Mr C is Pointing to Something Useful

ESC101: Fundamentals of Computing

Purushottam Kar

Announcements

- Advanced Track meetings on Friday – check your mail
- Emergency joint tutorial for sections B1 and B13
  - Friday, 28 September 2018, 12noon (same time as usual)
  - L19 (not the same room)
  - Only for B1, B13 – rest of sections go as usual to TB room
  - Special arrangement only for this week – next week as usual in TB for all

The sizeof various variable types

The sizeof various variable types

8 bits □ make a byte □□□□□□□□

The sizeof various variable types

8 bits □ make a byte □□□□□□□□

char takes 1 byte = 8 bits

The sizeof various variable types

8 bits □ make a byte □□□□□□□□

char takes 1 byte = 8 bits

= □□□□□□□□

The sizeof various variable types

8 bits □ make a byte □□□□□□□□

char takes 1 byte = 8 bits

Max value in a char is 127 = $2^{(8-1)}-1$

= □□□□□□□□

The sizeof various variable types

8 bits □ make a byte □□□□□□□□

char takes 1 byte = 8 bits

Max value in a char is 127 = $2^{(8-1)}-1$

= □□□□□□□□

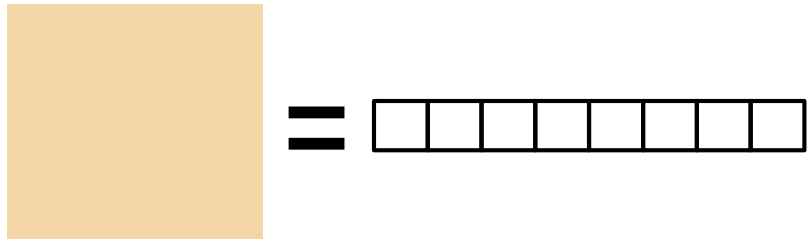
int/float takes 4 bytes = 32 bits

The sizeof various variable types

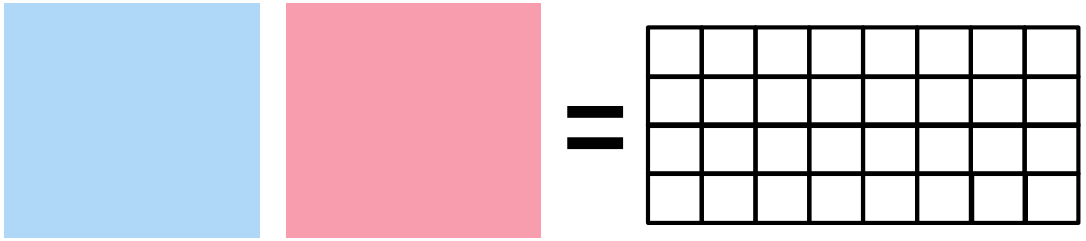
8 bits □ make a byte □□□□□□□□

char takes 1 byte = 8 bits

Max value in a char is 127 = $2^{(8-1)}-1$



int/float takes 4 bytes = 32 bits

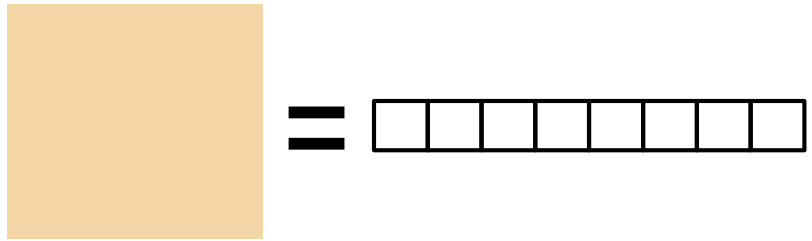


The sizeof various variable types

8 bits □ make a byte □□□□□□□□

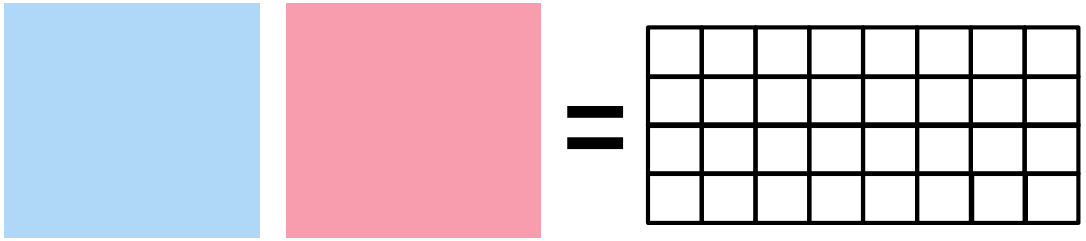
char takes 1 byte = 8 bits

Max value in a char is 127 = $2^{(8-1)}-1$



int/float takes 4 bytes = 32 bits

Max value in int is 2,147,483,647
equal to $2^{(32-1)}-1$ – verify

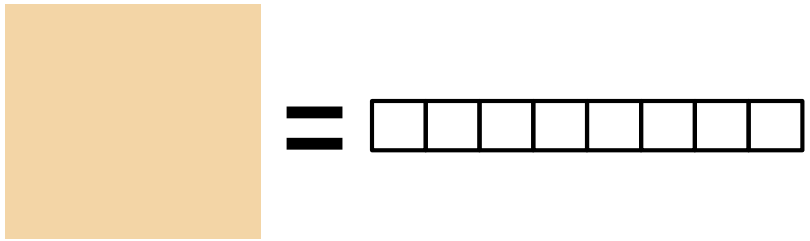


The sizeof various variable types

8 bits □ make a byte □□□□□□□□

char takes 1 byte = 8 bits

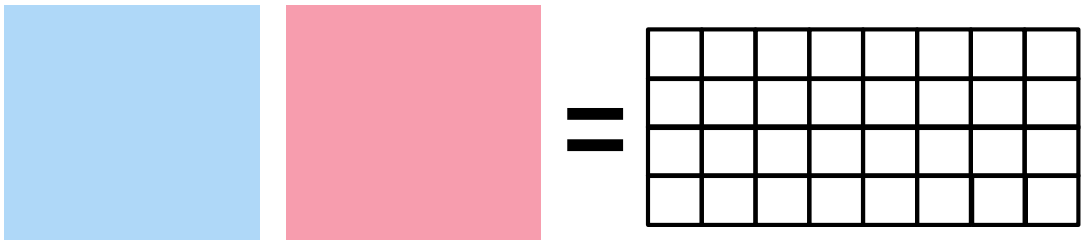
Max value in a char is 127 = $2^{(8-1)}-1$



int/float takes 4 bytes = 32 bits

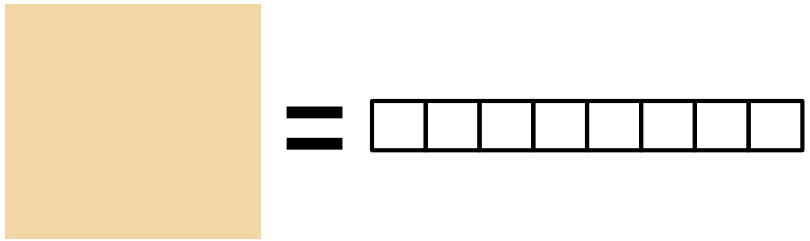
Max value in int is 2,147,483,647
equal to $2^{(32-1)}-1$ – verify

Max value of float discussed later



The sizeof various variable types

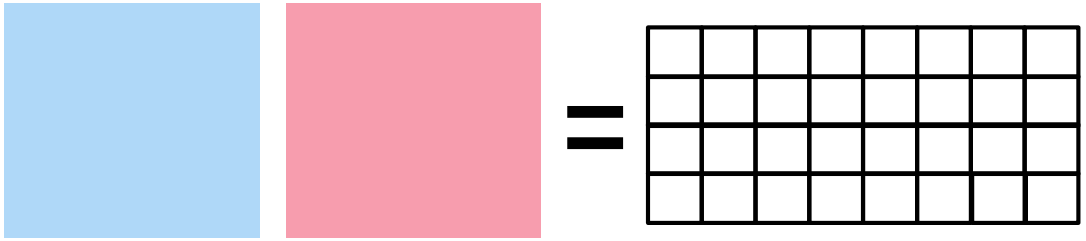
8 bits □ make a byte □□□□□□□□



char takes 1 byte = 8 bits

Max value in a char is 127 = $2^{(8-1)}$-1

int/float takes 4 bytes = 32 bits



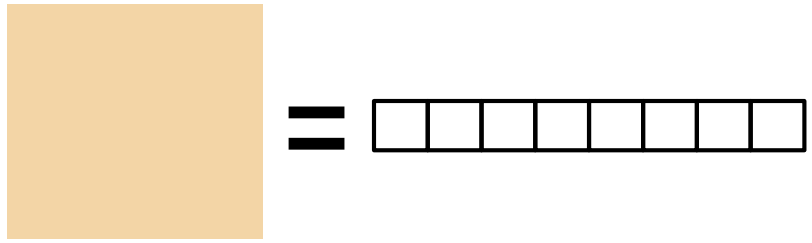
Max value in int is 2,147,483,647
equal to $2^{(32-1)}$-1 – verify

Max value of float discussed later

long/double takes 8 bytes = 64 bits

The sizeof various variable types

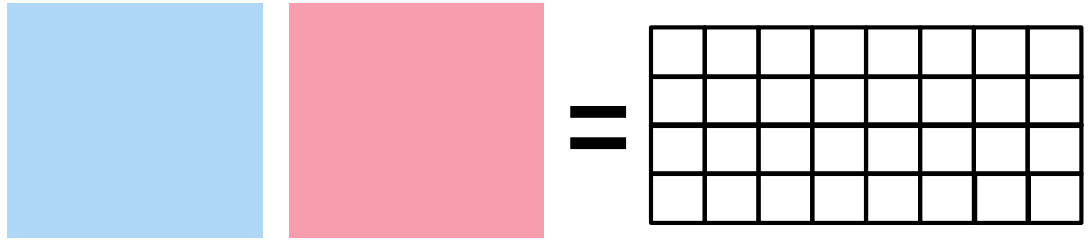
8 bits □ make a byte □□□□□□□□



char takes 1 byte = 8 bits

Max value in a char is 127 = $2^{(8-1)}-1$

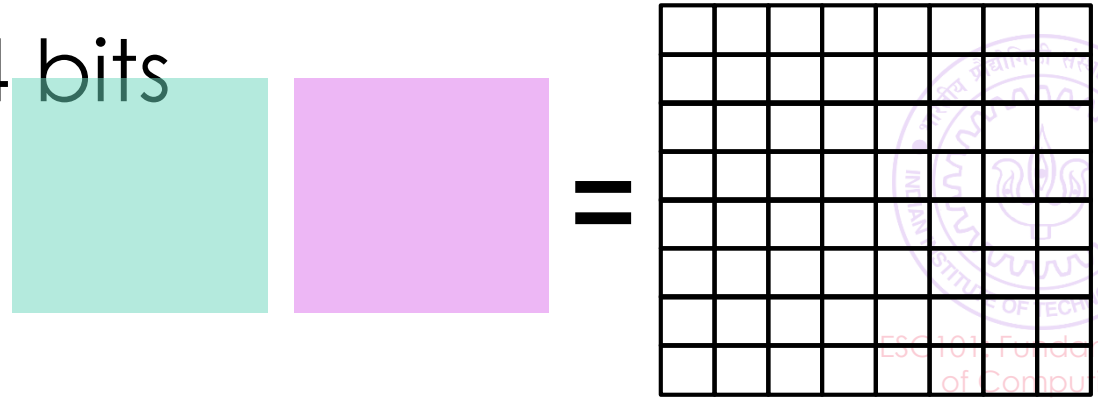
int/float takes 4 bytes = 32 bits



Max value in int is 2,147,483,647
equal to $2^{(32-1)}-1$ – verify

Max value of float discussed later

long/double takes 8 bytes = 64 bits

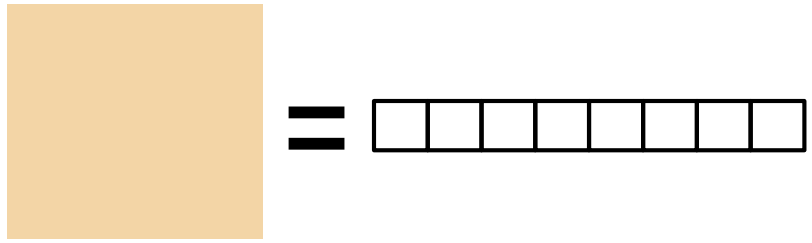


The sizeof various variable types

8 bits □ make a byte □□□□□□□□

char takes 1 byte = 8 bits

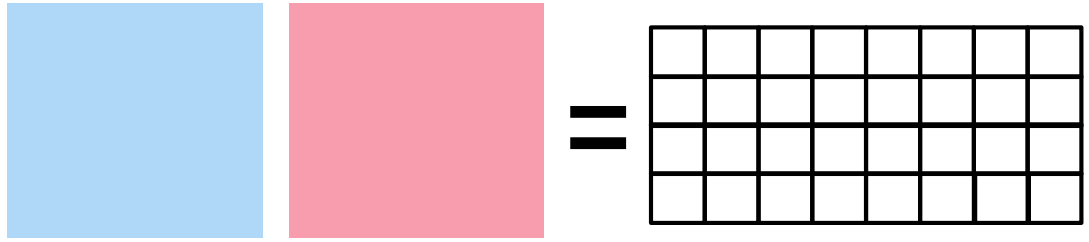
Max value in a char is 127 = $2^{(8-1)}-1$



int/float takes 4 bytes = 32 bits

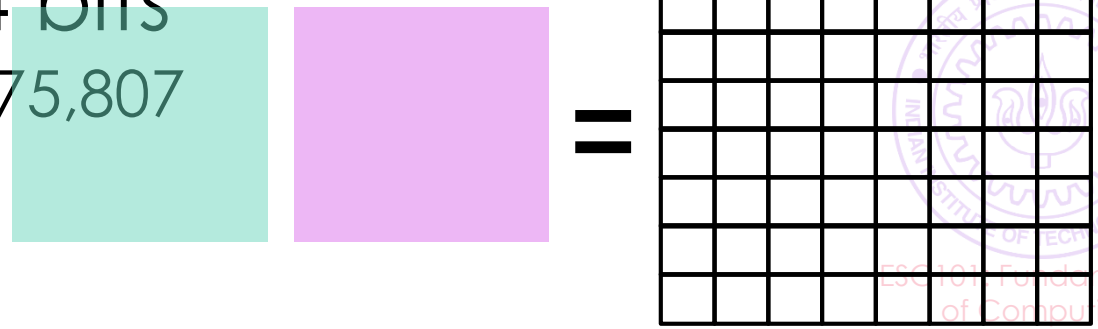
Max value in int is 2,147,483,647
equal to $2^{(32-1)}-1$ – verify

