

Programming Fundamentals- I

Rao Muhammad Umer

Lecturer,

Web: raoumer.github.io

Department of Computer Science & IT,
The University of Lahore.

Administrative Stuff

- Course related stuff will be available soon (**coming Monday**) on “**PIAZZA**” discussion forum
- Due Date for submission of bonus mark coding problem (i.e. calculate money per second / minute a student pays to university) is **Tuesday 25, 2016 at 2:00 pm.**
- Please strictly follow **Plagiarism Rules**

Data Types

Data Types in C

- We must **declare** the type of **every variable** we use in C.
- Every variable has a **type** (e.g. `int`) and a **name** (e.g. `no_students`), i.e. `int no_students`
- Basic data types in C
 - **char**: a single byte, capable of holding one character
 - **int**: an integer of fixed length, typically reflecting the natural size of integers on the host machine (i.e., 32 or 64 bits)
 - **float**: single-precision floating point
 - **double**: double precision floating point

Data Types in C

- Floating-point variables represent numbers with a **decimal** place—like 3.1415927, 0.0000625, and −10.2.
- They have both an integer part, to the left of the decimal point, and a fractional part, to the right.
- Floating-point variables represent what mathematicians call **real numbers**.

Data Types in C

Unsigned and signed

- **Unsigned** means that an int or char value can only be positive. **Signed** means that it can be positive or negative.
- **long** means that **int**, **float** or **double** have more precision (and are larger) than **short** means they have less.

Data Types in C

A typical 32-bit machine

Type	Keyword	Bytes	Range
character	char	1	-128...127
integer	int	4	-2,147,483,648...2,147,438,647
short integer	short	2	-32768...32367
long integer	long	4	-2,147,483,648...2,147,438,647
long long integer	long long	8	-9223372036854775808 ... 9223372036854775807
unsigned character	unsigned char	1	0...255
unsigned integer	unsigned int	2	0...4,294,967,295
unsigned short integer	unsigned short	2	0...65535
unsigned long integer	unsigned long	4	0...4,294,967,295
single-precision	float	4	1.2E-38...3.4E38
double-precision	double	8	2.2E-308...1.8E308

Data Types in C

- All types have a **fixed size** associated with them
- These numbers are highly variable between C compilers and computer architectures.
- Programs that rely on their sizes must be very careful to make their code portable (i.e. Don't rely on the size of the predefined types)

Conversion Specifiers

Specifier	Meaning
<code>%c</code>	Single character
<code>%d</code>	Signed decimal integer
<code>%x</code>	Hexadecimal number
<code>%f</code>	Decimal floating point number
<code>%e</code>	Floating point in “scientific notation”
<code>%s</code>	Character string (more on this later)
<code>%u</code>	Unsigned decimal integer
<code>%%</code>	Just print a % sign
<code>%ld, %lld</code>	long, and long long

- There must be one conversion specifier for each argument being printed out.
- Ensure you use the correct specifier for the type of data you are printing.

Data Types in C

Numeric Variable Types

- **Integer Types:**
 - Generally 32-bits or 64-bits in length
 - Suppose an int has b -bits
 - a signed int is in range $-2^{b-1}..2^{b-1}-1$
-32768 .. 32767
 - an unsigned int is in range $0..2^b-1$
0 .. 65535 (Total 65536)
 - no error message is given on this "overflow"
- **Floating-point Types:**
 - Only use floating point types when really required
 - they do a lot of rounding which must be understood well
 - floating point operations tend to cost more than integer operations

sizeof()

The **sizeof()** function returns the number of bytes in a data type.

```
Int main() {  
printf("Size of char ..... = %d byte(s)\n", sizeof(char));  
printf("Size of short ..... = %d byte(s)\n", sizeof(short));  
printf("Size of int ..... = %d byte(s)\n", sizeof(int));  
printf("Size of long long..... = %d byte(s)\n", sizeof(long long));  
printf("Size of long ..... = %d byte(s)\n", sizeof(long));  
printf("Size of unsigned char. = %d byte(s)\n", sizeof(unsigned char));  
printf("Size of unsigned int.. = %d byte(s)\n", sizeof(unsigned int));  
printf("Size of unsigned short = %d byte(s)\n", sizeof(unsigned short));  
printf("Size of unsigned long. = %d byte(s)\n", sizeof(unsigned long));  
printf("Size of float ..... = %d byte(s)\n", sizeof(float));  
printf("Size of double ..... = %d byte(s)\n", sizeof(double));  
printf("Size of long double .. = %d byte(s)\n", sizeof(long double));  
return 0;  
}
```

