ROBOTICS CLUB

SCIENCE AND TECHNOLOGY
COUNCIL
IIT KANPUR

Winter Workshop

General Programming

Courtesy: Professor Swaprawa Nath (ESC101 Instructor)

Robotics Club IITKanpur

What is Programming?

Computer programming is the process of designing and building an executable computer program for accomplishing a specific computing task.

Robotics Club IITKanpur

The Programming Cycle

1. Write your program or edit (i.e., change or modify) your program
2. Compile your program. If compilation fails, return to editing step
3. Run your program on an input. If output is not correct, return to editing step
    a. Repeat step 3 for other inputs, if any

Robotics Club IITKanpur

Simple Program

On the first day, we practice the simplest C

```
# include <stdio.h>
int main ()  {
   printf(“Welcome to Robotics Workshop”);
   return 0;
}
```

The program prints the message “Welcome to Robotics Workshop”

Robotics Club IITKanpur

Program Components

```
# include <stdio.h>
int main ()
{
      printf(“Welcome to Programming”);
      return 0;
}
```

1. This tells the C compiler to include the standard input output library.

2. as

printf is the function called to output from a C program. To print a string, enclose it in “ ” and it gets printed.

“return” returns the control to the caller (program finishes in this case.)

main() is a function. All C programs start by executing from the first statement of

printf(“Welcome to Programming”); is a statement in C. Statements in C end in semicolon ;

printf

- printf is the “voice” of the C program
  - Used to interact with the users
- printf prints its arguments in a certain format
  - Format provided by user

Understand this program?

Program to add two integers (17 and 23)

```
# include <stdio.h>
int main ()  {
   int a = 17;
   int b = 23;
   int c;
   c = a + b;
   printf(“Result is %d”, c);
   return 0;
}
```

The program prints the message:

Result is 40

Robotics Club IITKanpur

Printing the sum of two numbers

HOW WE MUST SPEAK TO THE C COMPILER

```
#include<stdio.h>
int main(){
int a, b, c;
a = 5, b = 4;
c = a + b;
printf(“%d”,c);
return 0;
}
```

5	4
a	b
	9
	c

HOW WE USUALLY SPEAK TO A HUMAN

I’m speaking English

Hello

a,b,c are variables.

a = 5 and b = 4.

Please add them and put the result in variable c.

Please tell me value of c.

Goodbye

Robotics Club IITKanpur

Words

- Made of alphabets
- Used to convey meaning
- English words have fixed meanings
- C *keywords* have fixed meanings
- All other C words (identifiers) have variable meanings
  - They take the meaning you want to give them

C Keywords

auto	double	int ●	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return ●	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

● Seen already

These 32 keywords mean the same across every C compiler

Some compilers reserve a few extra keywords, but those are less important.

int

- Computers store data in binary code
  - A 0 or 1 is a bit
  - 8 bits make a byte
  - 2/4/8 bytes make a word (depending on architecture)
- The keyword *int* asks the computer to assign one *word* of memory to store an integer value
  - int a = 34;
  - `0000 0000 | 0010 0010`
- How many integers can you store using N bits?
- Can only use *int* to store integers in a limited range
  - If you exceed the range, you will get a compilation error

Robotics Club IITKanpur

C Identifier/Variable Syntax

- Can use
  - A – Z
  - a – z
  - 0 – 9
  - The underscore character
- Cannot begin with a number
- A_3, abcDS2, this_variable are fine
- 321, 5_r, dfd@dhr, this variable, no-entry are not

Robotics Club IITKanpur

Keyword Usage

```
#include <stdio.h>
int main(){
    int else = 3;
    printf("%d", e
    return 0;
}
```

This won’t work

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

C character constants

```
#include <stdio.h>
int main(){
    int a = 'B';
    printf("%d\n", a);
    return 0;
}
```

What do you think the output will be?

Robotics Club IITKanpur

ASCII character set

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Translates letters to numbers for the computer to understand.

