Course Materials

Slides available at http://cse.iitkgp.ac.in/pds/current

More materials available at http://cse.iitkgp.ac.in/pds

Books:

1. Programming with C, Byron Gottfried
2. The C Programming Language, Brian W Kernighan, Dennis M Ritchie
3. Programming in ANSI C, E. Balaguruswamy
4. Data Structures, S. Lipschutz, Schaum’s Outline Series

... and many more options
Teachers and Class Timings

- Section 1, 2
  - Monday (3-4:55 pm), Tuesday (3-3:55 pm)
  - Teacher: Prof. Debasis Samanta (DSM)
- Section 3, 4
  - Monday (3-4:55 pm), Tuesday (3-3:55 pm)
  - Teacher: Prof. Pallab Dasgupta (PDG)
- Section 5, 6
  - Monday (3-4:55 pm), Tuesday (3-3:55 pm)
  - Teacher: Prof. Partha Pratim Chakraborty (PPC)
- Section 7, 8
  - Wednesday (10-10:55 am), Thursday (9-9:55 am), Friday (11-11:55 am)
  - Teacher: Prof. Somindu Chaya Ramanna (SCR)
- Section 9, 10
  - Wednesday (10-10:55 am), Thursday (9-9:55 am), Friday (11-11:55 am)
  - Teacher: Prof. Sudebkumar Prasant Pal (SPP)
Evaluation Plan (Tentative)

- 2 short tests (about 1 hour each)
  - Around 30-40% of the marks

- 2 long tests (about 2 hours each)
  - Around 70-60% of the marks
Basic Components in a Computer

- Control Unit
- Arithmetic Logic Unit
- Memory Unit
- Central Processing Unit
- Input Unit: Mouse, Keyboard, Microphone, etc.
- Output Unit: Monitor, Printer, Speakers, etc.
- Storage (Permanent)
- Memory (Temporary)
A computer needs to be programmed to do tasks. Programming is the process of writing instructions in a language that can be understood by the computer so that a desired task can be performed by it. A program is a sequence of instructions to do a task, computer processes the instructions sequentially one after the other. Software are programs for doing tasks on computers.
Three steps in writing programs

**Step 1:** Write the program in a high-level language (in your case, C)

**Step 2:** Compile the program using a C compiler

**Step 3:** Run the program (as the computer to execute it)
The decimal number system we use is base-10

- 10 digits, from 0 to 9, Positional weights 10^0, 10^1, 10^2,...from right to left for integers
- Example: 723 = 3x10^0 + 2x10^1 + 7x10^2

Numbers are represented inside computers in the base-2 system (Binary Numbers)

- Only two symbols/digits 0 and 1
- Positional weights of digits: 2^0, 2^1, 2^2,...from right to left for integers
- Example: 1101 in binary = (1 x 2^0) + (0 x 2^1) + (1 x 2^2) + (1 x 2^3) = 13 in decimal
## Binary Numbers

<table>
<thead>
<tr>
<th>Dec</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
</tbody>
</table>
### Binary Numbers

#### Binary to Decimal Conversion

<table>
<thead>
<tr>
<th>Dec</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
</tbody>
</table>

- **101011** ➔ $1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 43$
  
  
  $\text{(101011)}_2 = (43)_{10}$

- **111001** ➔ $1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 57$
  
  $\text{(111001)}_2 = (57)_{10}$

- **10100** ➔ $1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 20$
  
  $\text{(10100)}_2 = (20)_{10}$
Bits and Bytes

• Bit – a single 1 or 0

• Byte – 8 consecutive bits
  • 2 bytes = 16 bits
  • 4 bytes = 32 bits

• Maximum integer that can represented
  • in 1 byte = 255 (=11111111)
  • In 4 bytes = 4294967295 (= 32 1’s)

• Number of integers that can be represented in 1 byte = 256 (the integers 0, 1, 2, 3,…,255)
Fundamentals of C
```c
#include <stdio.h>

int main()
{
    printf("Hello, World! \n");
    return 0;
}
```

**Output**

```
Hello, World!
```
A Simple C program

#include <stdio.h>

int main()
{
    int x, y, sum, max;

    scanf("%d%d", &x, &y);

    sum = x + y;
    if (x > y) max = x;
    else max = y;

    printf("Sum = %d\n", sum);
    printf("Larger = %d\n", max);
    return 0;
}

When you run the program

Output after you type 15 and 20

15 20
Sum = 35
Larger = 20
Structure of a C program

- A collection of **functions** (we will see what they are later)

- Exactly one special function named **main()** must be present. Program always starts from there.
  - Until we study functions in detail, this is the only function your programs will have for now

- Each function has statements for variable declarations, assignment, condition check, looping etc.
- Statements are executed one by one in order
```c
#include <stdio.h>

int main()
{
    int x, y, sum, max;
    scanf("%d%d", &x, &y);
    sum = x + y;
    if (x > y)
        max = x;
    else
        max = y;
    printf("Sum = %d\n", sum);
    printf("Larger = %d\n", max);
    return 0;
}
```
Writing a C program

- You will have to understand what different statements do to decide which you should use in what order to solve your problem.

