CS 240 Programming in C

Variables and Data types

Jan 31, 2023

So, a char variable is essentially a one-byte integer, why and how it is used to represent a letter?

Char

- Char type in C takes up only 1 byte of memory
- There are signed char and unsigned char, whose values range from -128 to 127 or 0 to 255. For example,

```
char c1 = 65;
unsigned c2 = 65;
```

ASCII Table

- Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@'.
- To utilize the letter meaning of a char variable, we have to treat it as associated with the ASCII table.
- In the standard library ‘stdio.h‘, there are functions like putchar and printf which will print out the letter of a char onto the screen.

Decimal-Binary Conversion

I want to introduce the decimal-binary number conversion. Because we are familiar with decimal numbers, whereas computer CPUs are dealing with binary numbers.

```
For a number in base 10 for example  15;


15   =   2 * 7 + 1
7    =   2 * 3 + 1
3    =   2 * 1 + 1
1    =   2 * 0 + 1
```

Thus $15_{10} = 1111_2$. Specifically,

$$15 = 2^3 * 1 + 2^2 * 1 + 2^1 * 1 + 2^0 * 1$$

Integer

- Things are simple when integers are all positive.
- Things become complex when there are negative integers.
- Because the CPU calculates the equivalent addition for a subtraction which results in the computer storing a negative as its 2's complement form.
- Let's take an example

2's Complement

1, Assume we have a data type of int_4 which contains 4 bits and the first bit is a signed bit, for which 0 stands for positive and 1 for negative.
For example:

```
int_4 a =  3; //  0011
int_4 b = -2; //  1010 (without 2's complement form)
```

2's Complement

2, The process of 2's complement for negative integers,

```
-2    1010    (without 2's complement form)
      1101    1's complement form
      1110    2's complement form
```

3, 2's complement of a positive stays the same.

2's Complement

- How to calculate 1's complement for a binary number?
  - The 1's complement can be easily calculated by inverting the 0s & 1s of a given binary number.

- How to calculate 2's complement for a binary number?
  - Find the one's complement by inverting 0s 1s of a given binary number.
  - Add 1 to the one's complement provides the two's complement.

What will be printed out?

```
int main(void)
{
  unsigned int a = 1000;
  signed int b = -1;

  if (a > b)
      printf("a is more than b");
  else
      printf("a is less or equal than b");

  return 0;
}
```

Overflow of Integers'

- Arithmetic data types in C have fixed memory storage, which means if the value to be assigned is larger than a variable can hold, it will just discard those highest bits.
- Like in the above demo which 11101 is cut into 1101, we will see some demos in real C.
- The implementation overflow of an unsigned integer is a defined behavior, however, the overflow of a signed integer is not, or implementation-defined behavior.

Overflow of Integers'

```
#include <stdio.h>
#include <limits.h>

int main()
{
    /* INT_MAX is the maximum representable integer. */
    int a = INT_MAX;
    printf("a = %d\n", a);
    printf("Adding 1 to a...\n");
    a = a + 1;
    printf("a = %d\n", a);
    return 0;
}
```

Underflow of Integers'

```
#include <stdio.h>
#include <limits.h>

int main()
{
    /* INT_MIN in the least representable integer. */
    int a = INT_MIN;
    printf("a = %d\n", a);
    printf("Subtracting 1 from a...\n");
    a = a - 1;
    printf("a = %d\n", a);
    return 0;
}
```

Float precision

```
#include <stdio.h>
int main()
{
    float f = 0.1;
    double d = 0.1;
    printf("float: %f, double: %f\n", f, d);

    if (f == 0.1)   printf("f == 0.1\n");
    else            printf("f != 0.1\n");

    if (d == 0.1)   printf("d == 0.1\n");
    else            printf("d != 0.1\n");

    return 0;
}
```