Character Constant Operations

```
#include <stdio.h>
int main(){
  int a = 'C' - '3';
  printf("%d\n", a);
  return 0;
}
```

```
#include <stdio.h>
int main(){
  int a = 'c' - '3';
  printf("%d\n", a);
  return 0;
}
```

Another Example: Playing with ASCII

A program that converts Capital to small characters

```
# include <stdio.h>
int main(){
    char first = ‘D’;
    char second =_______;
    printf(“__ is now __\n”, first, second);
    return 0;
}
```


Robotics Club IITKanpur

Playing with ASCII

A program that converts Capital to small

```c
# include <stdio.h>
int main(){
    char first = ‘D’;
    char second = first + ‘a’ - ‘A’;
    printf(“%c is now %c\n”, first, second);
    return 0;
}
```

Tracing the Execution

```
1   # include <stdio.h>
2   int main()
3   {
4           printf("Welcome to ");
5           printf("C Programming");
6           return 0;
    }
```

After lines 3,4

After lines 5,6

Output:

Welcome to C Programming

- Line numbers of C program are given for clarity
- Program counter (reader part of C Compiler) starts at the first executable statement of main
- Program terminates gracefully when main "returns"

Variables

- A name associated with memory cells (boxes) that store data
- Type of variable determin… of th… .

    int m = 64;

    char c = ‘X’;

    float f = 3.1416;

- Variables can change their value during program

    f = 2.7183;

m 64 c 88

f 2.7183

Robotics Club IITKanpur

Variable Declaration

- To communicate to compiler the names and types of the variables used by the program
  - Type tells size of the box to store value
  - Variable must be declared before used
  - Optionally, declaration can be combined with definition (initialization)

Declaration without initialization

int count;

Declaration with initialization

int min = 5;

Robotics Kanpur

Data Types in C

- int
  - Bounded integers, e.g. 732 or -5
- float
  - Real numbers, e.g. 3.14 or 2.0
- double
  - Real numbers with more precision
- char
  - Single character, e.g. a or C or 6 or $

Some modern compilers use 4 bytes for int

1 byte | 1 byte

1 byte | 1 byte | 1 byte | 1 byte

1 byte | 1 byte | 1 byte | 1 byte

1 byte | 1 byte | 1 byte | 1 byte

1 byte

Assignment Statement

- A simple assignment statement

  variable = expression / value to be assigned;

- Computes the value of the expression on the right hand side (RHS), and stores it in the "box" of the variable on the left hand side (LHS)
- = is known as the assignment operator
- Examples

```
x = 10;
ch = 'c';
disc_2 = b*b – 4*a*c;
count = count + 1;
```

Robotics Club IITKanpur

Input/Output

- Input: receive data from external sources (keyboard, mouse, sensors)
- Output: produce data (results of computations) (to monitor, printer, projector, ...)

Robotics Club IITKanpur

Input/Output

- `printf` function is used to display results to the user. (output) - **voice** of C compiler
- `scanf` function is used to read data from the user. (input) - **ear** of C compiler
- Both of these are provided as library functions.
  - `#include <stdio.h>` tells compiler that these (and some other) functions may be used by the programmer.

Robotics Club IITKanpur

Output - printf

string to be displayed, with placeholders

\n is the newline character.

```
printf("%d kms is equal\nto %f miles.\n", km, mi);
```

The string contains placeholders (%d and %f). Exactly one for each expression in the list of expressions.

Placeholder and the corresponding variable have compatible type.

While displaying the string, the placeholders are replaced with the value of the corresponding expression: first placeholder by value of first expression, second placeholder by value of second expression, and so on.

Input - scanf

Similar to printf: string with placeholders, followed by list of variables to read

& is the *addressof* operator. To be covered later.

```
scanf("%d", &km);
```

Note the & before the variable name. DO NOT FORGET IT.

- String in " " contains only the placeholders corresponding to the list of variables after it.
- Best to use one scanf statement at a time to input value into one variable.

