Course Webpage

- Google Classroom
  - Class Code: i7juva5

Administrivia

- Instructor: Piyus Kedia
- Lectures: Wed, Fri – 9:30 - 11:00 AM – C101
- Office hours: Wed – 3:00 - 4:00 PM – A505, Research Block
  - You can visit during my office hours without an appointment
- TF: Nupur Ahluwalia (nupur@iiitd.ac.in)
- The details of other TAs and their office hours will be shared on the Google Classroom page

Tentative lecture plan

- Introduction to C
- Revision of recursion (Factorial, Fibonacci sequence, Tower of Hanoi)
- Time complexity of algorithms (Big-Oh notation)
- Array (search and sorting algorithms)
- Stacks, queues, and lists
- Trees
- Binary-tree (Binary search trees, AVL trees, B-trees)
- Heap Trees (Priority queues, Heap sort)
- Hash tables
- Sets (Disjoint sets, Union, Find algorithms)
- Graphs (BFS, DFS, Topological sort, Minimum spanning trees, Shortest path, Huffman coding)

Grading components

MidSem Theory	30
EndSem Theory	35
Take home + in lab assignments	20
Surprise Quizzes + Homework	15
Bonus Assignments	Nil / May get an A+

References

- Introduction to Algorithms, Third Edition
  - Thomas H. Cormen. Charles E. Leiserson. Ronald L. Rivest, Clifford Stein
- Data Structures & Algorithms Analysis in C, Second Edition
  - Mark Allen Weiss
- Data Structures & Algorithms Made Easy
  - Narasimha Karumanchi
- Online C reference: https://users.cs.cf.ac.uk/Dave.Marshall/C/CE.html

Academic dishonesty

- Please refer to the institute’s plagiarism policy to know more
  - https://www.iiitd.ac.in/academics/resources/acad-dishonesty

