Quick Review of C Programming

- Data Types
- ANSI C
  - Console Output: printf
  - Console Input: scanf, gets, getchar
  - File I/O: fopen, fclose, fprintf, fscanf, fgets, fgets, feof
  - String Manipulation: sprintf, sscanf
- Arithmetic Functions
- Bitwise Operations

Data Types

Data Types

Important for ACM questions as there are usually restrictions on memory usage and input/output number range

Type	Storage	Maximum	Minimum
char	1 byte	127	-128
short	2 byte	32,767	-32,768
long	4 byte	2,147,483,647	-2,147,483,648
int	4 byte	2,147,483,647	-2,147,483,648
float	4 byte	3.4E +/- 38 (**7 digits**)	3.4E +/- 38 (**7 digits**)
double	8 byte	1.7E +/- 308 (**15 digits**)	1.7E +/- 308 (**15 digits**)
bool (C++)	1 byte	true (if non-zero)	false (if zero)

Data Types

Tips for 'int' type:

- 'int' is actually a type of *undefined* size.
- For old compilers (e.g. Turbo C), it is equivalent to 'short' (16 bits)
- For newer compilers (Visual C), it is the same as 'long' (32 bits)

Tips for 'float/double' type:

- 'float' uses 32-bit. It has at least 7-digit precision
- 'double' uses 64-bit, the range is larger and has 15-digit precision. (so we usually use 'double')

Data Types – Unsigned Version

- Each integer data type (i.e. *char*, *short*, *long*, *int*) has an 'unsigned' version, whose positive range is doubled. The smallest value is 0.

  e.g.

  char = [-128 ...127]
  unsigned char = [0 … 255]

- A floating point data type (i.e. *float*, *double*) or *bool* does not have `unsigned' version.

Tips for 'char' type:

- ' char' is an 8-bit type, for storing ASCII character. Its array forms a string (e.g. char[10]).
- 'char' & 'unsigned char' can actually be used for storing small numbers. {-128..127} and {0..255}, respectively

Data Types

- If the number is too large…
  - Try ‘unsigned’ version
  - Try larger type such as ‘long’ or ‘double’
  - Some compiler has ‘long long’ and ‘long double’ type, which uses 64bits/80bits. But those types are not standard and may not be supported. (e.g. Visual C++ does not support it)
  - For ACM questions, a few even requires you to write your own type. (500 digits…)

ANSI C

Why ANSI C?

- Easy for formatting & parsing
- Runtime efficiency

We'll discuss

- Console I/O
- File I/O
- Output to Strings

Console output: printf()

- For all C I/O function, we need:

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

- To print a plain text, just place it in quotes:

```
printf("This is plain text\n");
```

- The string in quotes is actually a template. It contains your plain text and some place-holders for variables. E.g:

```
printf("This is a number: %d",1);
```

Console output: printf()

- E.g.

printf("One: %d Two: %d Three: %d", 1,2,3);

3 place-holders

3 parameters

Output:

One: 1 Two: 2 Three: 3

The number & types of parameters after the format string must match the number & types of place-holders in the string. Parameters can be immediate data (e.g. 1) or variables (e.g. (int) int_var);

Place-holders:

	Meaning	Param type	Example	Output
%d	Decimal	int, short	int a=1; printf("A = %d",a);	A = 1
%u	Unsigned Decimal	unsigned int	unsigned int b=3; printf("B = %u",b);	B = 3
%ld	Long Decimal	long	long c = 123; printf("C = %ld",c);	C = 123
%f	Floating point	float	float d = 3.141592; printf("D = %f",d);	D = 3.141592
%lf	Long Floating point	double	double e = 3.141592; printf("E = %lf",e);	E = 3.141592
%c	Character	char	char f = 'X'; printf("F = %c",f);	F = X
%s	String	char[]	char g[] = "Test"; printf("G = %s",g);	G = Test
%x %X	Hexadecimal	char, short int, long	int h = 255; printf("H = %x, %X,h);	H = ff, FF
%%	The '%' sign	N/A	printf("100%%");	100%

printf Formatting - Integer

- Commonly used ones: %d, %f and their long versions: %ld, %lf
- It is easy to specify spacing, justification & places