Robotics Club IITKanpur

Some Placeholders

Placeholder	Type
%d	**int**
%f	**float**
%lf	**double**
%c	**char**
%%	**literal percent sign (%)**

If placeholder and expression/variable type do not match, you may get unexpected results.

Robotics Club IITKanpur

Special Characters

Escape Sequence	**Character**	ASCII Value
\0	null	000
\t	horizontal tab	009
\n	new line (line feed)	010
\v	vertical Tab	011
\”	quotation mark	034
\ \	backslash	092

Robotics Club IITKanpur

Composite data types

- signed short int = signed short = short (%hi)
- signed long int = signed long = long (%li)
- unsigned int (%u)
- float (%f)
- double (%lf)
- long double (%Lf)

Comments in C

- Anything written between /* and */ is considered a comment.

diameter = 2*radius; /* diameter of a circle */

- Comments cannot be nested.

/* I am /* a comment */ but I am not */

First */ ends the effect of all unmatched start-of-comments (/*).

Comments in C

- Anything written after // up to the end of that line

```
diameter = 2*radius; // diameter of a circle
area = pi*radius*radius; // and its area
```

- Not all C compilers support this style of comments.
    - Our online compiler **does** support it.

Relational Operators

- Compare two quantities
- Work on int, char, float, double

Operator	Function
>	Strictly greater than
>=	Greater than or equal to
<	Strictly less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

Examples

Rel. Expr.	**Result**	Remark
3>2	1	
3>3	0	
‘z’ > ‘a’	1	ASCII values used for char
2 == 3	0	
‘A’ <= 65	1	'A' has ASCII value 65
‘A’ == ‘a’	0	Different ASCII values
(‘a’ – 32) == ‘A’	1	
5 != 10	1	
1.0 == 1	AVOID	May give unexpected result due to approximation

Avoid mixing int and float values while comparing.
Comparison with floats is not exact!

Example

- Problem: Input 3 positive integers. Print the count of inputs that are even and odd
  - Do not use if-then-else

INPUT
10
5
3

OUTPUT
Even=1
Odd=2

```
int a; int b; int c;
int cEven; // count of even inputs
scanf("%d%d%d", &a,&b,&c); // input a,b,c

// (x%2 == 0) evaluates to 1 if x is Even,
// 0 if x is Odd

cEven = (a%2 == 0) + (b%2 == 0) + (c%2 == 0);
printf("Even=%d\nOdd=%d", cEven, 3-cEven);
```

Logical Operators

Logical Op	**Function**	Allowed Types
&&	Logical AND	char, int, float, double
||	Logical OR	char, int, float, double
!	Logical NOT	char, int, float, double

Remember

- value 0 represents false.
- any other value represents true. Compiler returns 1 by default

Robotics Club IITKanpur

Examples

Expr	**Result**	Remark
2 && 3	1	
2 || 0	1	
‘A’ && ‘0’	1	ASCII value of ‘0’≠0
‘A’ && 0	0	
‘A’ && ‘b’	1	
! 0.0	1	0.0 == 0 is **guaranteed**
! 10.05	0	Any real ≠ 0.0
(2<5) && (6>5)	1	Compound expr

Precedence and Associativity

- NOT has same precedence as equality operator
- AND and OR are lower than relational operators
- OR has lower precedence than AND
- Associativity goes left to right
- 2 == 2 && 3 == 1 || 1 == 1 || 5 == 4 is true
- Recommended: use brackets

Robotics Club IITKanpur

Operator Precedence

HIGH

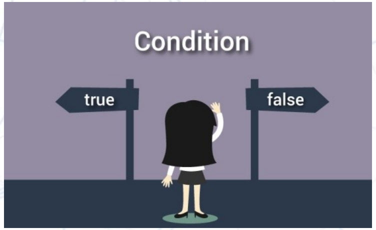
INCREASING

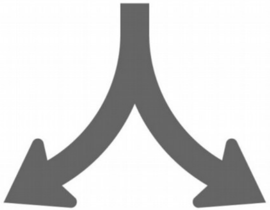
LOW

Operators	**Description**	Associativity
< > >= <=	Relational operators	Left to right
== !=	Equal, not equal	Left to right
&&	And	Left to right
||	Or	Left to right
=	Assignment	Right to left