- There is a fixed format ("syntax") for writing each statement and other things. Need to remember the syntax.
  - Do not question why you have to type exactly like this, you just have to or it is not a C program!!
  - Compiler will give error if your typed program does not match required C syntax.

- There are other rules to follow.
Things you will see in a C program (we will look at all these one by one)

- Variables
- Constants
- Expressions (Arithmetic, Logical, Assignment)
- Statements (Declaration, Assignment, Control (Conditional/Branching, Looping))
- Arrays
- Functions
- Structures
- Pointers
The C Character Set

The C language alphabet

- Uppercase letters ‘A’ to ‘Z’
- Lowercase letters ‘a’ to ‘z’
- Digits ‘0’ to ‘9’
- Certain special characters:

```
! # % ^ & * ( )
- _ + = ~ [ ] \ 
| ; : ‘ “ { } ,,
. < > / ?
```

whitespace characters (space, tab, …)

A C program should not contain anything else
Variables

- Very important concept for programming
- An entity that has a value and is known to the program by a name
- Can store any temporary result while executing a program
- Can have only one value assigned to it at any given time during the execution of the program
- The value of a variable can be changed during the execution of the program

Variables are stored in memory

Memory is a list of consecutive storage locations, each having a unique address

A variable is like a bin
- The contents of the bin is the value of the variable
- The variable name is used to refer to the value of the variable
- A variable is mapped to a location of the memory, called its address
#include <stdio.h>
int main()
{
    int x;
    int y;

    x=1;
    y=3;

    printf("x = %d, y = %d\n", x, y);
    return 0;
}
Variables in Memory

Instruction sequence

X = 10
X = 20
X = X + 1
X = X * 5

Memory

X

(may contain whatever value at the beginning)
Variables in Memory

Instruction sequence

X = 10
X = 20
X = X +1
X = X*5

after executing X = 10

Memory

X

10
Variables in Memory

Instruction sequence

X = 10
X = 20
X = X + 1
X = X*5

Memory

X

20

after executing X = 20
Variables in Memory

Instruction sequence

X = 10
X = 20
X = X + 1
X = X * 5

Memory

X

21

after executing X = X + 1
Variables in Memory

Instruction sequence

X = 10
X = 20
X = X + 1
X = X * 5

after executing X = X * 5

Memory

X

105
Variables in Memory

Instruction sequence

1. \( X = 20 \)
2. \( Y = 15 \)
3. \( X = Y + 3 \)
4. \( Y = X / 6 \)

Memory

- \( X \) (may contain whatever value at the beginning)
- \( Y \) (may contain whatever value at the beginning)
Variables in Memory

Instruction sequence

X = 20
Y = 15
X = Y + 3
Y = X / 6

Memory

X
20

Y
?
Variables in Memory

Instruction sequence

X = 20
Y = 15
X = Y + 3
Y = X / 6

Memory

X

20

Y

15

after executing Y = 15
Variables in Memory

Instruction sequence

X = 20
Y = 15
X = Y + 3
Y = X / 6

Memory

X
18
Y
15

after executing X = Y + 3
Variables in Memory

Instruction sequence

X = 20
Y = 15
X = Y + 3
Y = X / 6

after executing Y = X / 6

Memory

X

Y

18
3
Data Types

- Each variable has a type, indicates what type of values the variable can hold
- Four common data types in C (there are others)
  - `int` - can store integers (usually 4 bytes)
  - `float` - can store single-precision floating point numbers (usually 4 bytes)
  - `double` - can store double-precision floating point numbers (usually 8 bytes)
  - `char` - can store a character (1 byte)
Rules and good practices

• First rule of variable use: **Must declare a variable** (specify its **type** and **name**) before using it anywhere in your program

• All variable declarations should ideally be at the beginning of the `main()` or other functions
  • There are exceptions, we will see later

• A value can also be assigned to a variable at the time the variable is declared.
  