Format	Output	Description
"*%5d*"	* 7*	Totally 5 places for number, right justify
"*%-5d*"	*7 *	Totally 5 places for number, left justify
"*%05d*"	*00007*	Totally 5 places, packed with leading zeroes

- Strings are enclosed in '*' for easy viewing

- What if

  printf("%5d\n",123456);

printf Formatting – Floating Point

Format	Output	Description
"*%.2lf*"	*12.30*	2 places after decimal point.
"*%+.2lf*"	*+12.30*	2 places after decimal point, show sign.
"*%6.2lf*"	* 12.30*	6 places (including decimal point), 2 decimal places, right justify
"*%06.2lf*"	*012.30*	Totally 6 places, 2 decimal places, packed with leading zeroes

Console input: scanf()

- scanf() is similar to printf(), except that it's for input from stdin.

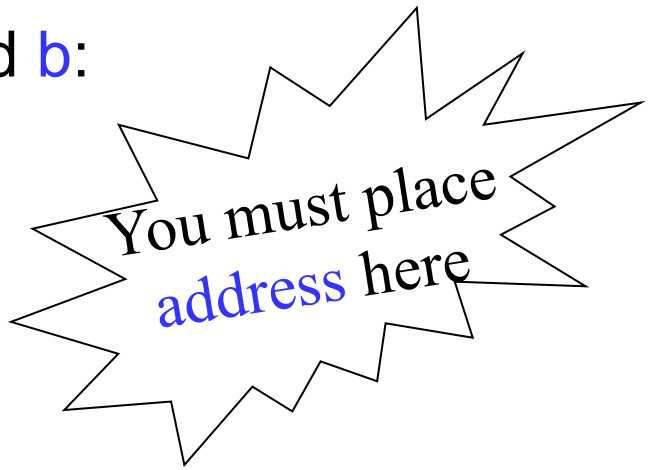
- To print integer variables a and b:

```
printf("%d %d", a, b);
```

- To read integer variable a and b:

```
scanf("%d %d", &a, &b);
```

~~`scanf("%d %d", a, b);`~~



Console input: scanf()

- You can use the format strings as those for printf(), except that you don't need to take care of any formatting issue.
  - there's NO such thing as: scanf("%5d",&a);
  - the correct one should be: scanf("%d",&a);
- scanf() can also perform some simple parsing:
  - E.g. Read x,y co-ordinates: scanf("(%d,%d)",&x,&y);
  - Input: "(123,456)", then x=123 and y=456
  - Even if the input is " (123 , 456) "
  - You still get the correct value of x=123, y=456
- scanf() returns the number of items it gets from the input or EOF if there is an error.

Console input: Strings and Characters

- Input of string is a bit tricky.
- If input string is "Hello World" and you use scanf("%s",Buffer); Then you get "Hello" only.

- To get the whole string, you should use: gets(Buffer); The <enter> at the end of line will be replaced by NULL character.

- To get a single character, use: c = getchar();
- If reading of character is not successful, it will return the constant EOF

File I/O

- To open a file to read, we use the following code:

```
FILE *fp;
fp=fopen("Input.txt","r");
```

- FILE is a structure storing various states about a file… actually we don't care what it is.
- A FILE pointer will be returned by fopen(), we need to store it and use in further file operations
- The second parameter of fopen() is used to specify the open mode of file: ("r": read "w": write)
- To close the file we use: fclose(fp); where fp is the pointer returned by fopen()