Max value of float discussed later



long/double takes 8 bytes = 64 bits

Max value in long is 9,223,372,036,854,775,807
equal to $2^{(64-1)}-1$ – verify

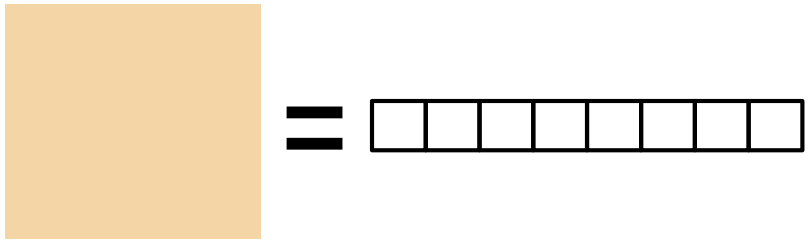


The sizeof various variable types

8 bits □ make a byte □□□□□□□□

char takes 1 byte = 8 bits

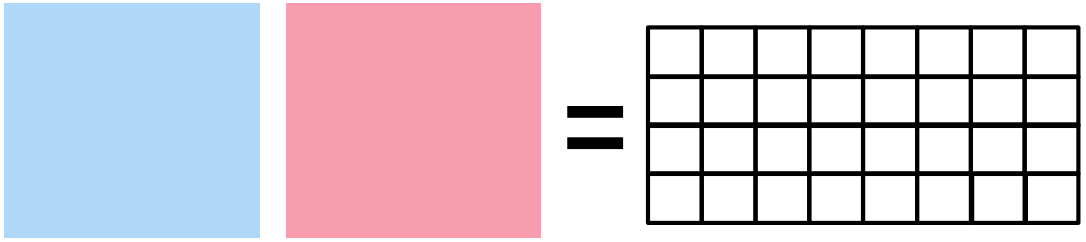
Max value in a char is 127 = $2^{(8-1)}-1$



int/float takes 4 bytes = 32 bits

Max value in int is 2,147,483,647
equal to $2^{(32-1)}-1$ – verify

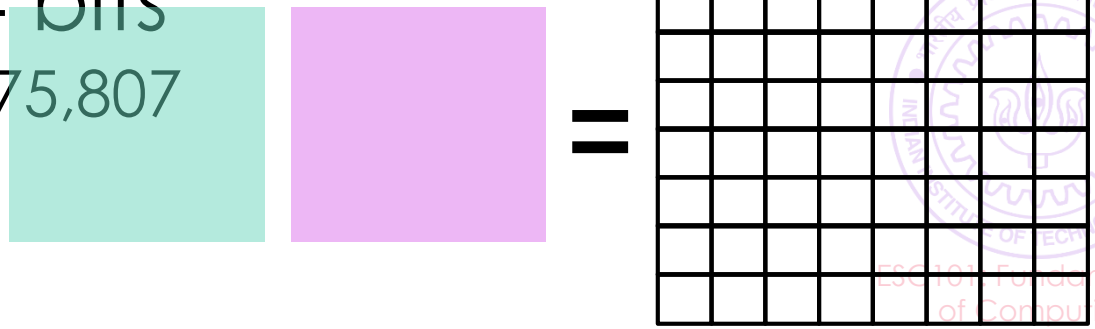
Max value of float discussed later



long/double takes 8 bytes = 64 bits

Max value in long is 9,223,372,036,854,775,807
equal to $2^{(64-1)}-1$ – verify

Max value of double discussed later

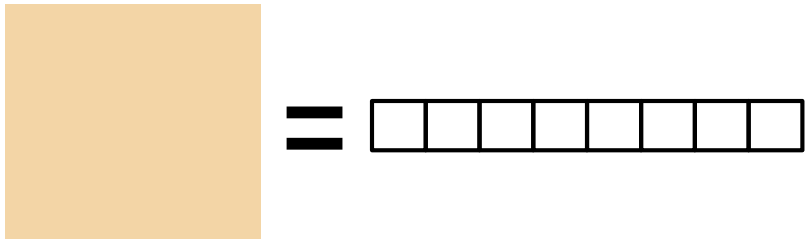


The sizeof various variable types

8 bits □ make a byte □□□□□□□□

char takes 1 byte = 8 bits

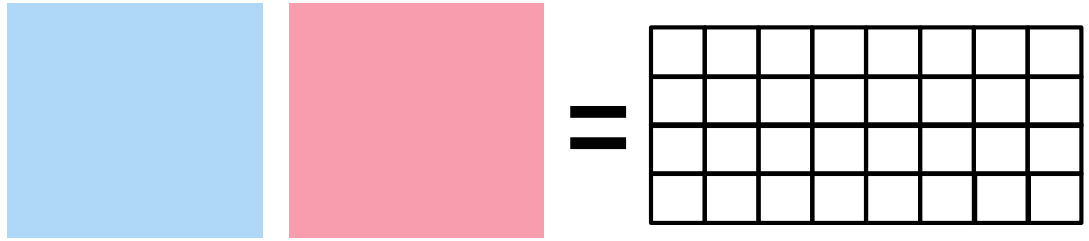
Max value in a char is 127 = $2^{(8-1)}-1$

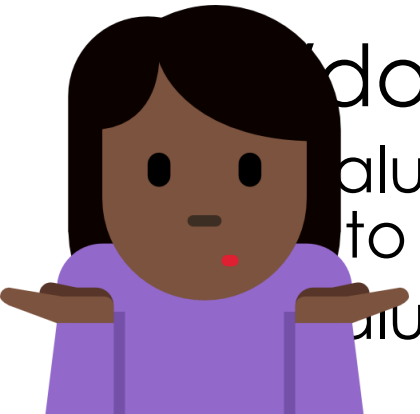


int/float takes 4 bytes = 32 bits

Max value in int is 2,147,483,647
equal to $2^{(32-1)}-1$ – verify

Max value of float discussed later

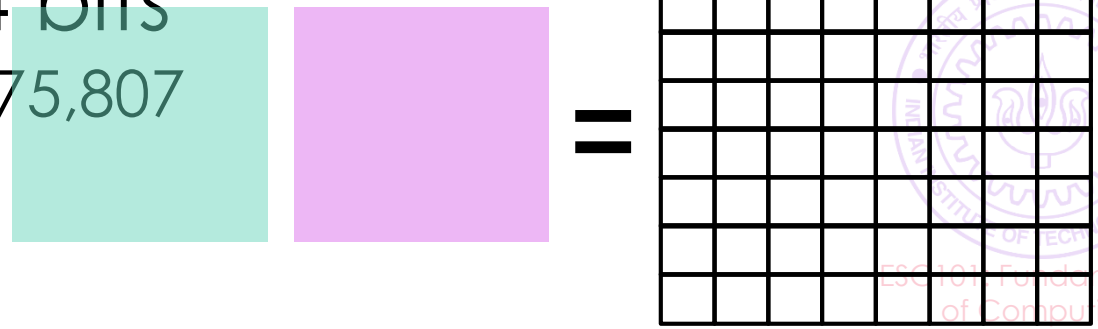




double takes 8 bytes = 64 bits

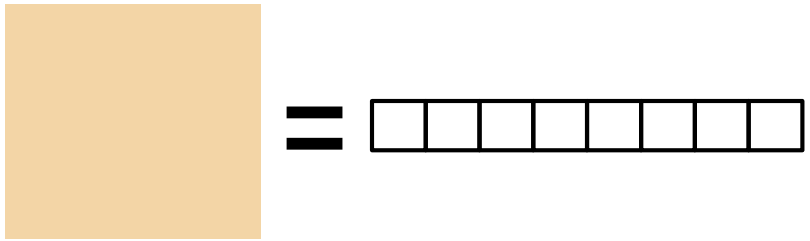
alue in long is 9,223,372,036,854,775,807
to $2^{(64-1)}-1$ – verify

alue of double discussed later



The sizeof various variable types

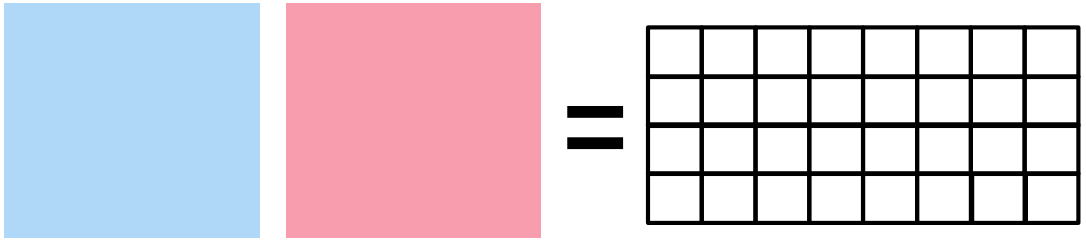
8 bits □ make a byte □□□□□□□□



char takes 1 byte = 8 bits

Max value in a char is 127 = $2^{(8-1)}-1$

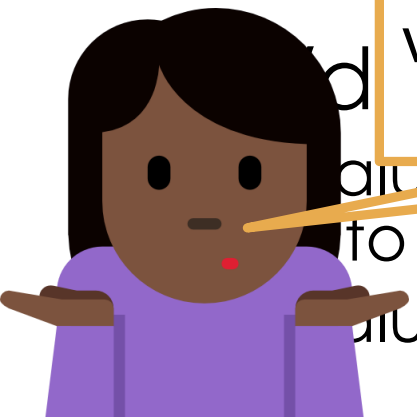
int/float takes 4 bytes = 32 bits



Max value in int is 2,147,483,647
equal to $2^{(32-1)}-1$ – verify

Max value of float discussed later

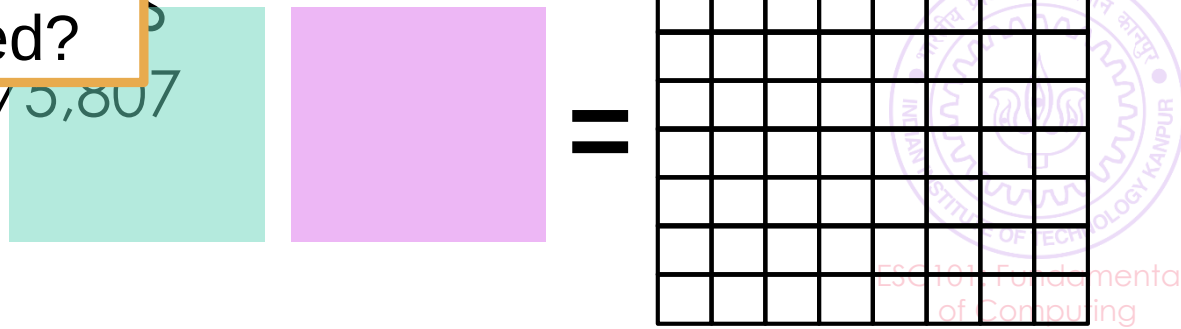
Why is max value for all these variables always $2^{(k-1)}-1$ and not 2^k-1 when there are k bits getting used?



d

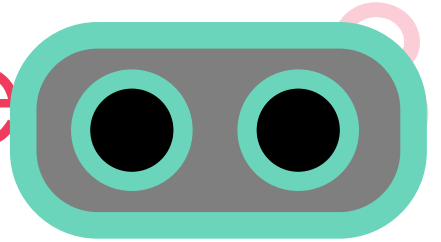
alue i g is 9,223,372,036,854,775,807

to $2^{(64-1)}-1$ – verify



alue of double discussed later

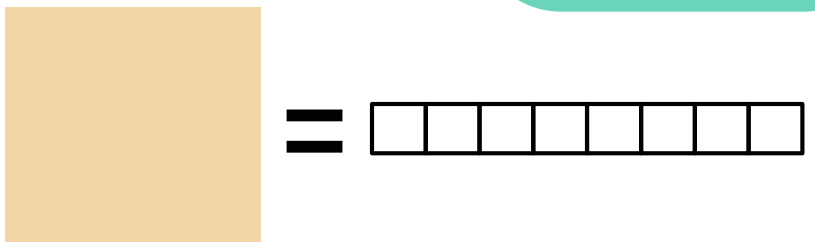
The sizeof various variable type



8 bits □ make a byte □□□□□□□□

char takes 1 byte = 8 bits

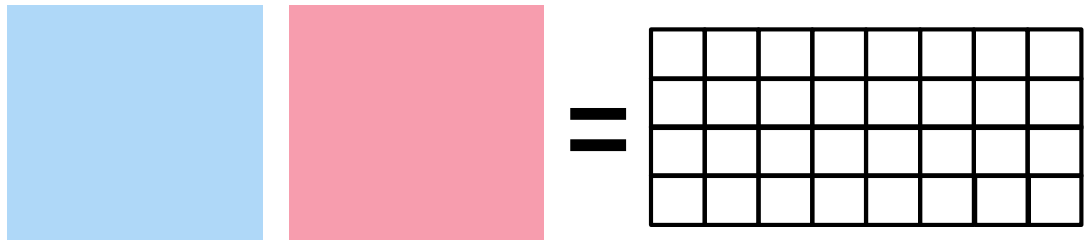
Max value in a char is 127 = $2^{(8-1)}-1$



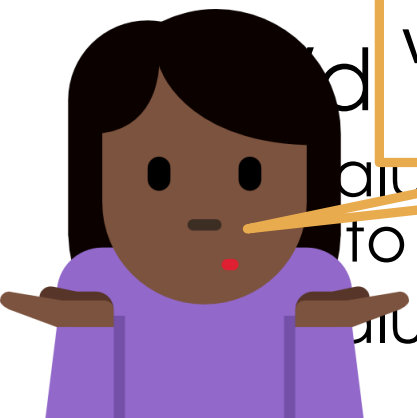
int/float takes 4 bytes = 32 bits

Max value in int is 2,147,483,647
equal to $2^{(32-1)}-1$ – verify

Max value of float discussed later



Why is max value for all these variables always $2^{(k-1)}-1$ and not $2^{k}-1$ when there are k bits getting used?