Other stuffs

- Please bring a paper a pen in each class for the surprise quiz

- Use of laptops, tablets, and similar devices is not allowed during the lecture

- Late entry after 9:35 AM is not permitted

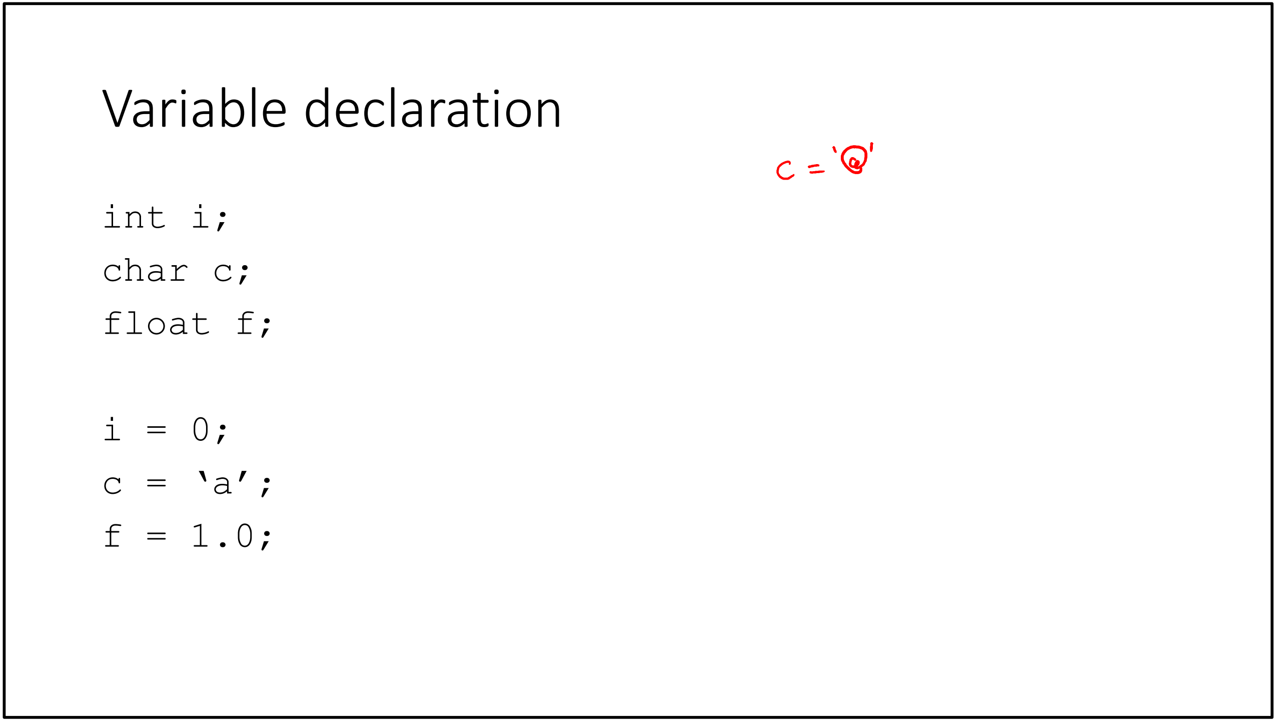
Intro to C

Intro to C

- Types
  - Every variable in a C program needs a type declaration
    - Unlike Python, in which types are automatically inferred by the compiler
  - Variables of the same type follow some common properties
    - e.g., all the variables of type int store a 4-byte integer value
    - You can use $sizeof(ty)$ to obtain the size of a given type ty

Basic types

TYPE	SIZE (TYPICAL SIZE)	RANGE (TYPICAL SIZE)
char	1	-128 to 127
unsigned char	1	0 to 255
short int	2	-32,768 to 32,767
unsigned short	2	0 to 65,535
int	>=2 (4)	-2,147,483,648 to 2,147,483,647
unsigned int	>=2 (4)	0 to 4,294,967,295
long long int	8	-2^{63} to $2^{63}-1$
unsigned long long	8	0 to $2^{64}-1$
float	4	1.175494e-38 to 3.402823e+38
double	8	2.225074e-308 to 1.797693e+308