File Read/Write Operations

- After the file is opened, we can perform the read/write operation as usual (as stdin).
- There are “file” version of printf() and scanf() with a ‘f’ prefix: fprintf() and fscanf().
- To print (int) Num to file: fprintf(fp,“%d”,Num);
- To read an int from file: fscanf(fp,”%d”,&Num);
- To read a character from file: Ch = fgetc(fp);
- To read a string from file, we use fgets();
- Note that syntax and behavior of fgets() is a bit different from gets()

File Read/Write Operations

- Syntax: fgets(Buffer, length, fp);
  - need to specify the length of Buffer. Reading stops when length-1 characters are read or the newline character is reached. The string is then terminated with a null byte.
  - Unlike gets(), fgets() **preserves** the newline character in string.
- To detect whether end-of-file is reached, we use feof(). If we've not yet reached end-of-file, zero is returned.

Output to String: sprintf() & sscanf()

- Apart from the 'file' version of fprintf() and fscanf(), printf() and scanf() have 2 more cousins!
- They are the 'string' version: sprintf() and sscanf()
- To print int N to string: sprintf(Buffer,"%d", N);
- To read an int from string: sscanf(Buffer,"%d", &N);

Examples:

- String copy (as strcpy()): sprintf(Dest,"%s", Src);
- String concatenation (as strcat()): sprintf(Dest,"%s%s",Src1,Src2);
- Number parsing (as atoi(), atol()):
  - sscanf(Buf,"%d",&Int_Var);
  - sscanf(Buf,"%f",&Float_Var);

Arithmetic Functions

Arithmetic Functions

- Apart from +, -, *, /, C can perform many other arithmetic operations.
- Modulus operator: %
  - e.g. a = 11 % 3; Then a = 2;
  - Only applied to integer types (char, short, int, long)
  - What would be the result of a = -11 % 3 ?
- Other mathematical functions (mostly for floating point) are in the math library:

```
#include <math.h>
```

Arithmetic Functions

- Other math functions:

	Explanation	Example	Result
log(double)	Natural log (base e)	x = log(100)	x = 4.61
log10(double)	Log, using base 10	x = log10(100)	x = 2.00
modf(double, double*)	Return integer & fractional part of a number	y = modf(12.34, &x)	x = 12.0 y = 0.34
pow(double, double)	x raised to the power of y	x = pow(2,3)	x = 8.00
sqrt(double)	square root of a number	x = sqrt(16)	x = 4.00
fabs(double)	absolute for floating-point	x = fabs(-12)	x = 12.0

- Of course C has functions for sin(), cos() and tan()
- But note that the angles are in radians

Floor and Ceilings

- Two functions that worth mentioning are floor() and ceil(). (The numeric floor and ceiling)
- They are different from casting:
  - floor() : largest integer no greater than x
  - ceil() : smallest integer no less than x
  - casting is round-towards-zero

x = 12.34	double y = floor(x);	y = 12
	double y = ceil(x);	y = 13
	int y = (int)(x);	y = 12
x = -12.34	double y = floor(x);	y = -13
	double y = ceil(x);	y = -12
	int y = (int)(x);	y = -12

Bitwise Operations

Bitwise Operations

- One powerful feature of C is that it supports bitwise operation for integer types:
  - Bitwise AND (&) (Note: && is logical)
  - Bitwise OR (|) (Note: || is logical)
  - Bitwise NOT (~) (Note: ! is logical)
  - Bitwise XOR (^)
  - Left arithmetic shift (<<)
  - Right arithmetic shift (>>)

Bitwise Operations

- Bitwise operation is efficient and has lots of usage:
- Checking Odd or Even:
  - if (x&1) printf("x is odd");
- Check for divisibility of 2's power:
  - if (x & 7) printf("x not divisible by 8");
- Round down to nearest multiple of 2's power
  - x = 27; y = x & (~7); // y becomes 24 (8's multiple)
- Quick multiply by 2's power
  - x = 5; x=(x<<2); //(or simply x<<=2;) x=20 (5*4)
- Quick division by 2's power, round down
  - x=27; x>>=2; // x = 6 int(27/4)