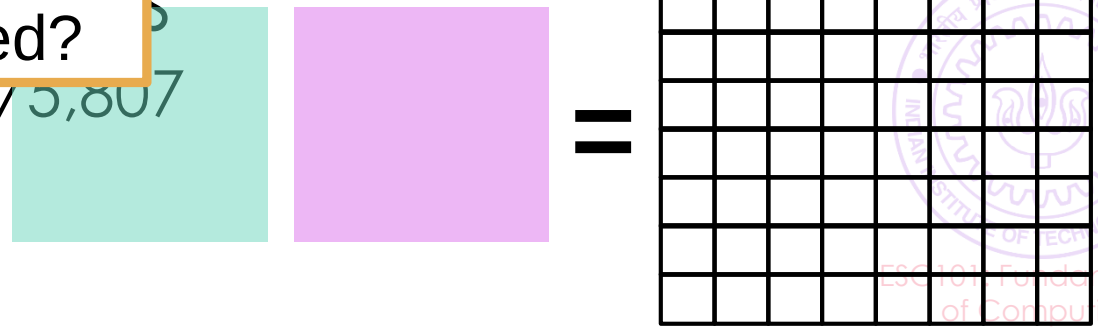


d

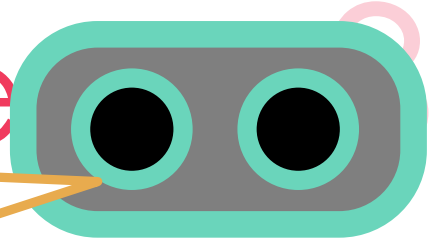
alue i g is 9,223,372,036,854,775,807

to $2^{(64-1)}-1$ – verify

alue of double discussed later



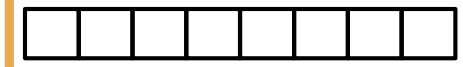
The sizeof various variable type



8 bits □ make

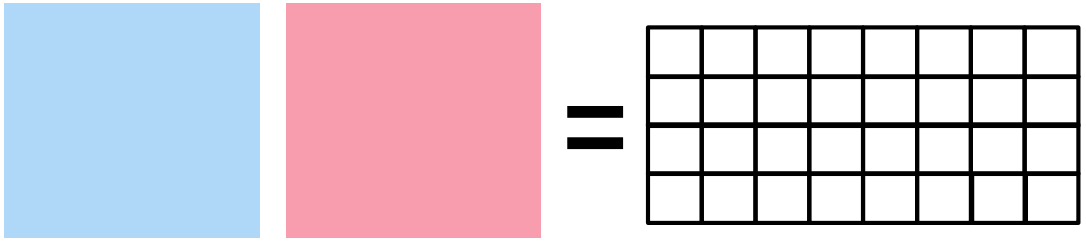
char takes 1 b

This has to do with the way I store negative numbers. Effectively, one bit gets used up in storing the sign of the number so only k-1 bits left to store the magnitude of the number



Max value in a char is 127 = $2^{(8-1)}$-1

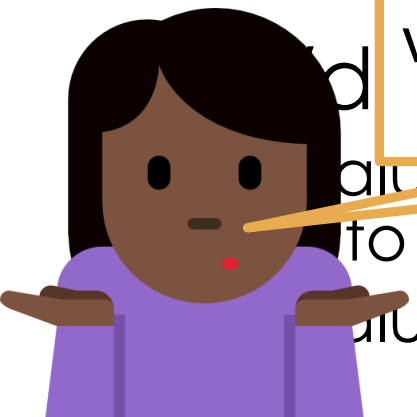
int/float takes 4 bytes = 32 bits



Max value in int is 2,147,483,647
equal to $2^{(32-1)}$-1 – verify

Max value of float discussed later

Why is max value for all these variables always $2^{(k-1)}$-1 and not 2^{k} -1 when there are k bits getting used?



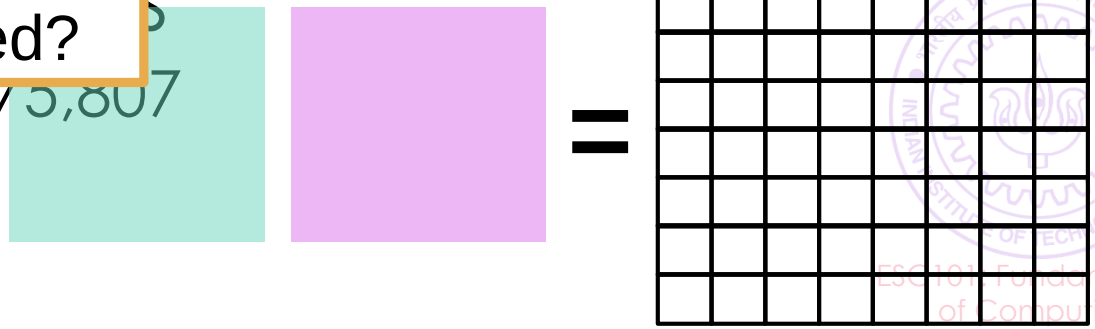
d

s

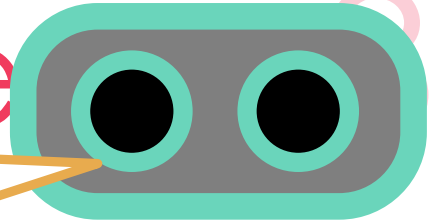
alue i g is 9,223,372,036,854,77 5,807

to $2^{(64-1)}$-1 – verify

lue of double discussed later



The sizeof various variable type

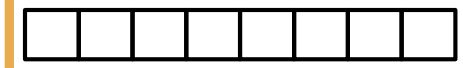


This has to do with the way I store negative numbers. Effectively, one bit gets used up in storing the sign of the number so only k-1 bits left to store the magnitude of the number

8 bits □ make

char takes 1 b

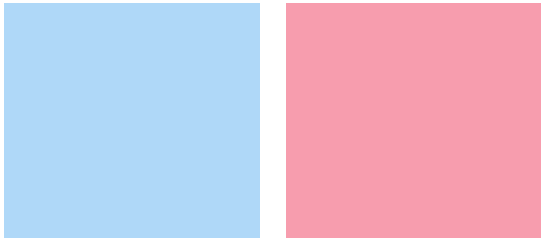
Max value in a char is 127 = $2^{(8-1)}-1$



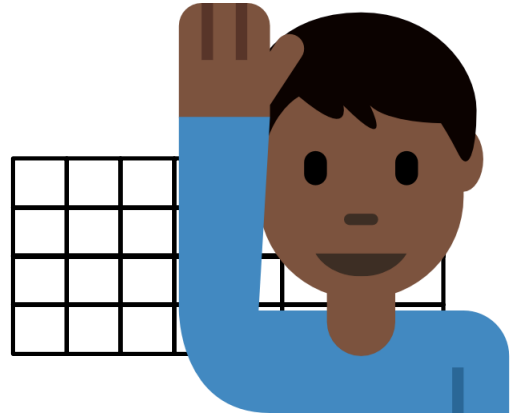
int/float takes 4 bytes = 32 bits

Max value in int is 2,147,483,647
equal to $2^{(32-1)}-1$ – verify

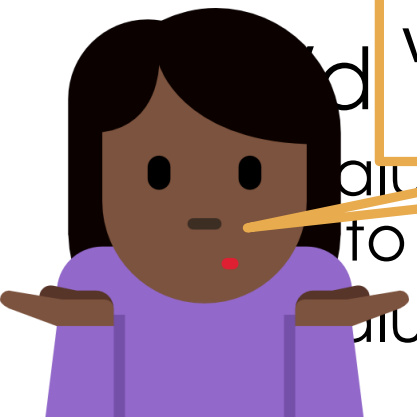
Max value of float discussed later



=



Why is max value for all these variables always $2^{(k-1)}-1$ and not $2^{k}-1$ when there are k bits getting used?



d

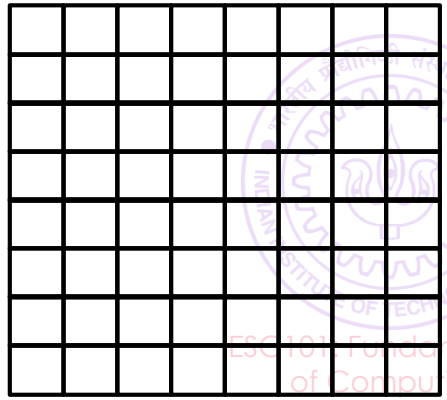
s

alue i g is 9,223,372,036,854,775,807

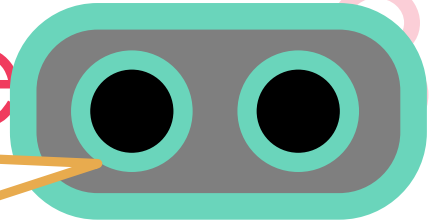
to $2^{(64-1)}-1$ – verify

alue of double discussed later

=



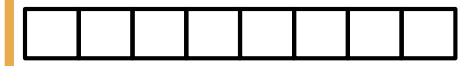
The sizeof various variable type



8 bits □ make

char takes 1 b

This has to do with the way I store negative numbers. Effectively, one bit gets used up in storing the sign of the number so only k-1 bits left to store the magnitude of the number



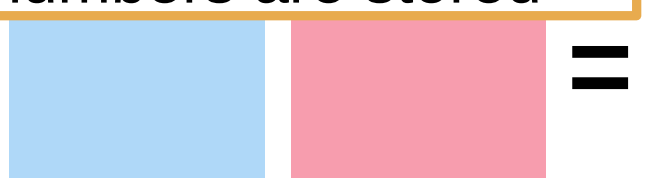
Max value in a char is 127 = $2^{(8-1)}$-1

We will wait a few weeks to learn how negative numbers are stored

int/float takes 4 bytes = 32 b

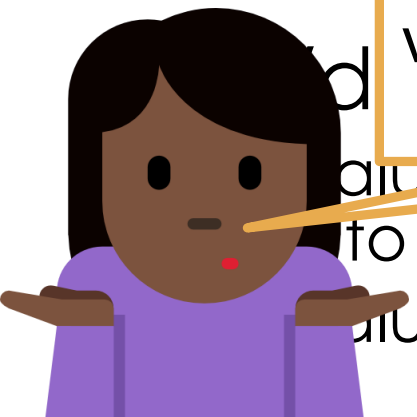
Max value in int is 2,147,483,647 equal to $2^{(32-1)}$-1 – verify

Max value of float discussed later



=

Why is max value for all these variables always $2^{(k-1)}$-1 and not 2^{k} -1 when there are k bits getting used?