  ```
  int speed = 30;
  char flag = 'y';
  ```
Variable Names

- Sequence of letters and digits
- First character must be a letter or ‘_’
- No special characters other than ‘_’
- No blank in between
- Names are case-sensitive (max and Max are two different names)

Examples of valid names:
- i rank1 MAX max Min class_rank

Examples of invalid names:
- a’s fact rec 2sqroot class,rank
### More Valid and Invalid Identifiers

<table>
<thead>
<tr>
<th>Valid identifiers</th>
<th>Invalid identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>10abc</td>
</tr>
<tr>
<td>abc</td>
<td>my-name</td>
</tr>
<tr>
<td>simple_interest</td>
<td>“hello”</td>
</tr>
<tr>
<td>a123</td>
<td>simple interest</td>
</tr>
<tr>
<td>LIST</td>
<td>(area)</td>
</tr>
<tr>
<td>stud_name</td>
<td>%rate</td>
</tr>
<tr>
<td>Empl_1</td>
<td></td>
</tr>
<tr>
<td>Empl_2</td>
<td></td>
</tr>
<tr>
<td>_avg_empl_salary</td>
<td></td>
</tr>
</tbody>
</table>
C Keywords

- Used by the C language, cannot be used as variable names

- Examples:
  
  \[\text{int, float, char, double, main, if else, for, while, do, struct, union, typedef, enum, void, return, signed, unsigned, case, break, sizeof}, \ldots\]

- There are others, see textbook…
Example 1

```c
#include <stdio.h>
int main()
{
    int x, y, sum;  // Three integer type variables declared

    scanf("%d%d", &x, &y);
    sum = x + y;   // Values assigned

    printf("%d plus %d is %d\n", x, y, sum);
    return 0;
}
```
Example 2

```c
#include <stdio.h>
int main()
{
    float x, y; /* Two floating point variables declared */
    int d1, d2 = 10; /* Here integer variable d2 is initialized to 10, can be changed later */

    scanf("%f%f%d", &x, &y, &d1);
    printf("%f plus %f is %f\n", x, y, x + y);
    printf("%d minus %d is %d\n", d1, d2, d1 - d2);

    return 0;
}
```
Read-only Variables

- Variables whose values can be initialized during declaration, but cannot be changed after that
- Declared by putting the `const` keyword in front of the declaration
- Storage allocated just like any variable
- Used for variables whose values need not be changed
  - Prevents accidental change of the value

```
int main() {
    const int LIMIT = 10;
    int n;
    scanf("%d", &n);
    if (n > LIMIT) printf("Out of limit");
    return 0;
}
```

```
int main() {
    const int Limit = 10;
    int n;
    scanf("%d", &n);
    Limit = Limit + n;
    printf("New limit is %d", Limit);
    return 0;
}
```

Correct

Incorrect: Limit changed
Constants

• Integer constants
  • Consists of a sequence of digits, with possibly a plus or a minus sign before it
  • Embedded spaces, commas and non-digit characters are not permitted between digits

• Floating point constants

• Two different notations:
  • Decimal notation: 25.0, 0.0034, .84, -2.234
  • Exponential (scientific) notation
    • $3.45e23$, $0.123e-12$, $123e2$

  e means “10 to the power of”
• Character constants
  • Contains a single character enclosed within a pair of single quote marks.
  • Examples :: ‘2’, ‘+’, ‘Z’

• Some special backslash characters
  • ‘\n’ new line
  • ‘\t’ horizontal tab
  • ‘\’ single quote
  • ‘\\’ double quote
  • ‘\0’ null
Typical Size of Data Types

- **char** – 1 byte
- **int** – 4 bytes
- **float** – 4 bytes
- **double** – 8 bytes

- “Typical”, because some of them vary depending on machine/OS type
- Therefore never use the values (1, 4, 8) directly, use the `sizeof()` operator given
  - `sizeof(char)` will give 1, `sizeof(int)` will give 4 and so on ... as per your PC/Laptop
Input: `scanf` function

- Performs input from keyboard
- It requires a format string and a list of variables into which the value received from the keyboard will be stored
- Format string = individual groups of characters (usually ‘%’ sign, followed by a conversion character), with one character group for each variable in the list

```c
int a, b;
float c;
scanf("%d%d%f", &a, &b, &c);
```

Commonly used conversion characters:
- `c` for char type variable
- `d` for int type variable
- `f` for float type variable
- `lf` for double type variable
Examples

```c
scanf("%d", &size);
  // Reads one integer from keyboard into an int type variable named size

scanf("%c", &nextchar);
  // Reads one character from keyboard into a char type variable named nextchar

scanf("%f", &length);
  // Reads one floating point (real) number from keyboard into a float type variable named length

scanf("%d%d", &a, &b);
  // Reads two integers from keyboard, the first one in an int type variable named a and the second one in an int type variable named b
```
Important

