Tutorial Link https://codequotient.com/tutorials/C - Data types/5a1d94b752795c1b16c0abea

TUTORIAL

C - Data types

Chapter

1. C - Data types

Topics

1.4 Type Qualifiers

C defines four basic primitive data types: character, integer, floating-point, valueless. These are declared using char, int, float, and void respectively. These data types will determine how much space will be allocated to a particular variable in memory of computer while executing a program. Also, they describe how to interpret the bits of a variable, while using them. As everything in memory is binary numbers. But depending on data type specifications, these binary numbers will be interpreted separately.

char: These variables take 1 byte of memory.

int: These variables depend on the word size of the computer. So, generally, they take 16-bits on old 16-bit computers, whereas on modern 32-bit and 64-bit computers they consume 32-bits of memory. you cannot make assumptions about the size of an integer if you want your programs to be portable to the widest range of environments.

float: They are also depend on implementation. Floating point number can be single-precision or double-precision. Both these will be represented by float and double keywords respectively.

void: It is generally used to declare a function returning no value.

We also have four modifiers to these data types: signed, unsigned, long, short. Signed and unsigned are related to negative numbers handling. Long and short are related to use of short length or long length of bits for a variable. The int base type can be modified by signed, short, long, and unsigned. The char type can be modified by unsigned and signed. You may also apply long to double.

If a variable takes 8-bits in memory, it can take all bits from 0000 0000 to all bits 1111 1111, making total of 256 combinations. Which results in range of this variable. Now either this range can hold half negative and half positive numbers or it can hold all positive numbers. So an 8-bit signed variable may hold -128 to 127 values. Which is -2^7 to 2^7 -1 values. So in general if a variable takes n-bits in memory, then its signed representsation will take -2^(n-1) to +2^(n-1)-1 values, and its unsigned representation may take 0 to 2^(n)-1 values.

So all these can be combined and we can declare some variables as below: -

```c
#include <stdio.h>


int main()
{
  char c;            // c is a character variable of 8 bits. May hold value from -128 to 127.
  signed char c1;         // c is a character variable of 8 bits. May hold value from -128 to 127.
  unsigned char c2;    // c is a character variable of 8 bits. May hold value from 0 to 255.
  int i;            // i is a integer variable of 16/32 bits.
  short int j;        // i will be short integer of 16-bits.
```

```
  long int li;        // li is a long int variable, taking 32-bits.
  float f;            // f is single-precision floating number taking 32-bits
  double    df;        // df is double-precision floating number taking 64-bits
  long double ldf;    // ldf is double-precision floating number taking 80-bits (Largest in size).

  printf("i = %d\n", i);

  return 0;
}

```

In C, there is a rule called 'implicit int'. Which says, in most cases if something is missing in statement, the C compiler presumes an int there. Although it is not a good practice to believe on it. We need to explicitly specify everything. Following are some equivalent declarations: -

```
signed     = signed int
unsigned   = unsigned int
long       = long int
short      = short int
```

Type Qualifiers

In C, we have two type qualifiers, which can allows us to control the variables: const and volatile.

Variables of type const may not be changed by program. They may be assigned an initial value only during declaration. For example,

```
const int i=10;      // I is an constant integer.
i = 20;         // It will be a error.
```

volatile tells the compiler that the value of a variable may change

outside of the program also, due to some external activity.