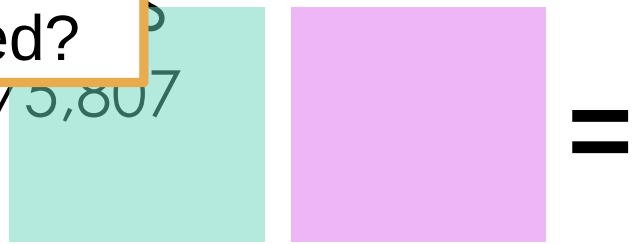
d

s

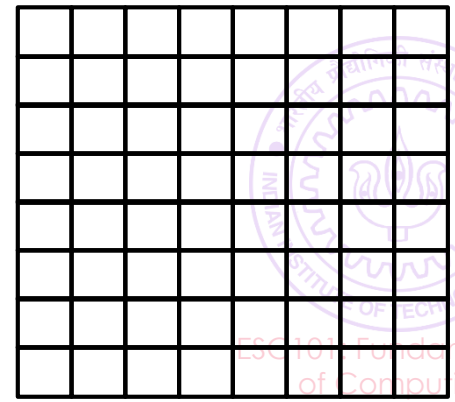
alue i g is 9,223,372,036,854,775,807

to $2^{(64-1)}$-1 – verify

alue of double discussed later



=



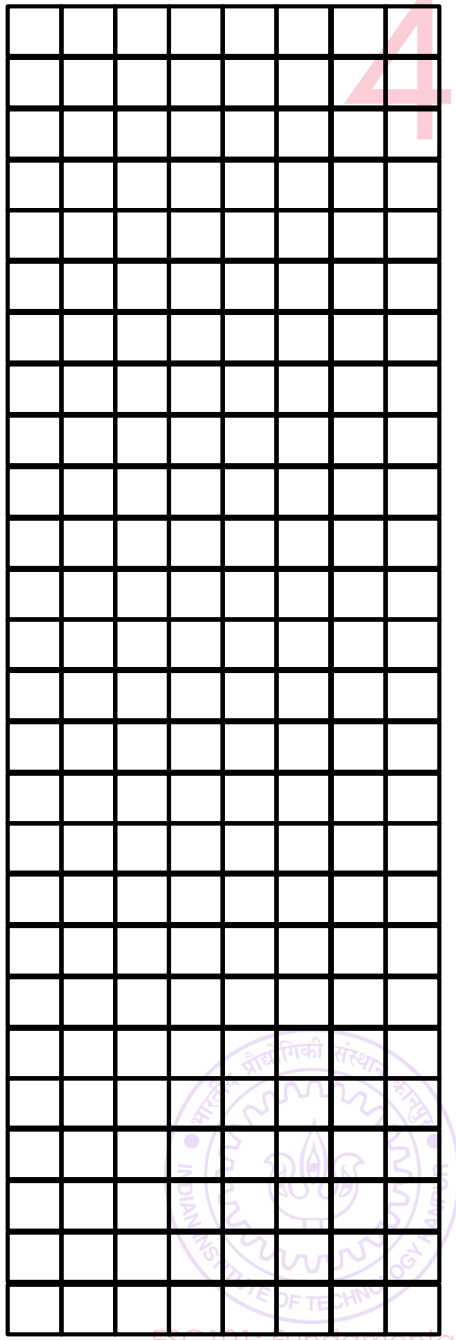
How Mr C stores variables

How Mr C stores variables

He has a very long chain of bytes

How Mr C stores variables

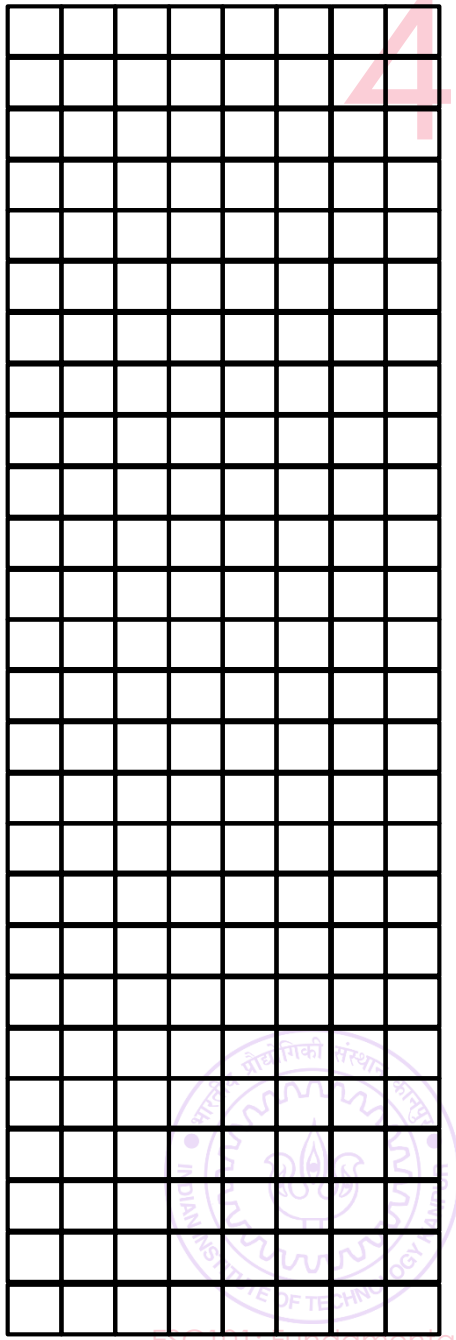
He has a very long chain of bytes



How Mr C stores variables

He has a very long chain of bytes

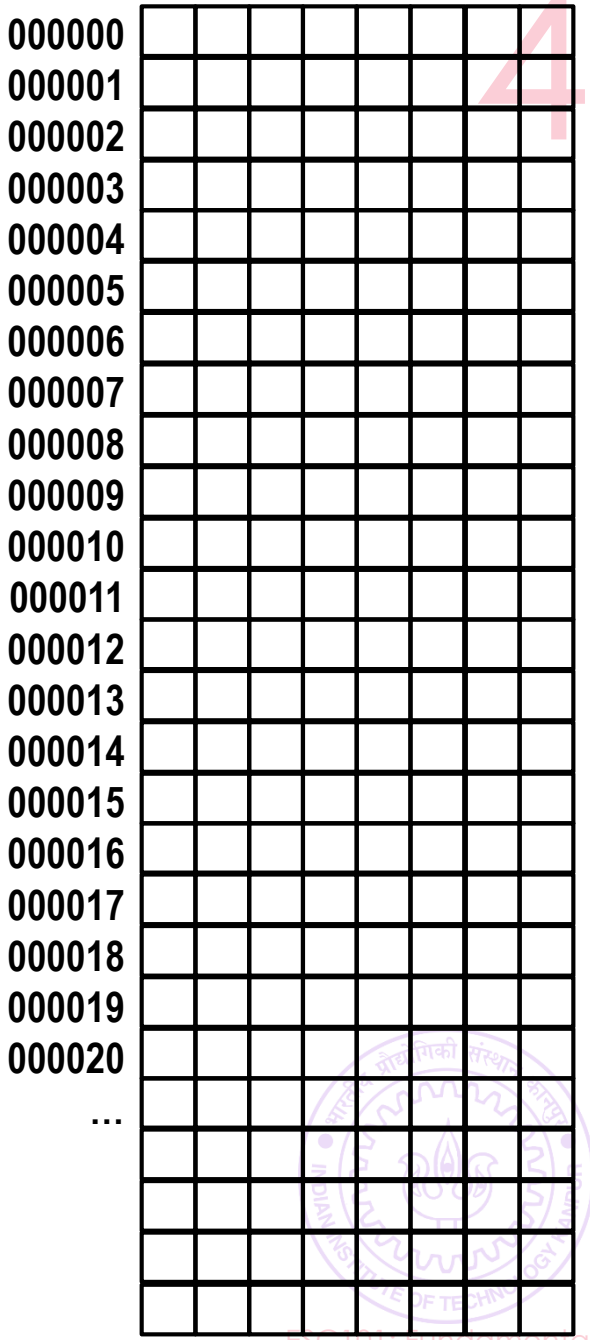
Each byte has an "address“



How Mr C stores variables

He has a very long chain of bytes

Each byte has an "address“

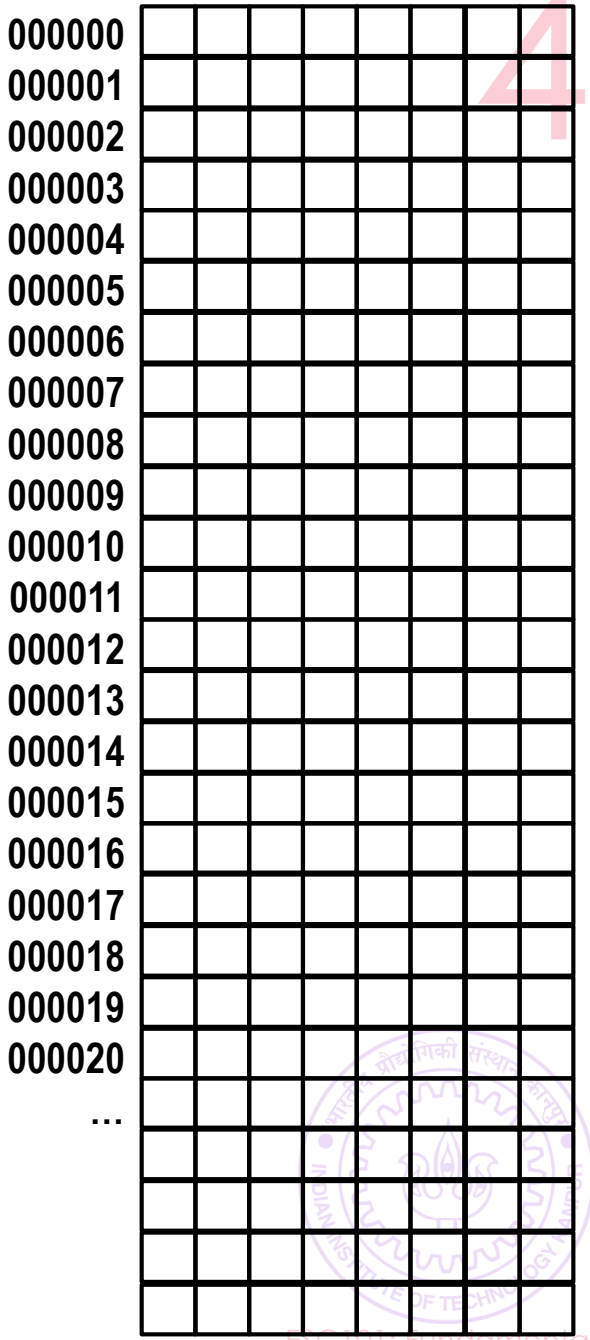


How Mr C stores variables

He has a very long chain of bytes

Each byte has an "address“

All addresses can be stored within 8 bytes



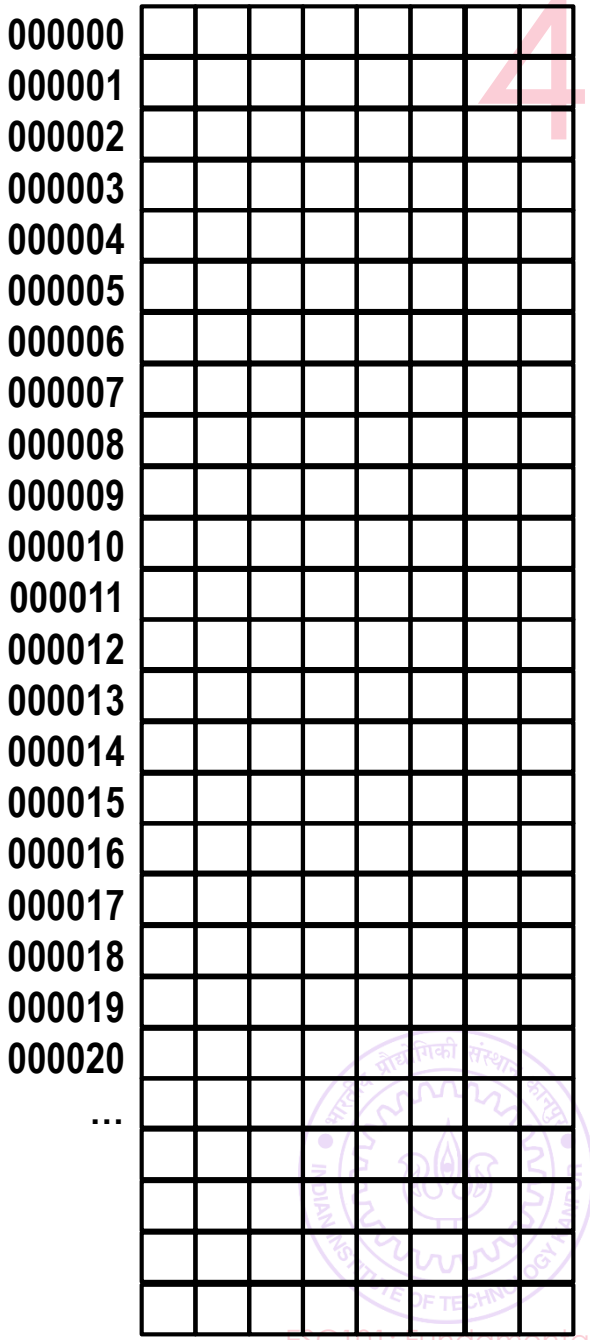
How Mr C stores variables

He has a very long chain of bytes

Each byte has an "address“

All addresses can be stored within 8 bytes

Some addresses are reserved for Mr C



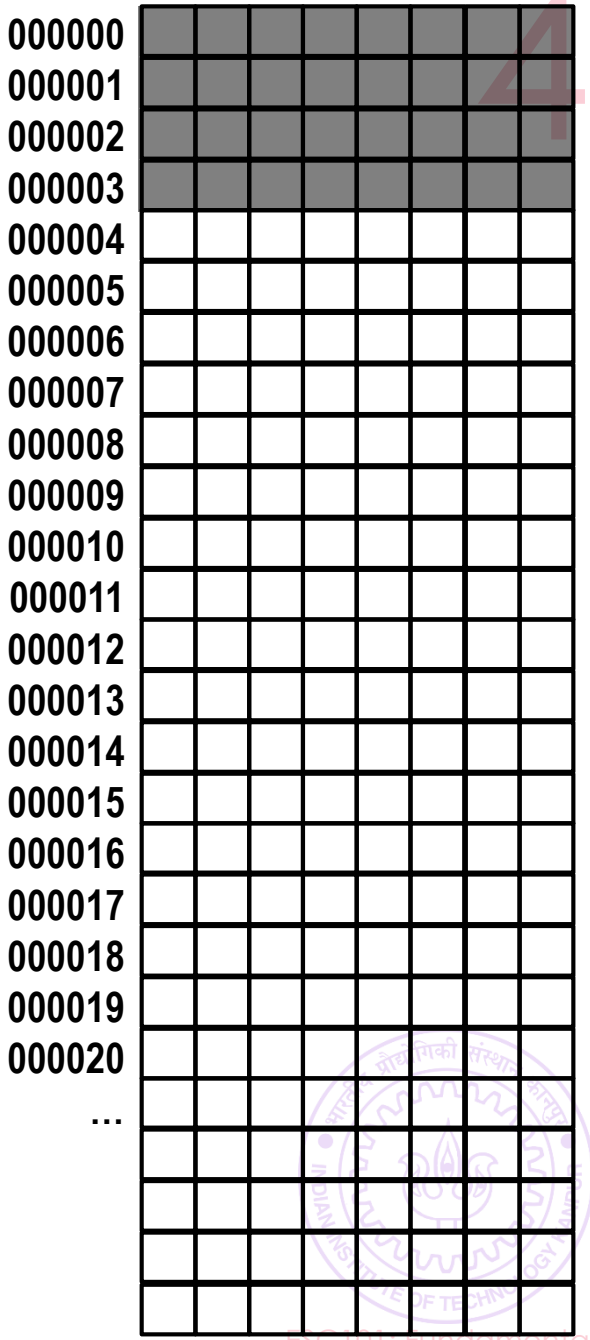
How Mr C stores variables

He has a very long chain of bytes

Each byte has an "address“

All addresses can be stored within 8 bytes

Some addresses are reserved for Mr C



How Mr C stores variables

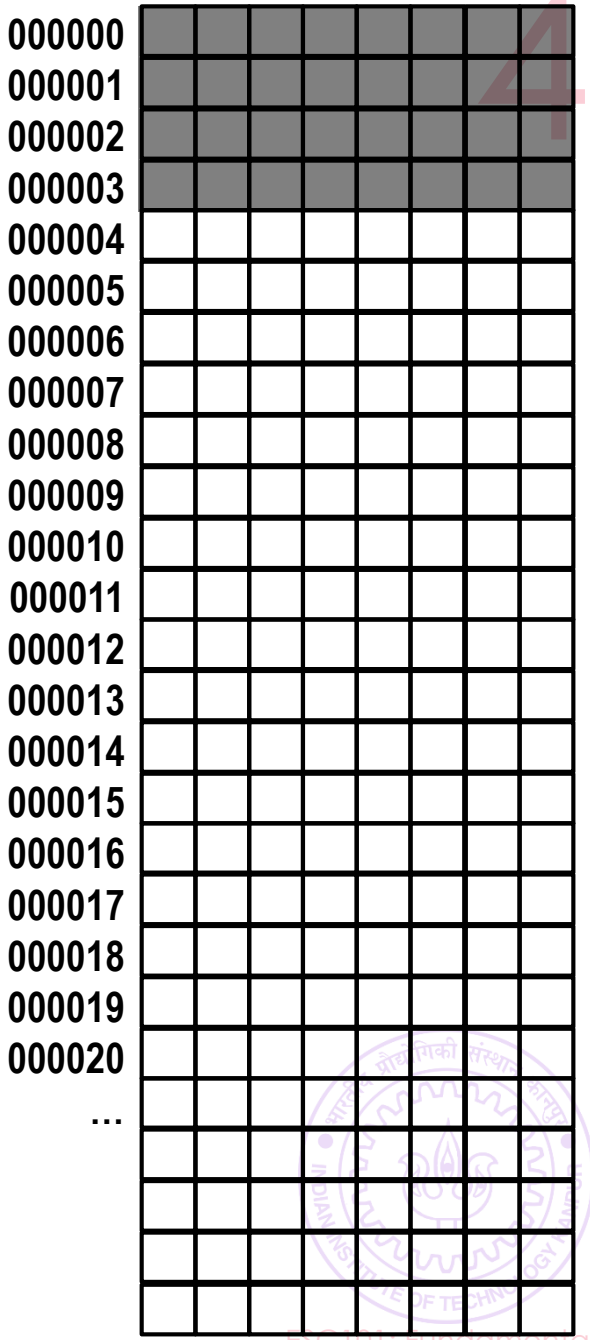
He has a very long chain of bytes

Each byte has an "address“

All addresses can be stored within 8 bytes

Some addresses are reserved for Mr C

Others can be used by you for variables



How Mr C stores variables

He has a very long chain of bytes

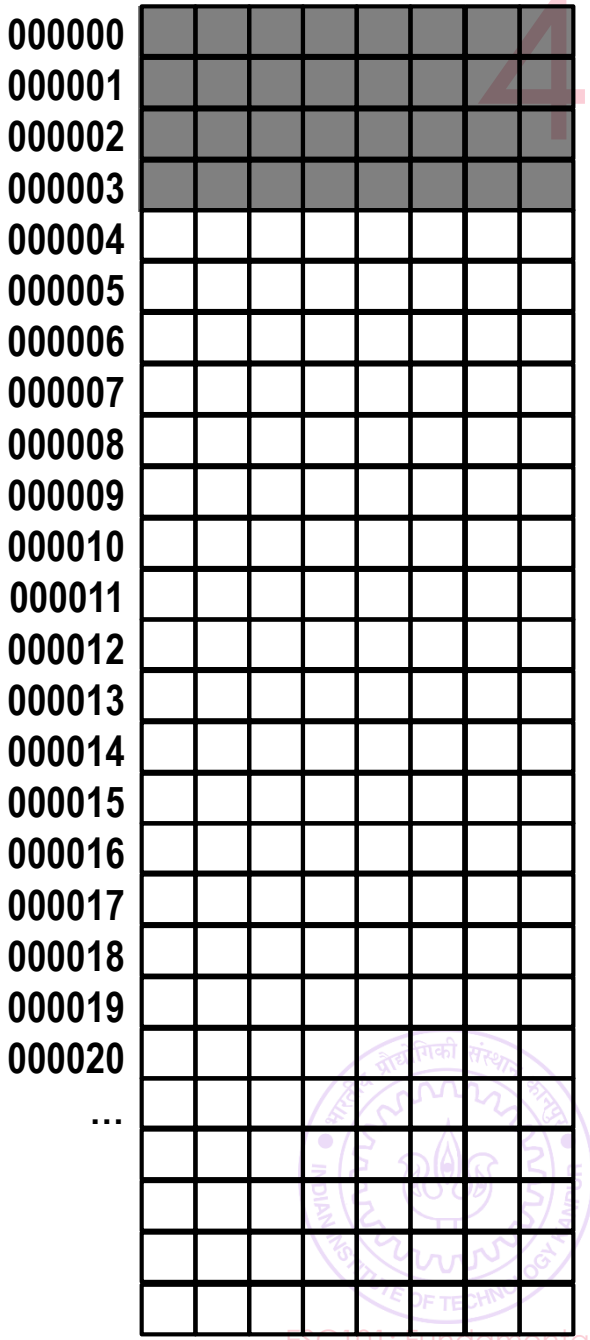
Each byte has an "address“

All addresses can be stored within 8 bytes

Some addresses are reserved for Mr C