• scanf( ) will wait for you to type the input from the keyboard

• You must type the same number of inputs as the number of %’s in the format string

• Example: if you have scanf(“%d%d”,...), then you must type two integers (in same line or different lines), otherwise scanf( ) will just wait and the next statement will not be executed
Reading a single character

• A single character can be read using `scanf` with `%c`

• It can also be read using the `getchar()` function
  ```c
  char c;
  c = getchar();
  ```

• Program waits at the `getchar()` line until a character is typed, and then reads it and stores it in `c`

• What happens here?
  ```c
  char a, b;
  printf("Enter the first character:\n");
  a = getchar();
  printf("Enter the second character:\n");
  b = getchar();
  ```

• You may encounter a situation where the computer does not wait for reading the second character. Why?
Output: `printf` function

- Performs output to the standard output device (typically defined to be the screen)

- It requires a format string in which we can specify:
  - The text to be printed out
  - Specifications on how to print the values
    ```c
    printf("The number is %d\n", num);
    ```
  - The format specification `%d` causes the value listed after the format string to be embedded in the output as a decimal number in place of `%d`

- Output will appear as: `The number is 125`
General syntax of `printf`:

`printf (format string, arg1, arg2, ..., argn);`

- **format string** refers to a string containing formatting information and data types of the arguments to be output.
- The arguments `arg1, arg2, ...` represent a list of variables/expressions whose values are to be printed.
- The conversion characters are the same as in `scanf`.

- **Examples:**

  ```
  printf ("Average of %d and %d is %f", a, b, avg);
  printf ("Hello \nGood \nMorning \n");
  printf("%3d %3d %5d", a, b, a*b+2);
  printf("%7.2f  %5.1f", x, y);
  ```

- Many more options are available for both `printf` and `scanf` – read from the book.
More Examples
(Explain the outputs to test if you understood format strings etc.)
```c
#include <stdio.h>

int main()
{
    printf("Hello, World! ");
    printf("Hello \n World! \n");
    return 0;
}
```

**Output**

Hello, World! Hello World!
Some more print

```c
#include <stdio.h>

int main()
{
    printf("Hello, World! \n");
    printf("Hello \n World! \n");
    printf("Hell\no \t World! \n");
    return 0;
}
```

Output

```
Hello, World!
Hello World!
Hello World!
```
```c
#include <stdio.h>

int main()
{
    float f1, f2;
    int x1, x2;

    printf("Enter values for f1 and f2: \n");
    scanf("%f%f", &f1, &f2);
    printf("Enter values for x1 and x2: \n");
    scanf("%d%d", &x1, &x2);

    printf("f1 = %f, f2 = %5.2f \n", f1, f2);
    printf("x1 = %d, x2 = %10d \n", x1, x2);

    return 0;
}
```

Output

```
Enter values for f1 and f2: 23.5 14.326
Enter values for x1 and x2: 54 7
f1 = 23.500000, f2 = 14.33
x1 = 54, x2 = 7
```

Can you explain why 14.326 got printed as 14.33?
Some more print

```c
#include <stdio.h>

int main()
{
    char c1, c2;
    scanf("%c%c", &c1, &c2);
    printf("%c%c", c1, c2);
    return 0;
}
```

Output

ab
ab
What about this?

```c
#include <stdio.h>

int main()
{
    char c1, c2;
    scanf("%c%c", &c1, &c2);
    printf("%c%c", c1, c2);
    return 0;
}
```

Output

```
Can you explain why only ‘a’ was printed this time, even though it is the same program as in the last slide? Note the difference from the last slide carefully. Also note that two characters were read this time also, or scanf would have waited
```
Practice Problems

Write C programs to

1. Read two integers and two floating point numbers, each in a separate scanf() statement (so 4 scanf’s) and print them with separate printf statements (4 printf’s) with some nice message
2. Repeat 1, but now read all of them in a single scanf statement and print them in a single printf statement
3. Repeat 1 and 2 with other data types like double and char
4. Repeat 1 and 2, but now print all real numbers with only 3 digits after the decimal point
5. Read 4 integers in a single scanf statement, and print them (using a single printf statement) in separate lines such that the last digit of each integer is exactly 10 spaces away from the beginning of the line it is printed in (the 9 spaces before will be occupied by blanks or other digits of the integer). Remember that different integers can have different number of digits
6. Repeat 5, but now the first integer of each integer should be exactly 8 spaces away from the beginning of the line it is printed in.