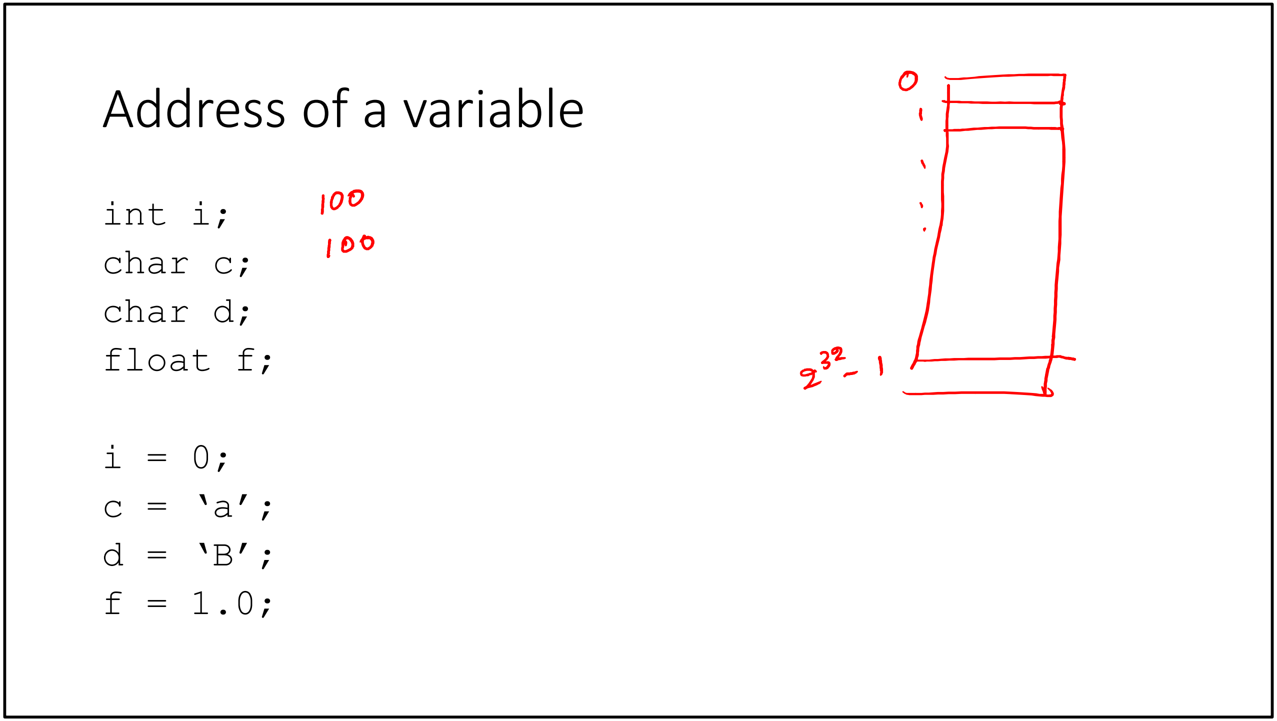
Variable declaration

c = '@'

```
int i;
char c;
float f;

i = 0;
c = ‘a’;
f = 1.0;
```

Every variable must be declared before its use. The declaration must specify the type of the variable.



Address of a variable

```
int i;        100
char c;       100
char d;
float f;

i = 0;
c = 'a';
d = 'B';
f = 1.0;
```

The variables are stored in RAM. Each byte of the RAM has a unique address. Suppose you have 4GB RAM; the total number of bytes would be 4*1024*1024*1024. Now, if the address of the first byte of the RAM is zero, the address of the second byte would be one, and so on. Similarly, the address of the 4*1024*1024*1024th byte would be 4*1024*1024*1024 -1. When you write a program, you don't decide at which address the variable will be stored in the RAM. It is decided by the compiler and the OS. However, all the bytes corresponding to a data type are stored at consecutive addresses. For example, the size of a variable, say "x", of type "int" is four bytes. If the compiler decides to store "x" at an address "xa", then the first byte of "x" is stored at address "xa", the second byte of "x" is stored at "xa+1", the third byte at "xa+2", and the fourth byte at "xa+3".

Address of a variable

```
int i;
char c;
char d;
float f;

i = 0;
c = ‘a’;
d = ‘B’;
f = 1.0;
```

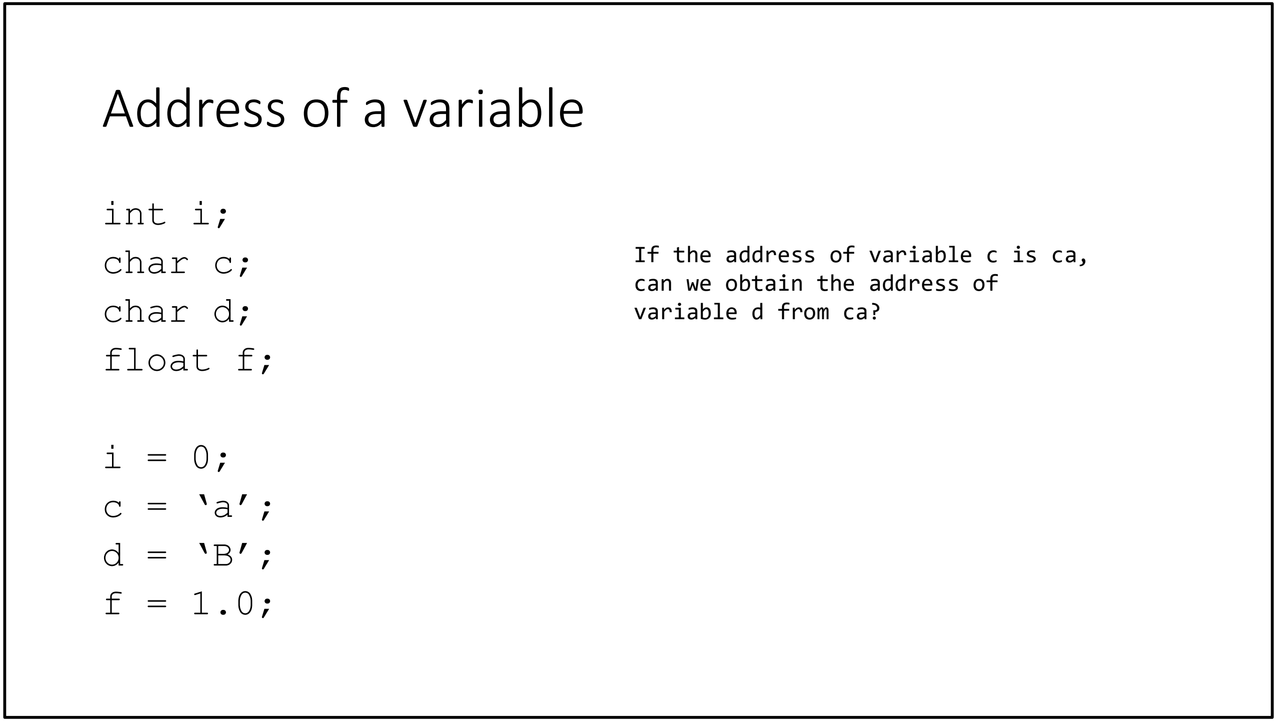
```
If the address of variable i is ia,
is it possible that the address of
variable c is ia+3?
```