Others can be used by you for variables

```
char c;
```



How Mr C stores variables

He has a very long chain of bytes

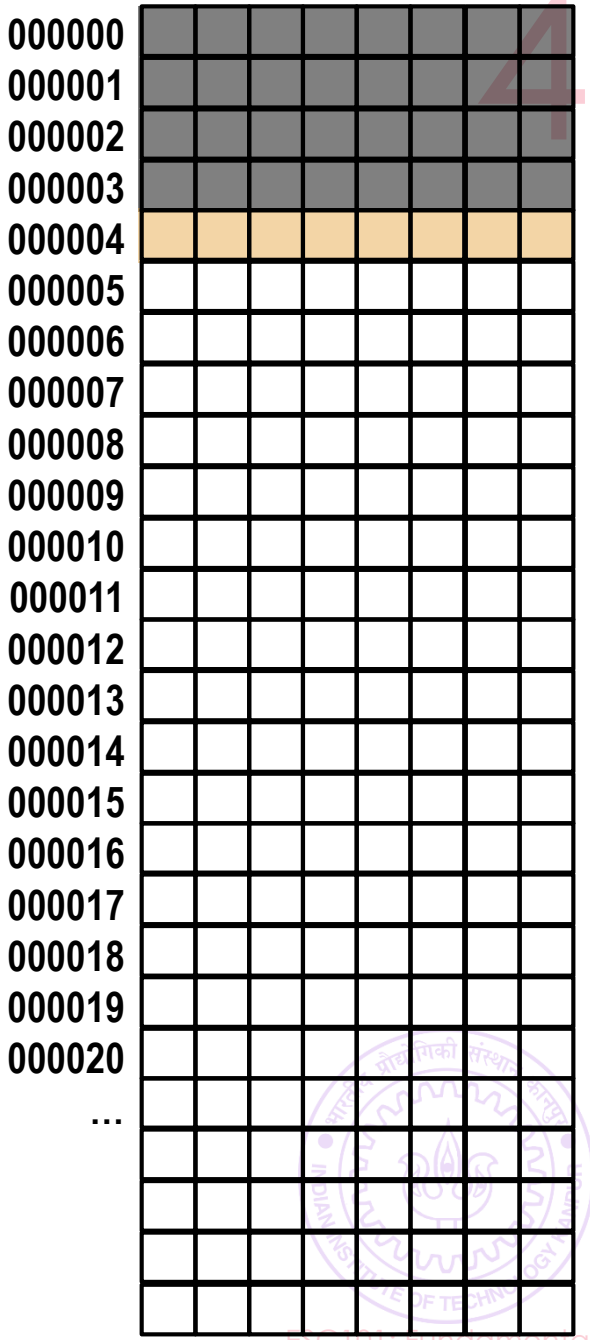
Each byte has an "address“

All addresses can be stored within 8 bytes

Some addresses are reserved for Mr C

Others can be used by you for variables

```
char c;
```



How Mr C stores variables

He has a very long chain of bytes

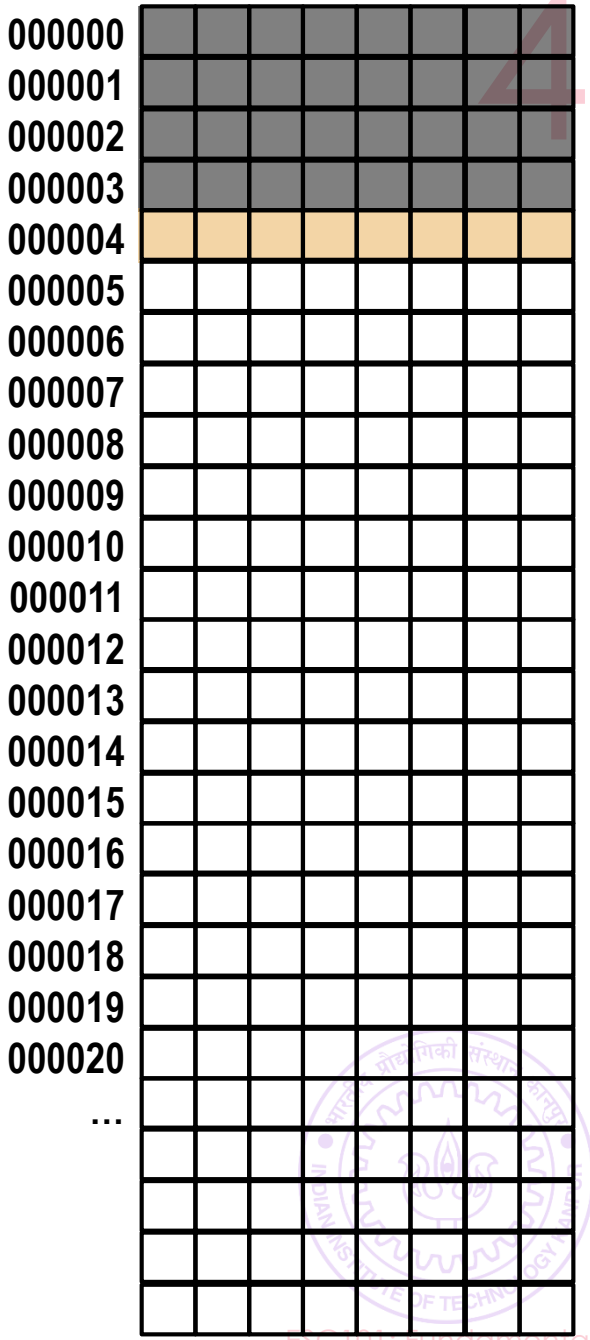
Each byte has an "address“

All addresses can be stored within 8 bytes

Some addresses are reserved for Mr C

Others can be used by you for variables

```
char c;
int a;
```



How Mr C stores variables

He has a very long chain of bytes

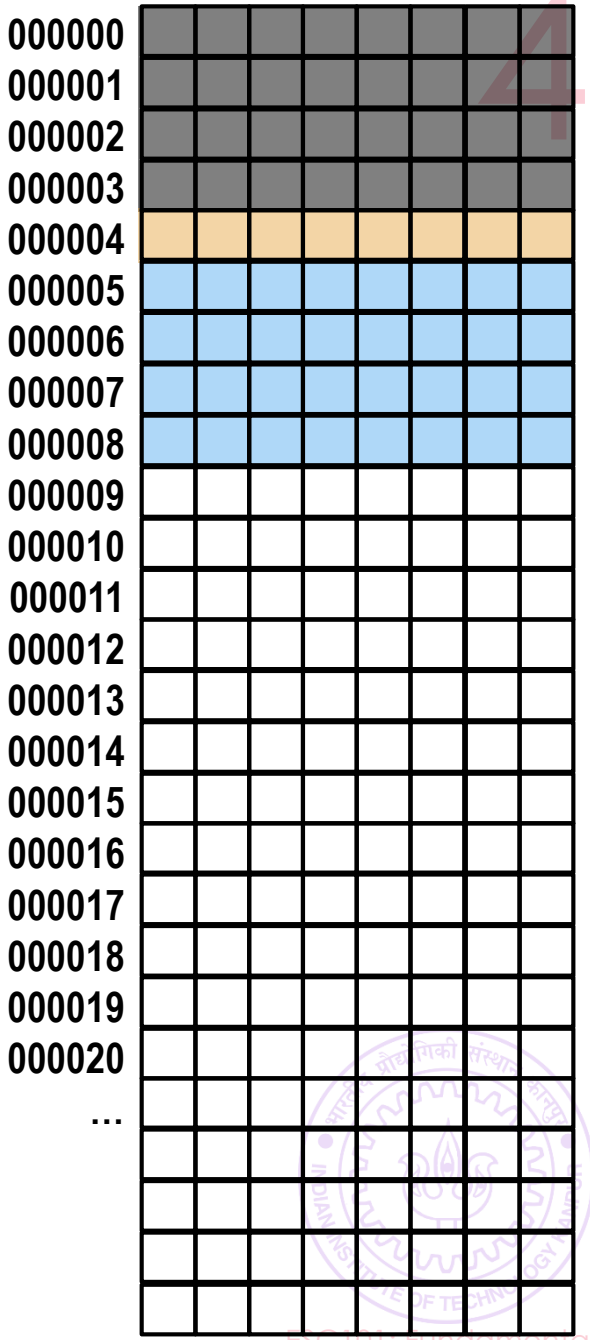
Each byte has an "address“

All addresses can be stored within 8 bytes

Some addresses are reserved for Mr C

Others can be used by you for variables

```
char c;
int a;
```



How Mr C stores variables

He has a very long chain of bytes

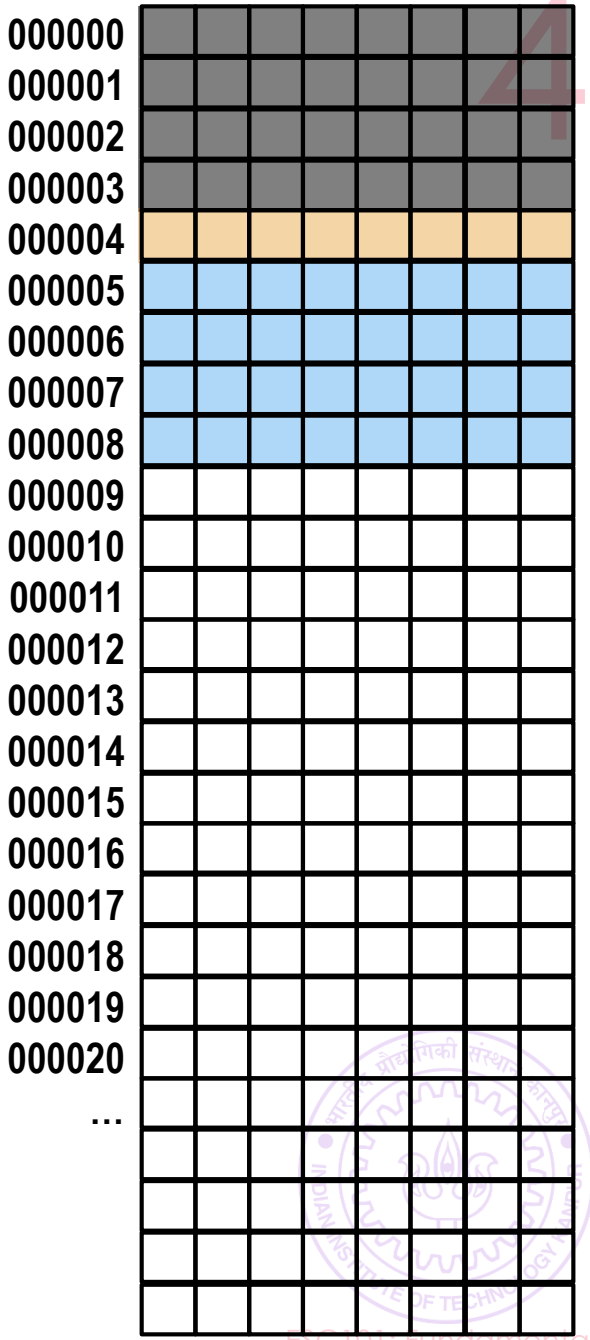
Each byte has an "address“

All addresses can be stored within 8 bytes

Some addresses are reserved for Mr C

Others can be used by you for variables

```
char c;
int a;
double d;
```



How Mr C stores variables

He has a very long chain of bytes

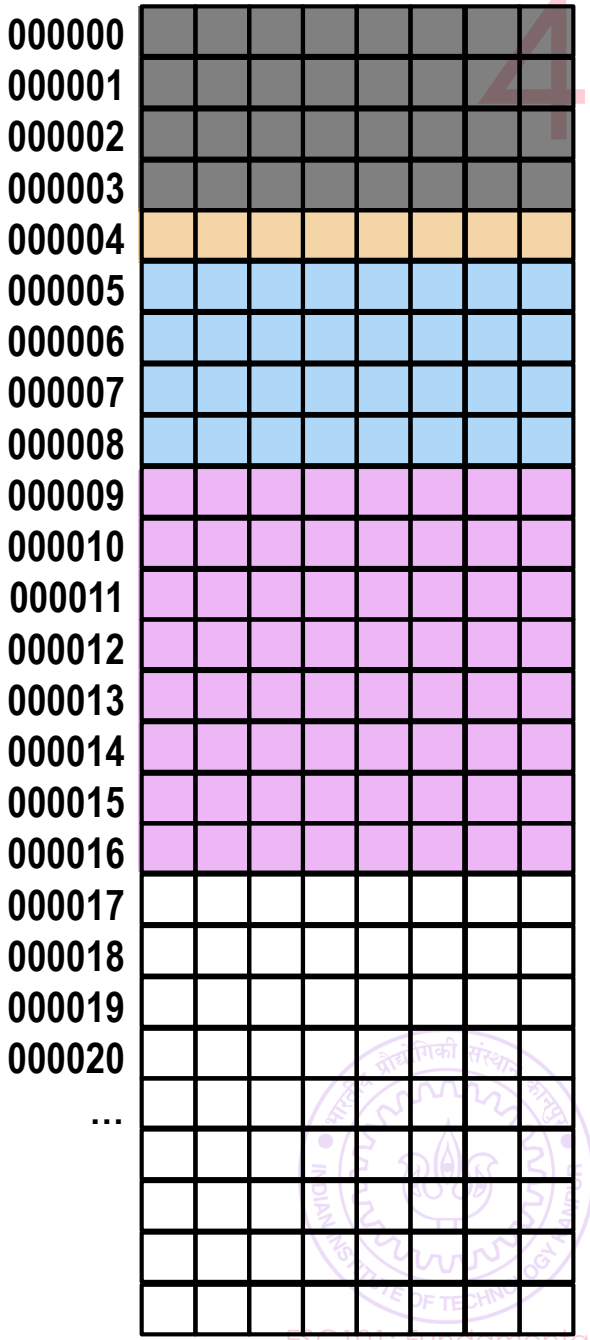
Each byte has an "address“

All addresses can be stored within 8 bytes

Some addresses are reserved for Mr C

Others can be used by you for variables

```
char c;
int a;
double d;
```



How Mr C stores variables

He has a very long chain of bytes

Each byte has an "address“

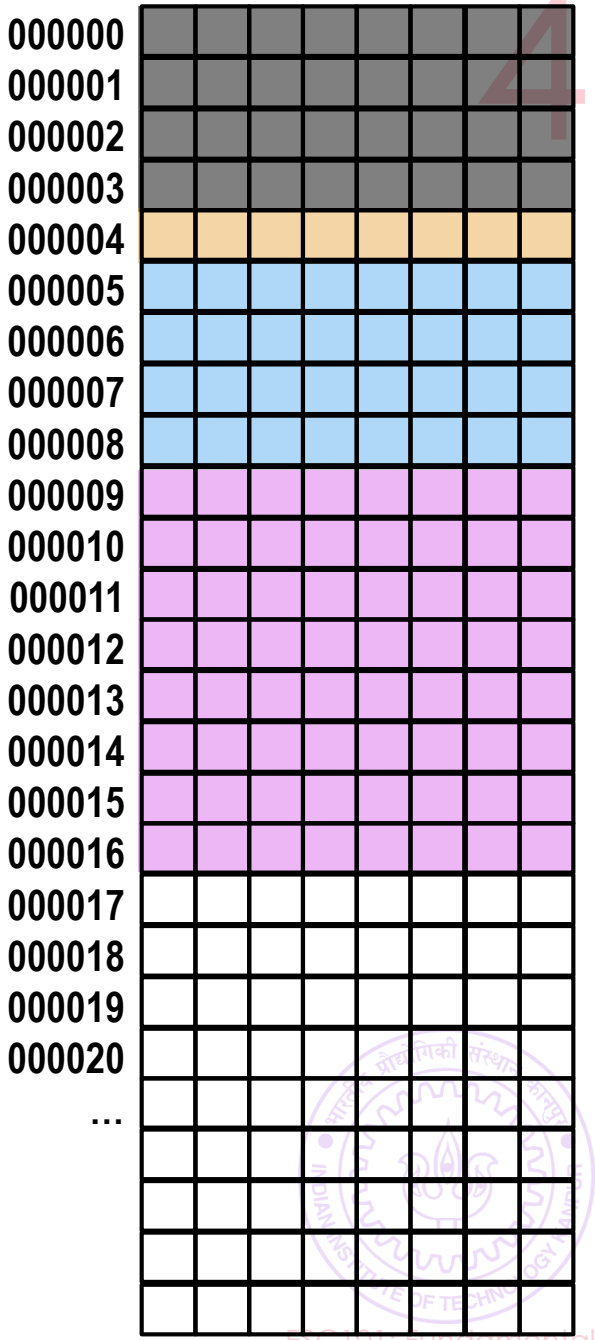
All addresses can be stored within 8 bytes

Some addresses are reserved for Mr C