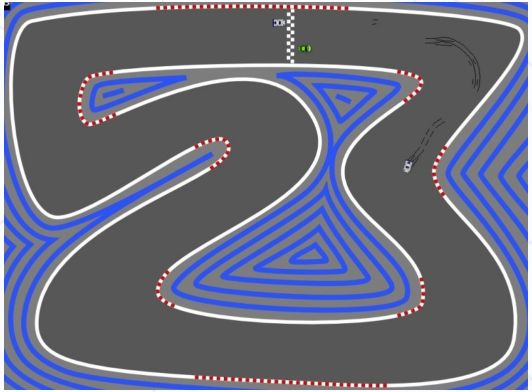
Control Statements

- Branching





- Looping



Branching Statements in C

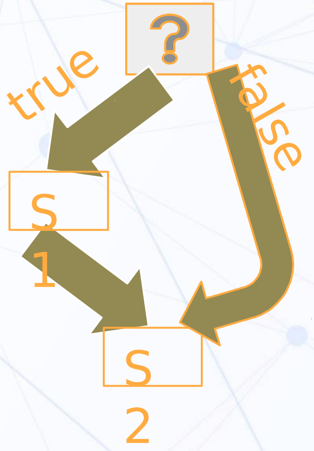
- 3 types of conditional statements in C
  - if (cond) action
  - if (cond) action
    else some-other-action
  - switch-case
- Each action is a sequence of one or more statements!

Robotics Club IITKanpur

if Statement

- General form of the if statement

```
if (expression)
            statement S1
statement S2
```

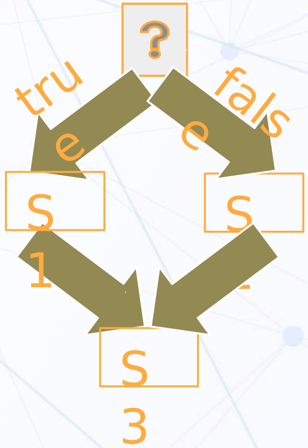


- Execution of if statement
  - First the expression is evaluated.
  - If it evaluates to a non-zero value, then S1 is executed and then control (program counter) moves to the statement S2.
  - If expression evaluates to 0, then S2 is executed.

if-else Statement

- General form of the if-else statement

```
if (expression)
            statement S1
else
            statement S2
statement S3
```



- Execution of if-else statement
  - First the expression is evaluated.
  - If it evaluates to a non-zero value, then S1 is executed and then control (program counter) moves to S3.
  - If expression evaluates to 0, then S2 is executed and then control moves to S3.
  - S1/S2 can be a **block** of statements!

Robotics Club IITKanpur

Example

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

int main() {
  int n;
  double m;
  printf(“Please enter a positive number: ”);
  scanf(“%d”,&n);
  if (n>0){
   m = log(n);          // natural log
   printf("%f\n", m);
  }
  else
   printf(“Why can’t you follow instructions?”);
  return 0;
}
```

Nested if, if-else

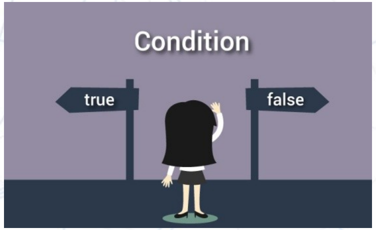
- Earlier examples showed us *nested* if-else statements

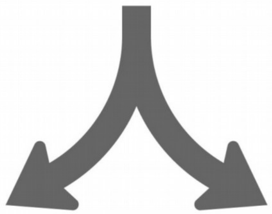
```
if (a <= b) {
        if (a <= c) { … }  else {…}
} else {
         if (b <= c)  { … } else { … }
}
```

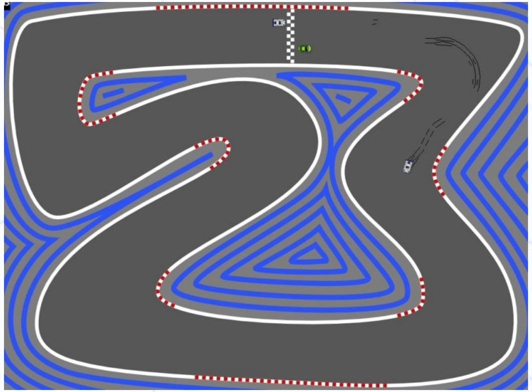
- Because if and if-else are also statements, they can be used anywhere a statement or block can be used.

Control Statements

- Branching ✔
- Looping







The for loop

Brackets essential if you want to do many things while looping

General form of a for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    statement2;
    ...
}
statement3;
statement4;
...
```