No

```
If the address of variable i is ia,
is it possible that the address of
variable c is ia+7?
```

Yes

The address of the variable c can’t be ia+3 because the variable i is four bytes long, and the addresses ia to ia+3 are occupied by i.



Address of a variable

```
int i;
char c;
char d;
float f;

i = 0;
c = ‘a’;
d = ‘B’;
f = 1.0;
```

```
If the address of variable c is ca,
can we obtain the address of
variable d from ca?
```

The compiler/OS can store the variable d at any address. There is no correlation between ca and the address of d.

“char” type

- The size of a variable of type “char” is one byte
- “char” can store one alphabetic letter or symbol

```
char ch;
ch = ‘a’;  // valid
ch = ‘A’;  // valid
ch = ‘@’;  // valid
ch = ‘ab’; // invalid
```

“char” type

- How can we store a character in one byte?

“char” type

- How can we store a character in one byte?
  - We can store -128 to 127 in one byte
  - Total number of characters is less than 127
  - We can assign a unique value between 0-127 to each of the characters
  - The corresponding encoding is also called ASCII code, e.g.,
    - ASCII value of ‘$’ is 36
    - ASCII value to ‘A’ is 65, ‘B’ is 66, and so on.
    - ASCII value of ‘a’ is 97, ‘b’ is 98, and so on.
    - ASCII value of ‘<’ is 60
    - etc.

Array type

- An array is a collection of elements of the same type

```
int arr[100]; // declaring an array of 100 elements
// the first element is indexed using 0, second is 1,
// and so on ..
a[2] = 20; // updating the 3rd element
a[139] = 2; // allowed but the behavior is undefined
// it might corrupt the values of the other variables
```

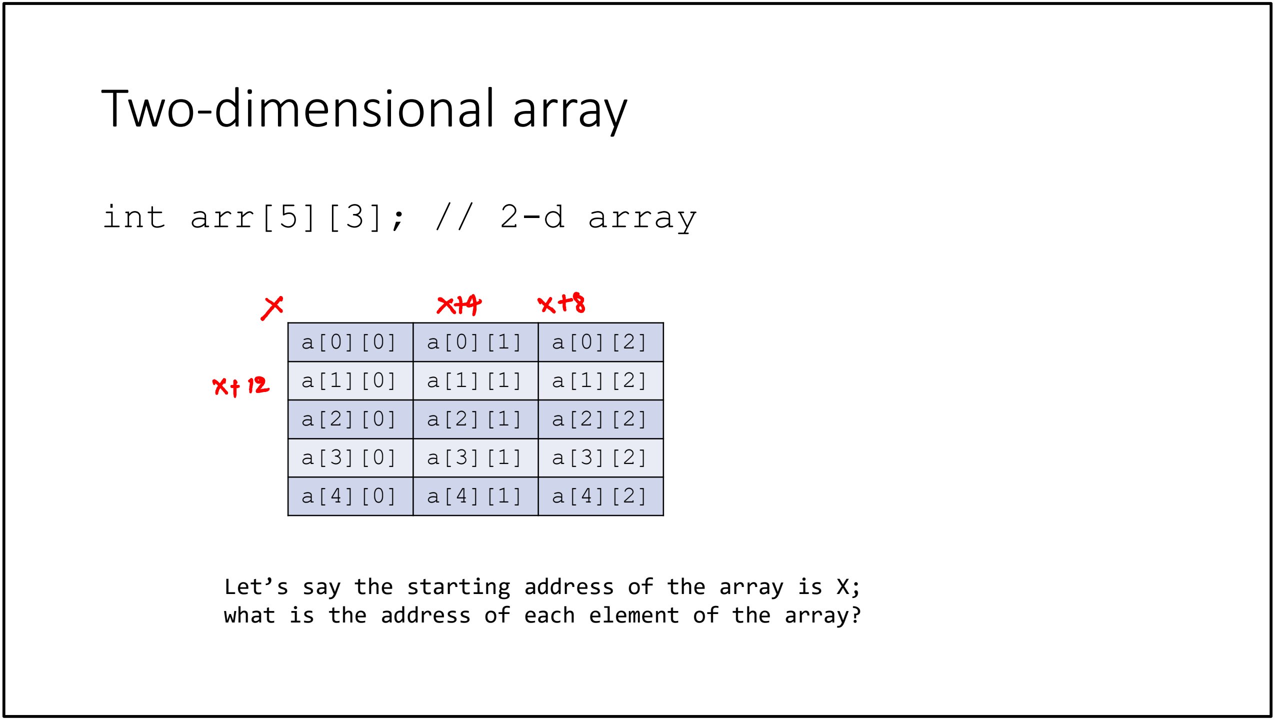
One-dimensional array

```
int arr[10]; // 1-d array
```