Others can be used by you for variables

```
char c;
int a;
double d;
```

So c is stored at address 000004, a at 000005 and d at address 000009



Pointers

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Pointers

Don’t let anyone scare you – pointers are just a way to store these addresses

Pointers

Don't let anyone scare you – pointers are just a way to store these addresses

Each pointer is a collection of 8 bytes (same size as long) that is storing one of these internal addresses

Pointers

Don't let anyone scare you – pointers are just a way to store these addresses

Each pointer is a collection of 8 bytes (same size as long) that is storing one of these internal addresses

Be careful not to confuse these internal addresses with array indices. Array indices are what **you** use to write nice code. These addresses are used by Mr C to manage stuff

Pointers

Don't let anyone scare you – pointers are just a way to store these addresses

Each pointer is a collection of 8 bytes (same size as long) that is storing one of these internal addresses

Be careful not to confuse these internal addresses with array indices. Array indices are what **you** use to write nice code. These addresses are used by Mr C to manage stuff

In some sense Mr C manages a ridiculously huge array!

Pointers

Don't let anyone scare you – pointers are just a way to store these addresses

Each pointer is a collection of 8 bytes (same size as long) that is storing one of these internal addresses

Be careful not to confuse these internal addresses with array indices. Array indices are what **you** use to write nice code. These addresses are used by Mr C to manage stuff

In some sense Mr C manages a ridiculously huge array!