How we usually speak to a human

1. Do what is told in initialization expression
2. Then check the stopping expression
3. If stopping expression is true
    Execute all statements inside braces
    Execute update expression
    Go back to step 2
  Else stop looping and execute rest of the code

The for loop

```
for(init_expr; stopping_expr; update_expr){
    statement1;
    Statement2;
}
```

The entire for loop is considered one statement

Can put inside for loops: printf statements, if-else/switch statements, even for loop statement (nested for loop)

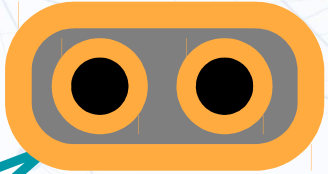
Usually init_expr, stopping_expr, update_expr involve the same variable, e.g. b in multiplication table example

Lovingly called variable of the loop/counter variable

Print sum of reciprocals of 1, 2, …, n

Take n >= 1 from the user and give as output

$$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$$



Oops! Integer division!

The repeating task can be

*Given the sum of first i-1 reciprocals and add **1/i** to it*

Define a variable (let's call it **sum**) to store parti sums

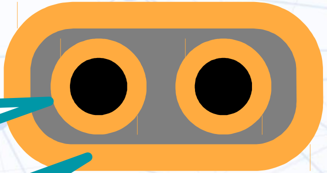
Also called *partial sums* or *running sums*

The above task is accomplished by th

```
sum = sum + 1/i;
sum = sum + 1.0/i;
sum = sum + (double)1/i;
```

Robotics Club IITKanpur

Loop Invariant



Very important once loops get more complicated

Loop invariants are powerful ways to ensure that your loop code is correct!

Notice that in previ

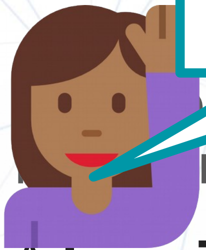
At the beginning of i-th iterati

$$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{i-1}$$

Excep... iteration with i = 1, where sum stored 0

After t... tored the value

$$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{i}$$



In i-th iteration the string 2 x i = 2i will get printed

...e known as *loop invariants* – few nice properties that ...r all iterations of a loop

A **loop invariant** is a formal statement about the relationship between variables in your program which holds true just before the **loop** is ever run (es... e **loop**, e...

Exercise 1: sum of reciprocals of the first n even numbers
Exercise 2: find if a number is prime or not
Exercise 3*: sum of reciprocals of the first n prime

Some common errors in loops

Initialization: forget to do it or else wrong initialization

Statements: Note, update_expr executed **after** statements

Update: Forget to do update step or wrong update step

Termination: wrong or missing termination

for(b=1;**b<10**;b++){…} not same as for(b=1;**b<=10**;b++){…}

Infinite loop: The loop goes on forever. Never terminates.

for(b=2;b>=1,b++){…}

Interesting Exercise

Euler series

$$1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \ldots + \infty$$

We know the solution $\frac{\pi^2}{6}$

How should we compute it numerically?