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
X	X+4	X+8	X+12	. . . .					

Let's say the starting address of the array is X;
what is the address of each element of the array?

All the elements of an array are stored at consecutive addresses.



Two-dimensional array

```
int arr[5][3]; // 2-d array
```

	X	X+4	X+8
	a[0][0]	a[0][1]	a[0][2]
X+12	a[1][0]	a[1][1]	a[1][2]
	a[2][0]	a[2][1]	a[2][2]
	a[3][0]	a[3][1]	a[3][2]
	a[4][0]	a[4][1]	a[4][2]

Let's say the starting address of the array is X;
what is the address of each element of the array?

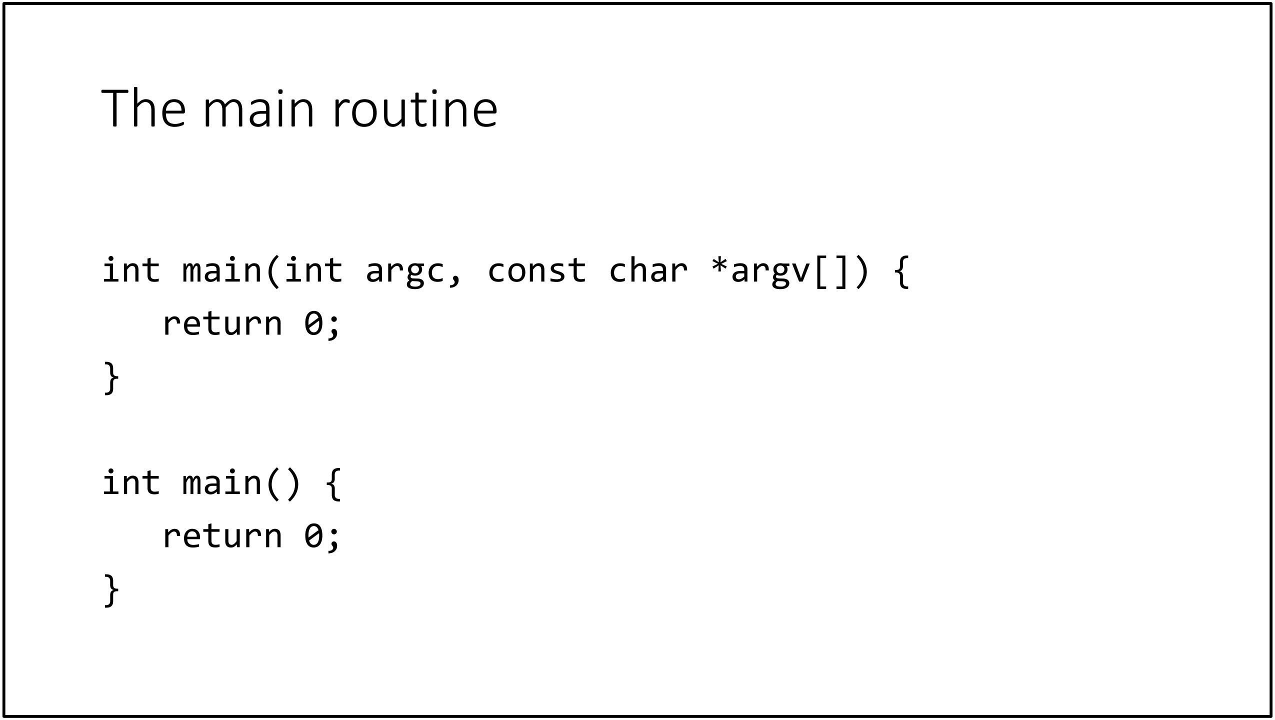
All the elements of a two-dimensional array are stored at consecutive addresses. If we know the address of any element of the array, we can calculate the address of the other elements of the array. All the elements in the rows are stored at consecutive addresses. Rows are also stored at consecutive addresses.