More types: const

- `const` means a variable which doesn't vary
 - useful for physical constants or things like `pi` or `e`
 - You can also declare variables as being constants
 - Use the `const` qualifier:

```
const double pi=3.1415926;  
const long double e = 2.718281828;  
const int maxlength=2356;  
const int val=(3*7+6)*5;
```
- (scientific) notation (mantissa/exponent)

```
const double PI = 3.14159e2;
```

Constants

Constants

- Constants are useful for a number of reasons
 - Tells the reader of the code that a value does not change
 - Tells the compiler that a value does not change
 - The compiler can potentially compile faster code
- Use constants whenever appropriate

Reading Numeric Data with scanf

Examples:

```
scanf("%d", &x); /* reads a decimal integer */
```

```
scanf("%f", &rate); /* reads a floating point  
value*/
```

Reading Numeric Data with scanf

- Reading more than one variable at a time:

- For example:

```
int n1, n2;  
float f;  
scanf("%d%d%f",&n1,&n2,&f);
```

- Use white spaces to separate numbers when input.

```
5      10      20.3
```

- In the format string:

- You can use other characters to separate the numbers

```
scanf("value=%d, ratio=%f", &value,&ratio);
```

- You must provide input like:

```
value =27, ratio=0.8
```

Type Conversion

- C allows for conversions between the basic types, implicitly or explicitly. It is also called **casting**.
- Explicit conversion uses the cast operator.
- Example 1:

```
int x=10;  
float y, z=3.14;  
y = (float) x; /* y=10.0 */  
x = (int) z; /* x=3 */  
x = (int) (-z); /* x=-3 --rounded approaching zero */
```

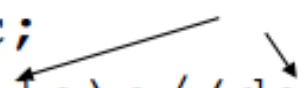
- Example 2:

```
int i;  
short int j=1000;  
i = j*j; /* wrong!!! */  
i = (int) j * (int) j; /* correct */
```

Casting between variables

- `int x = 14, y = 3;`
- Trouble when we divide ints
- A cast is a way of telling one variable type to temporarily look like another.

```
int a= 3;  
int b= 4;      Cast ints a and b to be doubles  
double c;  
c= (double)a/ (double)b;
```



By using *(type)* in front of a variable we tell the variable to act like another type of variable. We can cast between any type usually. However, the only reason to cast is to stop ints being rounded by division.

Variable Declaration

- Generic Form

typename varname1, varname2, ...;

- Examples:

int count;

float a;

double percent, total;

long int y;

unsigned long a_1, a_2, a_3;

Variable Declaration

Initialization

- ALWAYS initialize a variable before using it
 - Failure to do so in C is asking for trouble
 - The value of an uninitialized variables is undefined in the C standards
- Examples:

```
int count; /* Set aside storage space for count */  
count = 0; /* Store 0 in count */
```
- This can be done at definition:

```
int count = 0;  
double percent = 10.0, rate = 0.56;
```

Example

```
#include <stdio.h>

int main ()
{
    double radius, area;
    printf("Enter radius ");
    scanf( "%f", &radius);
    area = 3.14159 * radius * radius;
    printf("\nArea= %d\n\n", area);
    return 0;
}
```

Example

```
#include <stdio.h>

int main ()
{
    const float PI = 3.14;
    double radius, area;
    printf("Enter radius ");
    scanf( "%f", &radius);
    area = PI * radius * radius;
    printf("\nArea= %d\n\n", area);
    return 0;
}
```

Example

```
#include <stdio.h>
#define PI 3.14
int main ()
{
    double radius, area;
    printf("Enter radius ");
    scanf( "%f", &radius);
    area = PI * radius * radius;
    printf("\nArea= %d\n\n", area);
    return 0;
}
```