Advanced C Programming

Rodrigo Rodrigues

12-08-24

1 Examples

size of

```
int main(){
   if (sizeof(int) > -1)
      printf("True\n");
   else
      printf("False\n");
}
```

Output: **False**

Int has a size of 4, but comparing only works on the same data type. Sizeof(int) will be unsigned int, while -1 is a signed int. So by treating -1 as an unsigend value -1=0xFFFFFFFF.

float vs double

```
int main(){
   float f = 0.1;
   if( f == 0.1)
      printf("True\n");
   else
      printf("False\n");
}
```

Output: **False**

Double data-type has more precision than float because C is oriented to work with doubles. Float as 6 degress and double has 10 degrees. If we treat f as a double, f will be converted into a double, and its not going to havethe same value.

Sizeof2

```
int main(){
   int a, b = 1, c = 1;

   a = sizeof(c = ++b + 1);
   printf("a = %d, b = %d, c = %d\n, a, b, c");
}
```

Output: **a=4, b=1, c=1**

Sizeof is a runtime compilation, **a** is going to change, but not **b** or **c**.

charptr

```
int main(){
   char *p = 0;

   *p = 'A';
   printf("Value at p = %c\n", *p);
}
```

Output: **Segmentation fault (core dumped)**

We initialize a pointer in the memory address 0, which is read only. When we try to write A to it we get a segmentation fault.

nested if

```
int main(){
   int a = 1, b = 3, c = 2;
   if (a>b)
      if(b>c)
         printf("a>b and b > c\n");
   else
      printf("something else\n");
}
```

Output:

Nothing will be printed, the else statement bellongs to the second if.

2 Compiler

C is a robust language (kernel is writen in C i think) and we can write everything from simple to complex program.

C compiller allows o write both assembly language programming allong side C, allowing you to write both system software and application software.

Programming in C is efficient and fast. Compiling it produces object code which is linked by linker, and a binary executable code is generated.

The **main()** function is the entry point.

Any **variable** needs to be written in lowercase letters.

3 Data Types

3.1 Primitive type

Char is a byte (8bits). if we give char = 'A', we are not storing A, we store 65, 0b010001

1. char - 1 bytes (8 bits).
2. short (short integer) - 2 bytes (16 bits).
3. int - 4 bytes (32 bits).
   (a) for 32 enviroment
4. long (store integer) - 4 bytes (32 bits). Depends on the platform.
   (a) its for 64 enviroment
   (b) short, int and long all store integers, but for diferent objectives
5. float - 4 bytes (32 bits) 6 digits of precision
6. long long - 8 bytes (64 bites) 32 bit enviroment;
   C compiler unnderstands it.
   we can store 4GB

   sometimes we have 100Gb, 200GB or 300TB in our system, so:
   int = 4 byte
   long = 4 byte
   none of this wew enoughf
   long long was necessary
   long long = 8 bytes

   In a 64 enviroment, it's just a long.
7. double - 8 bytes (64 bits).
   (a) for 32 processors
   (b) removes the dependency from shot, int, long
   (c) to store more precision (10 digits of precision) compared to float.

8. void - 4 bytes (it means nothing, we do not alocate memory) Created for pointers.
   (a) created for functions that need to return nothing
   (b) usefull for functions that return something we dont know
   (c) for example malloc() returns a piece of memory (pointer [has no data type])

   Integral, real and character types can be signed or unsigned

3.2 Example

Which data type is most suitable to store size of a hard disk partition (10 GB) ?

example of memory alocation

```
// WARNING: wrong
int sizeofdisk;
// WARNING: wrong
unsigned int;
// Correct
unsigned long;
sizeofdisk = 10GB = 10 * 1024 * 1024 * 1024 * 1024
```

GB = * 1024 * 1024 * 1024 * 1024
1 KB = 1024 B; = 2^{10}
1 MB = 1024 KB;
1 GB = 1024 MB; 8 000 000 000 bites

integer: 4 bytes: 32-bits: max value = $2^32 - 1$

(msb) 11111111 11111111 11111111 11111111 (lsb)

$$
\begin{aligned}
2^{32} &= 2^2 * 2^{30} && (1)\\
&= 4 * 2^{10} * 2^{10} * 2^{10} && (2)\\
&= 4 * 1024 * 1024 * 1024 && (3)\\
&= 4 * KB * KB * KB && (4)\\
&= 4GB && (5)
\end{aligned}
$$

What data type to store data of 8-bit microprocessor?

