First C-program

```c
/*

  File with name lecture1.c

  My first c-program

*/

#include <stdio.h>

/* function main begins */
int main()

{
   printf("Hello world!!!\n");
   return 0; /* indicate that program ended successfully */
} /* end function main */
```

First, we will compile and run the program and see the difference with changes.

Compile and run the program on linux

```
sanjeev[8]gcc -o lecture1 lecture1.c
sanjeev[9]./lecture1
Hello world!!!
Sanjeev[10]
```

- Here, there are few words shown in green background. Those are known as prompt. Ignore that part, if it shows something.
- First line is showing a command as "`gcc  -o  lecture1   lecture1.c` "
- Here, we can put the command as "`gcc  lecture1.c  -o   lecture1` "
- Here `gcc` is a compiler. Other compilers are: `cc, g++, visual-c, smaller-c` etc. Some of them works on linux and some not.
- In this compiler line, `-o` is combined with the next name written after it. In above example it is `-o lecture1`. Here, we should not confuse with 0 (zero) and O (15th upper-case letter of English alphabet. We will use lower-case 15th letter of modern English language. Word written just after "-o" will represent the output executable file.

Compile continued...

- There is some other words may be there in the same line. One or more arguments will end with "`.c`". These files are the files which are written in c-programming language.
- If this command is successful, we will see the prompt in next line without any message.
- If compilation is successful, that doesn't mean that our work is done. There may be some other errors.
- In next page, we will discuss the errors in this compilation.

Compilation command errors

These errors are written as previous year students did many times. There may be more than these errors.

“`gcc  -o lecture1.c`”

In above case, our program will compile but will not give any output file. Sometimes, it can remove original program and write the executable output file with name “`lecture1.c`”

“`gcc  -o lecture1.c lecture1.c`”

In this case, it will remove the original program and write executable with same name. In this case, we may lose our original program.

Compilation command errors (continued…)

“`gcc-o lecture1 lecture1.c`”

This will show command error. Here, we haven’t kept a space between “`gcc`” and “`-o`”. So, our command looks like “`gcc-o`”, which is not understandable by system.

“`gcc -o l1 lecture1.c`”

This is not an error. Here our executable output file will be with name “`l1`”.

Run or execute the program

From slide 2, we can see the second command as

```
./lecture1
```

Here dot (.) represents the present working directory. And slash (/) represents to go inside that directory. Just after the slash (we call is forward-slash), we write a name which we got after successful compilation.

This command will print the output of the program which we have written.

Execution/Run command error

It is been seen that in starting, students make such errors and get confused.

```
. /lecture1
```

And error message will be `./lecture1`: No such file or directory. Here we have kept a space before slash (/).

```
./ lecture1
```

Error message, `./:` Is a directory. Because we have kept a space just after slash (/).

Run command error (continued…)

```
.\lecture1
```

Error message: `.lecture1:` command not found. Since, we have kept a backward-slash (\) instead of forward-slash (/). Backward slash is useful to connect some special characters and spaces. If we are writing any alphabet after backward-slash, it will ignore the backward-slash. So, in above command backward-slash will not work et.al. and it will behave like we are writing linux-command which is “`.lecture1`”.

If our compilation command is as given in slide 6. “`gcc -o l1 lecture1.c`” than we have to execute/run the program with command “`./l1`” instead.

Run command error (continued…)

```
./lecture1.c
```

Error message: `bash: ./lecture1.c: Permission denied`. Since, we have written `.c` in the end of command which prompt the machine to run or ask the machine to run `lecture1.c` file. This file is created by us not by the compiler. We have to run a file created by compiler.

Understanding our first c-program

```
/* .... */
```

In C-programming, anything written after `/*` and before `*/` is known as “comment in the program”.

Compiler ignore the text written as comment. We can write more than one lines too as comment.

We put comment in a program to put extra comments which doesn’t affect the program but make it easy to understand.

We should not use comments in a program in excess. If we will write comments more than required, we will not able to see our actual program.

Header file

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

- In a c-program, few lines start from `#`
- These lines are known as preprocessor commands.
- If we write “include”, just after `#`, it means it tells compiler to include “`stdio.h`” file before going to actual compilation.
- Such included files are known as header files.
- This header contains information used by the compiler when compiling calls to standard input/output library functions such as *`printf`*.

Main function

```
int main(void)
```

is a part of every c-program.

- Program execution starts from main function.
- Parentheses after after `main` indicate that main is a program building block called a function.
- Keyword “`int`” just before main indicates that main “`returns`” an integer (whole number) value.
- Here, we used a keyword “`void`” in parentheses means that main does not receive any information.
- We call small brackets as “parentheses”.

Brace and block

```
{  }
```

- A left brace, `{`, begins the body of every function.
- A right brace, `}`, ends each function.
- Braces will be always in pair (left and right).
- Portion of the program between the braces is called “block”.

Output action

```
printf("Hello world!!!\n");
```

- This instructs the computer to perform an action.
- This action says computer to print the string of characters on screen which are marked by quotation marks.
- Here full line can be understood in 3 parts.
  - First part is `printf` just before parentheses.
  - 2nd part is in between parentheses within quotation marks.
  - Third part is in the end of statement as semicolon `;`
- Normally text written between quotation marks will appear as it is written. But if we keep few special characters as backslash (\), percentage (%).
- Here in our example, it prints “`Hello world!!!`” in output.

Output action continue….