Pointers can allow us to write very beautiful code but it is a very powerful tool – misuse it and you may suffer ☺

Pointers

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Pointers

Can have pointers to a char variable, int variable, long variable, float variable, double variable

Pointers

Can have pointers to a char variable, int variable, long variable, float variable, double variable

Can have pointers to arrays of all kinds of variables

Pointers

Can have pointers to a char variable, int variable, long variable, float variable, double variable

Can have pointers to arrays of all kinds of variables

All pointers stored internally as 8 byte non-negative integers

Pointers

Can have pointers to a char variable, int variable, long variable, float variable, double variable

Can have pointers to arrays of all kinds of variables

All pointers stored internally as 8 byte non-negative integers

NULL pointer – one that stores address 00000000

Pointers

Can have pointers to a char variable, int variable, long variable, float variable, double variable

Can have pointers to arrays of all kinds of variables

All pointers stored internally as 8 byte non-negative integers

NULL pointer – one that stores address 00000000

- Named constant NULL can be used to check if a pointer is NULL

Pointers

Can have pointers to a char variable, int variable, long variable, float variable, double variable

Can have pointers to arrays of all kinds of variables

All pointers stored internally as 8 byte non-negative integers

NULL pointer – one that stores address 00000000

- Named constant NULL can be used to check if a pointer is NULL
- Do not confuse with NULL character '\0' – that has a valid ASCII value 0

Pointers

Can have pointers to a char variable, int variable, long variable, float variable, double variable

Can have pointers to arrays of all kinds of variables

All pointers stored internally as 8 byte non-negative integers

NULL pointer – one that stores address 00000000

- Named constant NULL can be used to check if a pointer is NULL
- Do not confuse with NULL character '\0' – that has a valid ASCII value 0
- NULL character **is actually used** to indicate that string is over

Pointers

Can have pointers to a char variable, int variable, long variable, float variable, double variable

Can have pointers to arrays of all kinds of variables

All pointers stored internally as 8 byte non-negative integers

NULL pointer – one that stores address 00000000

- Named constant NULL can be used to check if a pointer is NULL
- Do not confuse with NULL character '\0' – that has a valid ASCII value 0
- NULL character **is actually used** to indicate that string is over
- WARNING: NULL pointers may be returned by some string.h functions e.g. strstr

Pointers

Can have pointers to a char variable, int variable, long variable, float variable, double variable

Can have pointers to arrays of all kinds of variables

All pointers stored internally as 8 byte non-negative integers

NULL pointer – one that stores address 00000000

- Named constant NULL can be used to check if a pointer is NULL
- Do not confuse with NULL character '\0' – that has a valid ASCII value 0
- NULL character **is actually used** to indicate that string is over
- WARNING: NULL pointers may be returned by some string.h functions e.g. strstr

Do not try to read from/write to address 00000000

Pointers

Can have pointers to a char variable, int variable, long variable, float variable, double variable

Can have pointers to arrays of all kinds of variables

All pointers stored internally as 8 byte non-negative integers

NULL pointer – one that stores address 00000000

- Named constant NULL can be used to check if a pointer is NULL
- Do not confuse with NULL character '\0' – that has a valid ASCII value 0
- NULL character **is actually used** to indicate that string is over
- WARNING: NULL pointers may be returned by some string.h functions e.g. strstr

Do not try to read from/write to address 00000000

- Reserved by Mr C or else the operating system

Pointers

Can have pointers to a char variable, int variable, long variable, float variable, double variable

Can have pointers to arrays of all kinds of variables

All pointers stored internally as 8 byte non-negative integers

NULL pointer – one that stores address 00000000

- Named constant NULL can be used to check if a pointer is NULL
- Do not confuse with NULL character '\0' – that has a valid ASCII value 0
- NULL character **is actually used** to indicate that string is over
- WARNING: NULL pointers may be returned by some string.h functions e.g. strstr

Do not try to read from/write to address 00000000

- Reserved by Mr C or else the operating system
- Doing so will cause a segfault and crash your program/even your computer

My first pointer

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

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

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
```

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
```

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
```

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
```

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
```

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
```

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

a is an int variable, value 42

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

a is an int variable, value 42

ptr is a pointer that will store address to an int variable

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

a is an int variable, value 42

ptr is a pointer that will store address to an int variable

Please store address of a in ptr

My first pointer

HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

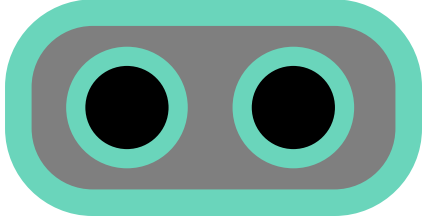
a is an int variable, value 42

ptr is a pointer that will store address to an int variable

Please store address of a in ptr

Please print the value of the int stored at the address in ptr

My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

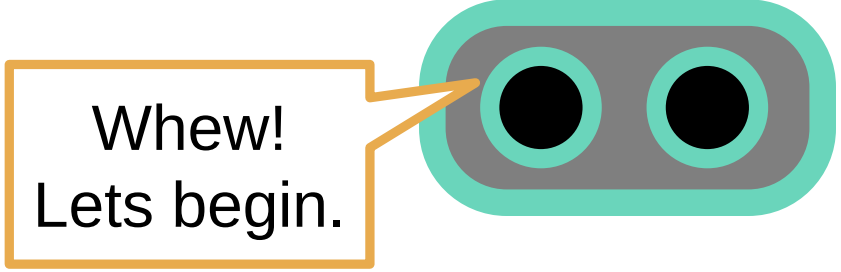
a is an int variable, value 42

ptr is a pointer that will store address to an int variable

Please store address of a in ptr

Please print the value of the int stored at the address in ptr

My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

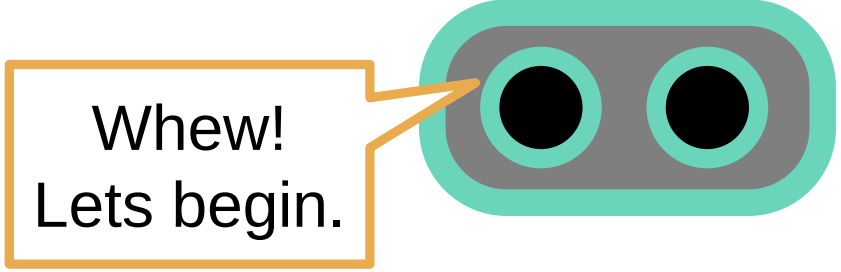
a is an int variable, value 42

ptr is a pointer that will store address to an int variable

Please store address of a in ptr

Please print the value of the int stored at the address in ptr

My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```


HOW WE USUALLY SPEAK TO A HUMAN

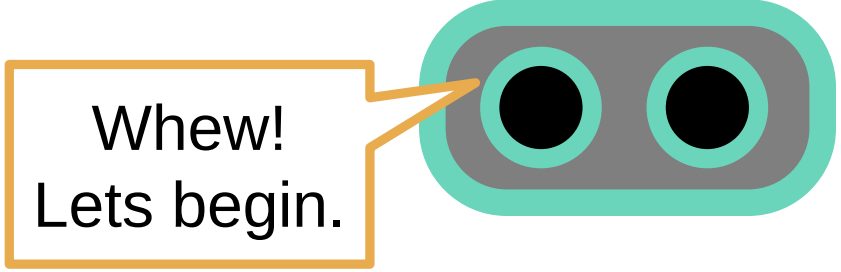
a is an int variable, value 42

ptr is a pointer that will store address to an int variable

Please store address of a in ptr

Please print the value of the int stored at the address in ptr

My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```

HOW WE USUALLY SPEAK TO A HUMAN

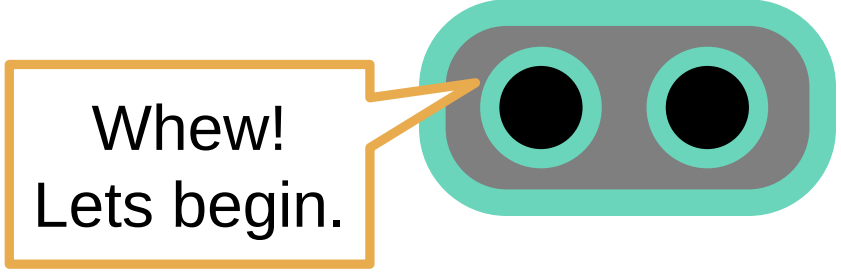
a is an int variable, value 42

ptr is a pointer that will store address to an int variable

Please store address of a in ptr

Please print the value of the int stored at the address in ptr

My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```

42

000023

a

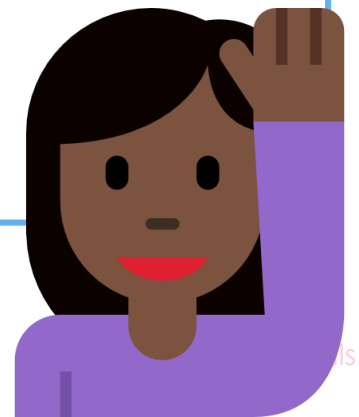
HOW WE USUALLY SPEAK TO A HUMAN

a is an int variable, value 42

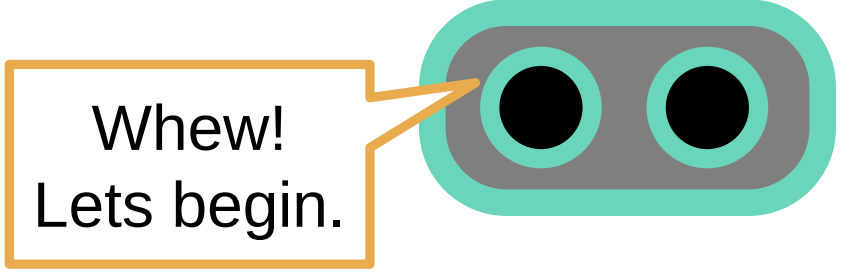
ptr is a pointer that will store address to an int variable

Please store address of a in ptr

Please print the value of the int stored at the address in ptr



My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```

42

000023

a

HOW WE USUALLY SPEAK TO A HUMAN

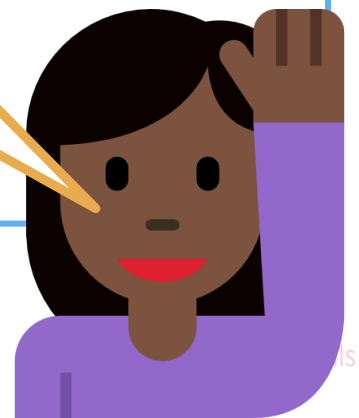
a is an int variable, value 42

ptr is a pointer that will store address to an int variable

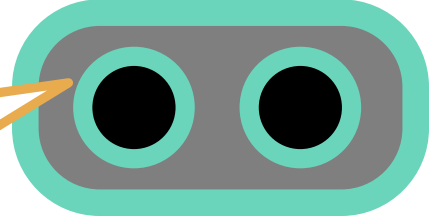
Please store address of a in ptr

Please print the value of the int stored at the address in ptr

a is stored at internal location 000023

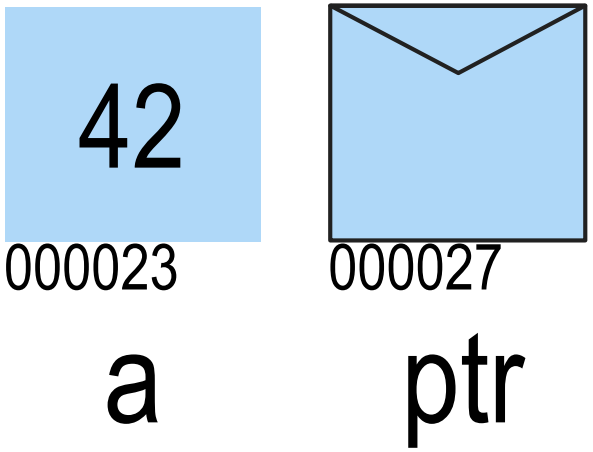


My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```



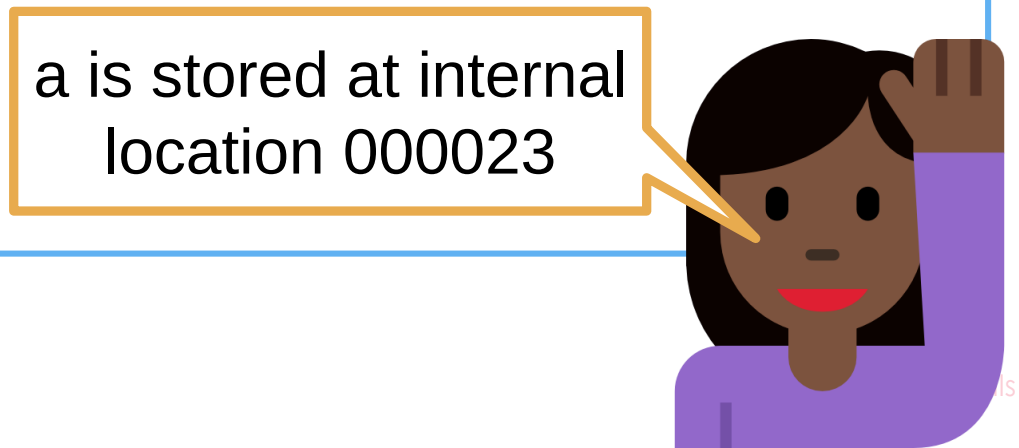
HOW WE USUALLY SPEAK TO A HUMAN

a is an int variable, value 42

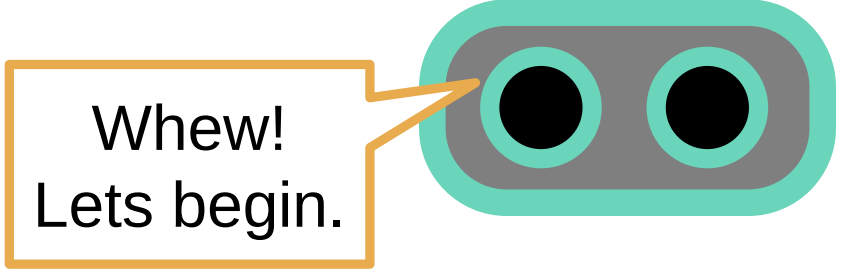
ptr is a pointer that will store address to an int variable