Unsigned Char - 8 bits

it doesnt make sense to send a negative number to a controler
for exmaple, so we dont use char.
0000 0000 - 2^8 functions (32)

Which data type to store the length of a string (C strlen() function) ?

unsigned long - i guess not (but close)

(a) can it be negative ?
no

(b) how big its it?
R.: unsigned int / unsigned long i dont know

strlen() returns 'size_t', which is not a primitive type. we use 'typedef'

defining size of an object

```
typedef unsigned long size_t;
```

3.3 User defined type

1. Defined by Programmer
2. structure
3. union
4. enumeration
5. typedef

```
struct empdb {
  int empno;
  char name[30];
  int dept_id;
};

union u1 {
  int a;
  int b;
};

struct empdb emp;
```

dont define a structure as a union, and dont define a union as a structure, they are for diferent things

The defined type will hold the memory for the biggest type, which is int. R.: 4 bytes

Union is for something that has "versions", more than one choice, i guess

code

```c
#include <stdio.h>
struct empdb {
  int empno;
  char name[30];
  int dept_id;
};

union u1 {
  int a;
  int b;
};

int main(){

  long long l;
  long sl;

  l = 0xAABB CCDD EEFF 00;
  // in a 32 bit enviroment long long is meaninfull
  sl = 0xAABBCCDDEEFF00;
  // long (sl) is only 4 bytes in a 32 env
  printf("value of l = 0x%llx\n", l);
  printf("value of sl = 0x%lx\n", sl);
}
```

3.4 Derived/Dependent type

1. arrays
2. pointer
3. function pointer
4. dependent on other types
5. array of char
6. pointer to char
7. pointer to pointer

4 Memory allocation

What is the memory size of char, int, long type?
char - 1, int - 4, long - 8

What is the size of memory allocated by the compiler for char, int and long pointer in a 32-platform?
??? no idea

What is the size of memory allocated for char, int, long pointer for 64 platform
????

What is the size of memory allocated by the compiler for void datatype and void pointer?
????

What is the output of the following in 64 plat?

sizeof cenas

```
int main(){
   char c;
   printf("Sizeof c = %d, %d \n", sizeof(c), sizeof(char));
}
```

R.: 1, 1

What is the output of the following in 64 plat?

sizeof cenas

```
int main(){
   char *cp;
   printf("sizeof cp= %d, %d\n", sizeof(*cp), sizeof(char*));
}
```

R.: 1, 8

What is the output of the following in 64 plat?

sizeof cenas

```
int main(){
   void v, *vp;
   printf("sizeof cp= %d, %d\n", sizeof(v), sizeof(vp));
}
```

R.:compilation error

4.1 Why?

We need to understand how many bytes a union or a strcture is because it will be helpfull when working with the user space debugging. Whenever we get a core dumbped we need to understand what is causing it.

We need to unserdatna a kernel space debuggin

Crash analysis or a memory dumps. We need to look at the values and understand what they mean.

4.2 Memmory alocation - Primirive Data Types

1. char - 1 bytes (8 bits).
2. short (short integer) - 2 bytes (16 bits).
3. int - 4 bytes (32 bits).
4. long (store integer) - 4/8 bytes. Depends on the platform.
5. float - 4 bytes (32 bits) 6 digits of precision
6. void - it means nothing, we do not alocate memory.

4.3 Unsigned vd Signed

The Bytes per data type do not change, but the value it can store differs. For example:
Char is 8 bytes storing values from -128 to 127.
Unsigned char has 8 bytes as well, but stores values from 0 to 255.
This happens to all the data types as far as i know.

Long long Type - created for 32 env, 4 bytes, 4 bytes [8 bytes 32 bit env]. use outside of 32 bit env. In a 64 env, it's viewed as a long.

Pointer is 4 bytes, long is 4 bytes long as well.

5 Data Types - Bits and Bytes

testing data types

```
#include <stdio.h>
int main(void) {
  unsigned char c;
  printf("c = 255, displayed value c = %d\n", c = 255);
}
```

If we had just a char, output would be -1, because char only stores values ranging from -128 to 127 (0xFF).
But because its a unsigned char and doesnt need to store a signal, the we can store up to 255 and prints the correct value.

code

```
  int i;
  printf("i = -2, displayed value i = %x\n", i =- 2);
```

For some reason, its not printed 2, but fffffffe. If i had to guess i would say its because ffff fffe is how you display negative numbers in hex, but i dont know why its beeing displayed in hex.
It turns out that '%x' is how you display hexadecimal numbers.

ScratchPad
8 bit value of 7
0000 0000 - 8 bits
0000 0111 - 7 in binary unsigned or signed

---

but what if we whant -7 in binary, now we have to use signed values
to do that we change the binary value we have to 2's complement:

0000 0111 - 7 in binary **invert bits**
1111 1000
add 1 in the end
1111 1001 - -7 in binary signed