n-dimension array

- Similarly, you can have n-dimensional array

Type

- All variables and functions must be declared before their use
- The function declaration contains the types of arguments and the return value
- Every C program must have the $main$ function



The main routine

```
int main(int argc, const char *argv[]) {
   return 0;
}

int main() {
   return 0;
}
```

You can define the main routine in two ways, as listed on this slide. The first definition is needed when your program takes command line input. Here, argc corresponds to the number of arguments, and argv is an array that contains the argument strings.

Hello world!

```
int main(){
    printf(“Hello World!”);
    // printf prints to console
    return 0;
}
```

Hello world!

```
int main(){
    printf(“Hello World!”);
    // printf prints to console
    // compiler gives a warning because
    // printf is not declared
    return 0;
}
```

Hello world!

```
#include <stdio.h>
// stdio.h contains a declaration for printf

int main(){
   printf(“Hello World!”);
   // printf prints to console
   return 0;
}
```

Compiling and running

```
gcc hello.c
./a.out
```

```
gcc hello.c –o hello
./hello
```

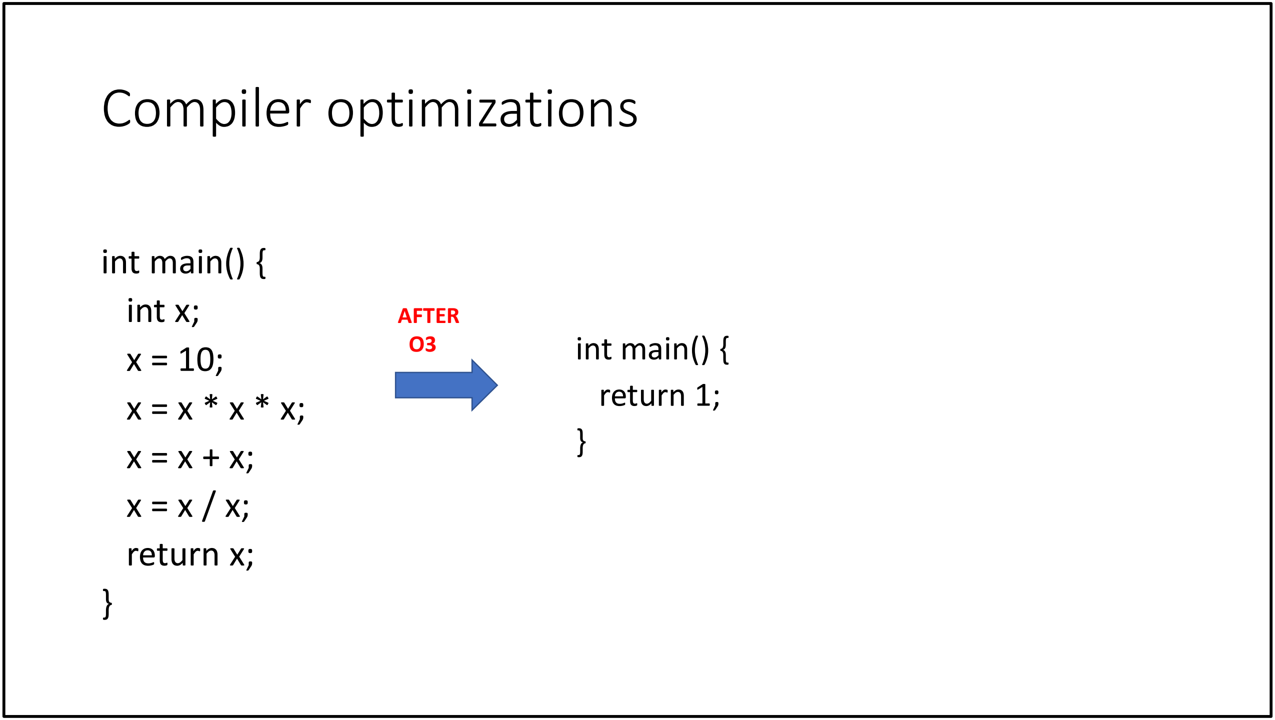
Compiling and running

```
gcc –O3 hello.c      // -O3 corresponds to the optimization level
./a.out
```

```
gcc –O3 hello.c –o hello
./hello
```