- Every statement must end with a semicolon (`;`)
- This is also known as the statement terminator.
- When we write special characters (`\` or `%`) between quotations marks, it looks ahead at next characters.
- In case of backslash (`\`), it looks at only one character just after backslash.
- In our above case, next character is `n`, which suggest to go to next line.
- Combination of backslash (`\`) and next character is known as *escape sequence*.
- Some common escape sequences can be found on next page.

Escape sequences

`\a`	→ Alert	→ It will sound the system bell or beep.
`\b`	→ Backslash	→ Insert a backslash character in a string.
`\f`	→ Formfeed	→ It will take you on the same column in next line.
`\n`	→ Newline	→ Position the cursor at the beginning of the next line.
`\r`	→ Carriage Return	→ Take back to you in the beginning of the same line.
`\t`	→ Horizontal Tab	→ Leave a tab horizontally.
`\v`	→ Vertical Tab	→ It will leave empty vertical tab.
`\\`	→ Backslash	→ Insert a backslash character in a string.
`\’`	→ Single quotation mark	→ Insert a single-quote character in a string.
`\”`	→ Double quotation mark	→ Insert a double-quote character in a string.
`\?`	→ Question mark	→ Insert a question mark in a string.

Exit the program

```
return 0;
```

- It will be included in every main function.
- It means exit a function.
- For now, read it as the successful ending of the program. We will study fruther uses of `return` in coming lectures.
- If we will write “`void`” instead of “`int`” just before “`main`”, then we should not use “`return`”.
- For now, we should make this as our habit to use “`return 0`” at the end of the program.

Other ways to write the same program (1)

Here things will be shown only for compiler:

```
#include <stdio.h>
int main()
{
   printf("Hello world!!!\n");
   return 0;
}
```

We can write the same program without using any comment in program.

Other ways to write the same program (2)

In less number of lines:

```
#include <stdio.h>
int main(){ printf("Hello world!!!\n");  return 0;}
```

We can write whole program in single line.

Other ways to write the same program (3)

Here things will be shown only for compiler:

```
#include <stdio.h>

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

We can start braces just after parentheses.

Other ways to write the same program (4)

Here things will be shown only for compiler:

```
----------------------------------------------------------------------

#include                   <stdio.h>
int          main (     )       {
                 printf (           "Hello world!!!\n"      )    ;
                 return           0;
}
----------------------------------------------------------------------
```

We can make spaces between parentheses, braces and on many other places.

Other ways to write the same program (5)

Here things will be shown only for compiler:

```
#include <stdio.h>

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

We can put semicolon “;” in next line before new text instead of same line.

Other ways to write the same program (6)

Here things will be shown only for compiler:

```
#include <stdio.h>

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

We can use more than one semicolon (;).

Other ways to write the same program (6)

Here things will be shown only for compiler:

```
#include <stdio.h>

int main(){
   printf("Hello ");

   printf("world!!!\n");
   return 0;
}
```

We can use more than one `printf` statements.

There are many other ways to write the same code. Reader can try others.

Common mistakes (1)

```
#includ <stdio.h>

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

Error message: invalid preprocessing directive #includ

Here spelling of keyword “include” in first line is wrong.

Common mistakes (2)

```
#include < stdio.h>

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

Error message: Unwanted space in the name of header file name.
Other similar mistakes are: `<stdio .h>`, `<stdio. h>`, `<stdio.h >`

- In first one, there is a space just before “ `.h`”
- In second one, there is a space between dot “`.`” and “`h>`”.
- In third one, there is a space just after “`.h`” and before “`>`”

Common mistakes (3)

```
#include <stdio.h>

intmain(){
   printf("Hello world!!!\n");
   return 0;
}
```

Error message: undefined reference to `main'

Here, program is not able to find “`main`” function. Because we have mixed two keywords “`int`” and “`main`” as single word.

Common mistakes (4)

```
#include <stdio.h>

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

Error message: expected ‘=’, ‘,’, ‘;’, ‘asm’ or ‘__attribute__’ before ‘{’ token

Here “`main`” can not be read as a function. Because to show “`main`” as a function, we should use parentheses just after keyword “`main`”.

We should write “`main()`” insted of “`main`”.

Common mistakes (5)

```
#include <stdio.h>

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

Error message: stray ‘\’ in program and expected ‘)’ before ‘n’

In our program, we have “`\n`” outside the quotation marks. We should always write it inside quotation marks.

Common mistakes (5)

```
#include <stdio.h>

int main(){
   printf("Hello

world!!!\n");
   return 0;
}
```

Error message: missing terminating "

We should not write direct return key in between constant string.

Common mistakes (6)

```
#include <stdio.h>

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

Error message: expected ‘;’ before ‘return’

It suggests to keep a semicolon just before “`return`”. So, there should be a semicolon (`;`) before “`return`” and after completing “`prinf`” parentheses.

Common mistakes (7)

```
#include <stdio.h>

int main(){
   printf("Hello world!!!\n");
   return0;
}
```

Error message: ‘return0’ undeclared (first use in this function)

There should be a space between “`return`” and zero “`0`”.

Common mistakes (8)

```c
#include <stdio.h>

int main(){
   printf("Hello world!!!\n");
   return o;
}
```

Error message: ‘o’ undeclared (first use in this function)

Here, we have written English character “O” in lower case instead of zero (`0`).

Be careful in “`O`”, “`o`” and “`0`”.

Common mistakes (8)

```
#include <stdio.h>

int main(){
   Printf("Hello world!!!\n");
   return o;
}
```

Error message: implicit declaration of function ‘Printf’

Here, we have written `Printf`) instead of `printf`. We have to use only lowercase letters.