enough" to "know" that `+ X` really means add `X * sizeof(*ip)` to `ip`

```
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.

- why not say `cp` is a *pointer* to a `char` array?

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

```
1 #include <stdio.h>
2
3 void swap(int n1, int n2) {
4     int tmp = n1;
5     n1 = n2;
6     n2 = tmp;
7 }
8
9 int main() {
10     int v1 = 11, v2 = 77;
11     printf("BEFORE  v1=%d, v2=%d\n", v1, v2);
12     swap(v1, v2);
13     printf("AFTER   v1=%d, v2=%d\n", v1, v2);
14     return 0;
15 }
```

- what's wrong with this program?
- fix the program so that it correctly swaps the two variables' values

heap memory

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

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

0x000063DA | r | @ | ! | X | t | v | 9 | 1 | S | ? |) | . | ...

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

- allocates enough consecutive memory for 100 `int` values

heap memory

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

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

0x000063DA | r | @ | ! | X | t | v | 9 | 1 | S | ? |) | . | ...

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

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

```
1 free(ip);
```

- allocates enough consecutive memory for 100 `int` values

heap memory cont.

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

0x000063DA | 00 | 00 | 00 | 07 | t | v | 9 | 1 | S | ? |) | . | ...

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

heap memory cont.

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

0x000063DA | 00 | 00 | 00 | 07 | t | v | 9 | 1 | S | ? |) | . | ...

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

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

0x000063DA | 00 | 00 | 00 | 07 | 00 | 00 | 00 | 0A | S | ? |) | . | ...

```
1 #include <stdio.h>
2
3 int main(int argc, char * argv[]) {
4     int m, n;
5     FILE * fp;
6
7     if (!(fp = fopen("example.txt", "r")))
8         return -1;
9     if (2 != fscanf(fp, "%d %d", &m, &n))
10        return -1;
11    if (!fclose(fp))
12        return -1;
13
14    return 0;
15 }
```

- matrices



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