Compiler optimizations

```
int main() {
  int x;
  x = 10;
  x = x * x * x;
  x = x + x;
  x = x / x;
  return x;
}
```



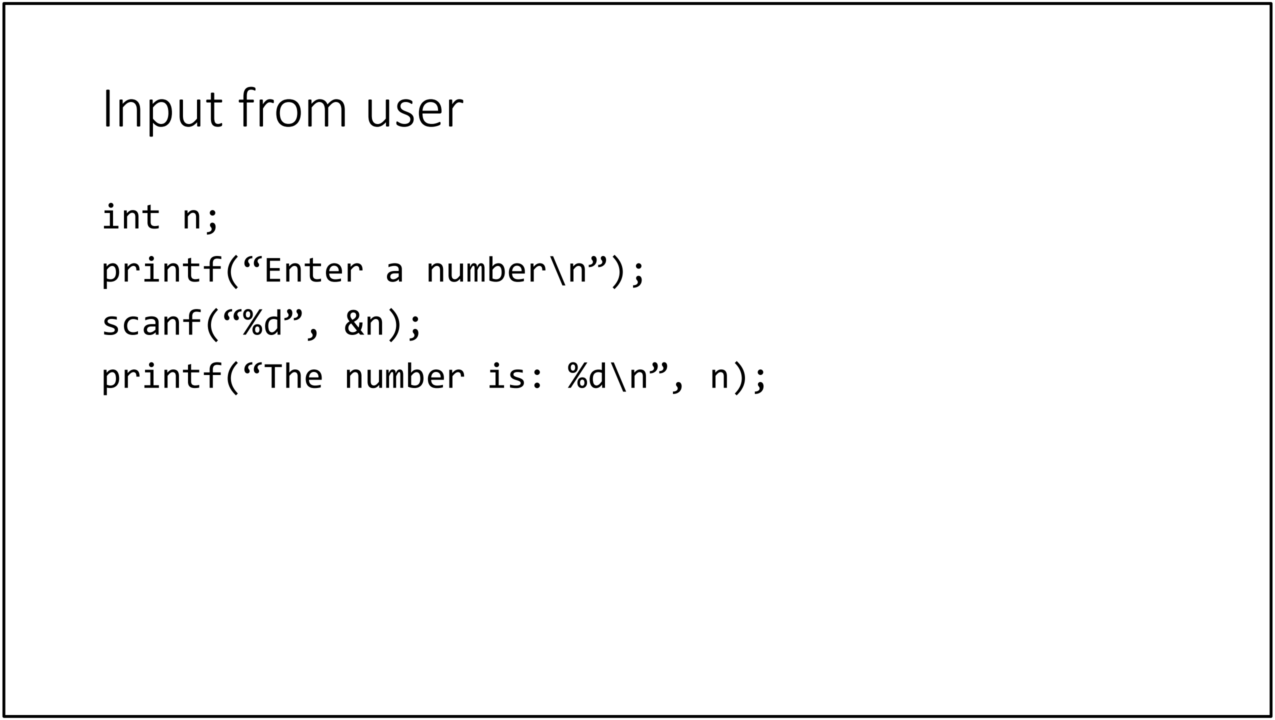
Compiler optimizations

```
int main() {
   int x;
   x = 10;
   x = x * x * x;
   x = x + x;
   x = x / x;
   return x;
}
```

AFTER
O3

```
int main() {
   return 1;
}
```

Notice that the only observable behavior of the main routine in LHS is the return value, which is always one. Therefore, when optimizations are enabled, the compiler can transform the code in LHS to the code in RHS.



Input from user

```
int n;
printf(“Enter a number\n”);
scanf(“%d”, &n);
printf(“The number is: %d\n”, n);
```

The ‘&’ operator is used to get the address of the variable (i.e., the address at which the variable is stored in the RAM). The scanf routine will read the input from the keyboard and store it in the address of n. After the scanf returns that when we print the value of n, it will print the value entered by the user.

Loop (for)

```
for (initialization; condition; update) {
   // body of the for loop
}
```

Loop (for)

```
int i;
for (i = 0; i < 5; i += 1) {
   printf(“Hello world!”);
}
```

i=0 i<5

Hello world!

i=i+1 i=1 i<5

Hello world!

i=i+1 i=2 i<5

Hello world!

i=i+1 i=3 i<5

Hello world!

i=i+1 i=4 i<5

Hello world!

i=i+1 i=5 i<5

The initialization is done only once on entry

The loop body is executed if the condition is satisfied

The update is done after the execution of the loop body

The loop condition is again checked after the update

Loop (while)

```
while (condition) {
   // body of the while loop
}
```

Loop (while)

```
int i = 0;
while (i < 5) {
  printf(“Hello world!”);
  i += 1;
}
```

Loop body is executed if the condition is true

Condition is again checked after the execution of the loop body

i=0
i<5
Helloworld!
i=i+1   i=1   i<5
Hello world!
i=i+1   i=2   i<5
Hello world!
i=i+1   i=3   i<5
Hello world!
i=i+1   i=4   i<5
Hello world!
i=i+1   i=5   i<5

Conditional

```
if (condition) {
  // if body
}
```

```
if (condition) {
  // if body
}
else {
  // else body
}
```

```
if (condition) {
  // if body
}
else if (condition) {
  // else-if body
}
else {
  // else-body
}
```

Conditional

```
int main() {
  int x;
  printf(“Enter a number\n”);
  scanf(“%d”, &x);
  if ((x % 2) == 0) {
     printf(“x is even\n”);
  }
  return 0;
}
```

What will be the output when input is:

10 x is even

13

Conditional

```
int main() {
   int x;
   printf(“Enter a number\n”);
   scanf(“%d”, &x);
   if ((x % 2) == 0) {
       printf(“input is even\n”);
   }
   else {
      printf(“input is odd\n”);
   }
   return 0;
}
```

What will be the output when input is:

10 input is even

13 input is odd

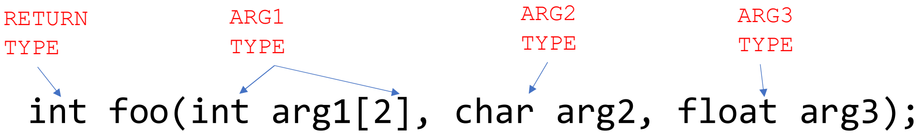
Conditional

```
int main() {
   int x;
   printf(“Enter a number\n”);
   scanf(“%d”, &x);
   if ((x % 2) == 0) {
       printf(“input is even\n”);
   }
   else if (x < 30) {
       printf(“input is less than 30\n”);
   }
   else {
      printf(“input is odd\n”);
   }
   return 0;
}
```

What will be the output when input is:

10	input is even
13	input is less than 30
50	input is even
51	input is odd

Function declaration



RETURN TYPE

ARG1 TYPE

ARG2 TYPE

ARG3 TYPE

```
int foo(int arg1[2], char arg2, float arg3);
```

A function must be declared/defined before its use

Function definition

```
int foo(int arg1[2], char arg2, float arg3) {
  // function body
}
```

A function must be declared/defined before its use