Please store address of a in ptr

Please print the value of the int stored at the address in ptr

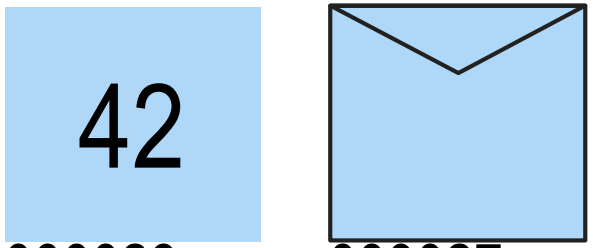


My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```



HOW WE USUALLY SPEAK TO A HUMAN

a is an int variable, value 42

ptr is a pointer that will store address to an int variable

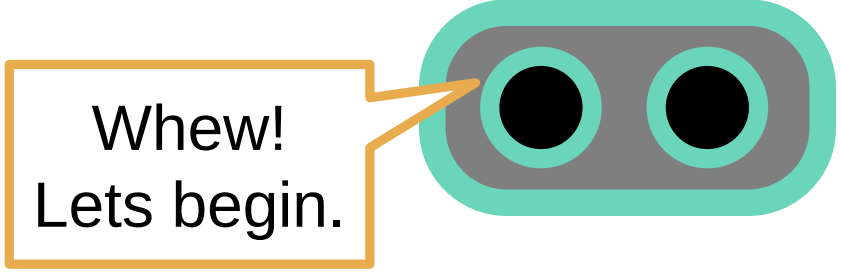
Please store address of a in ptr

Please print the value of the int stored at the address in ptr

a is stored at internal location 000023

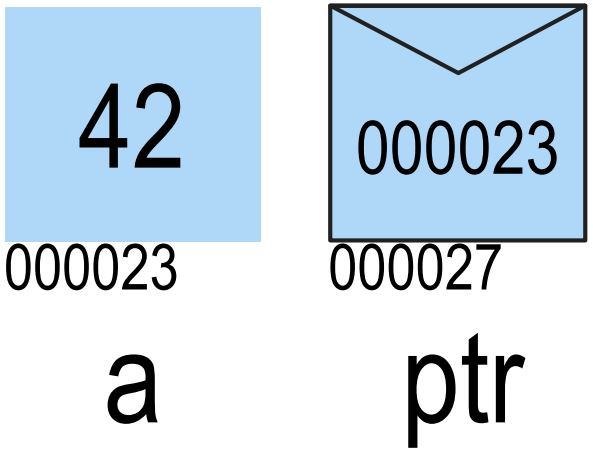
int takes 4 bytes to store

My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```



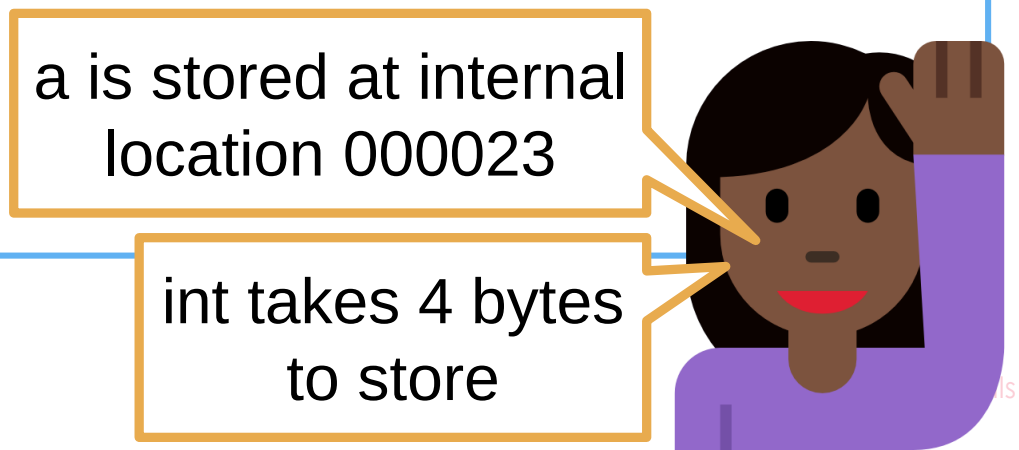
HOW WE USUALLY SPEAK TO A HUMAN

a is an int variable, value 42

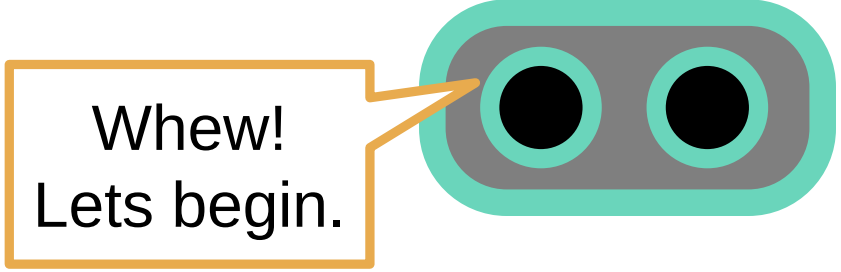
ptr is a pointer that will store address to an int variable

Please store address of a in ptr

Please print the value of the int stored at the address in ptr

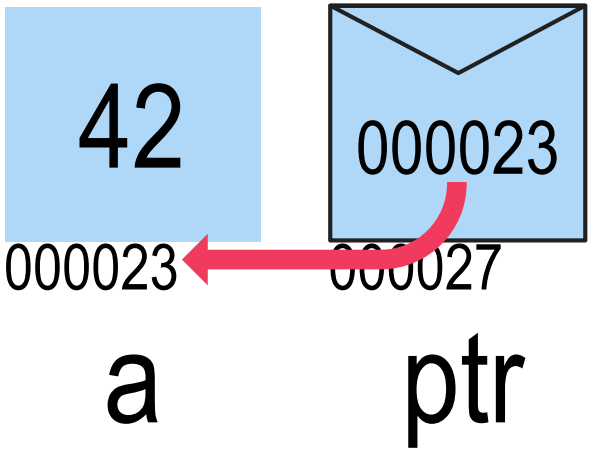


My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```



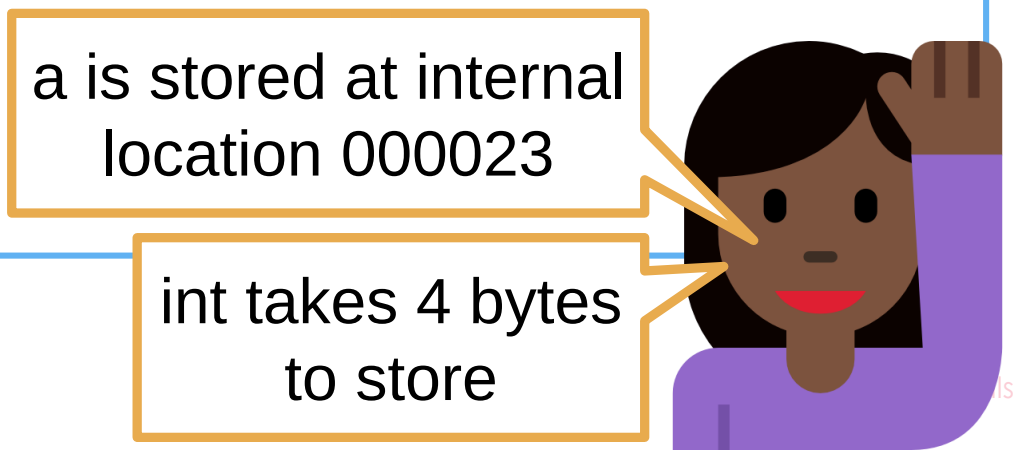
HOW WE USUALLY SPEAK TO A HUMAN

a is an int variable, value 42

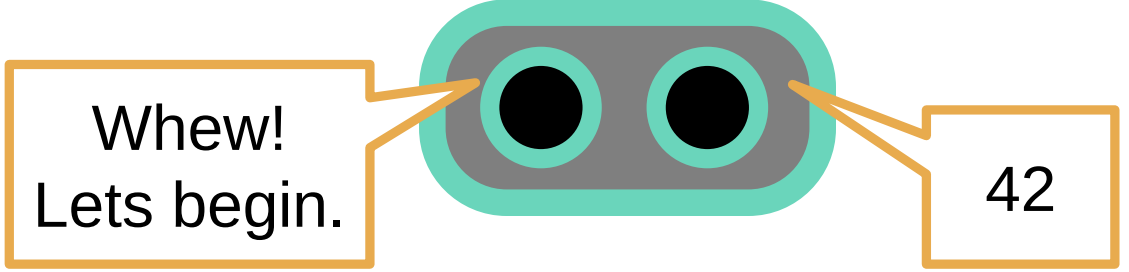
ptr is a pointer that will store address to an int variable

Please store address of a in ptr

Please print the value of the int stored at the address in ptr

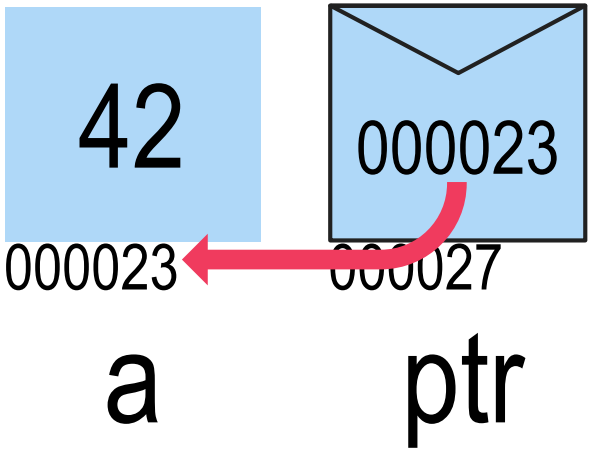


My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```



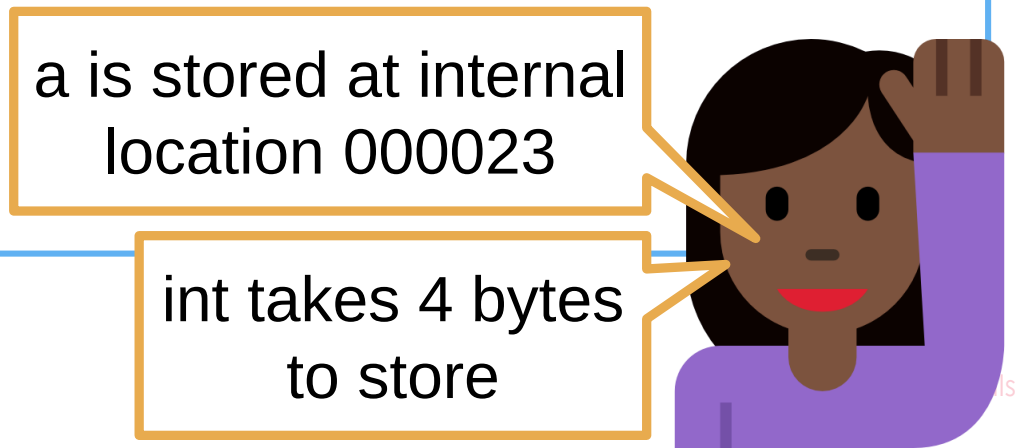
HOW WE USUALLY SPEAK TO A HUMAN

a is an int variable, value 42

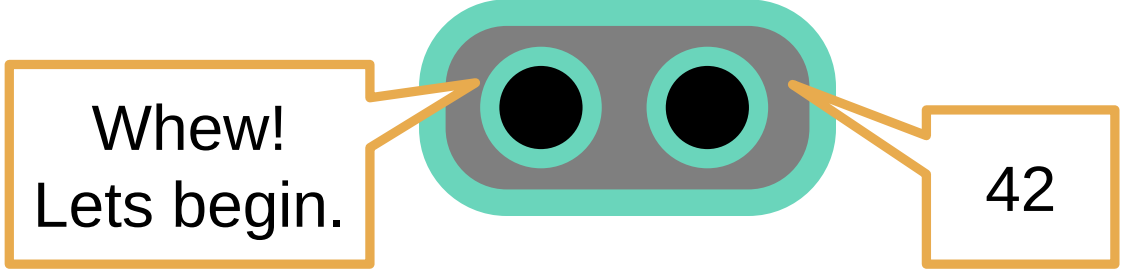
ptr is a pointer that will store address to an int variable

Please store address of a in ptr

Please print the value of the int stored at the address in ptr

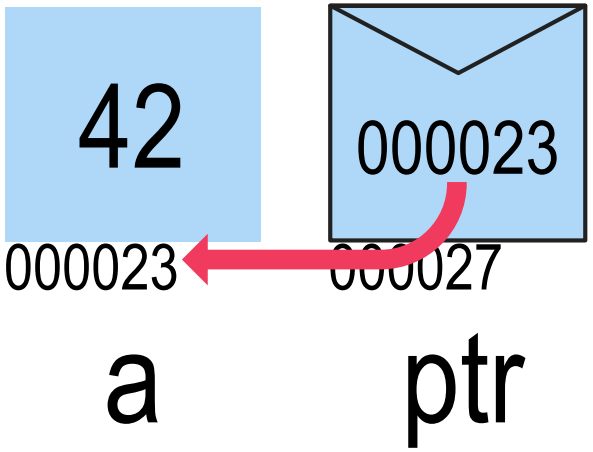


My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```



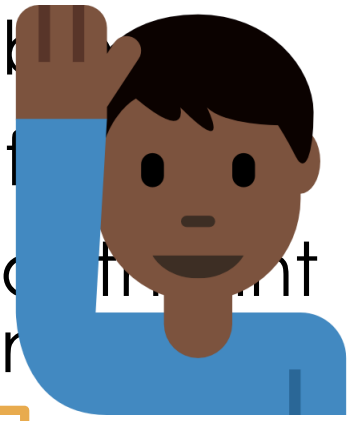
HOW WE USUALLY SPEAK TO A HUMAN

a is an int variable, value 42

ptr is a pointer that will store address to an int variab

Please store address of

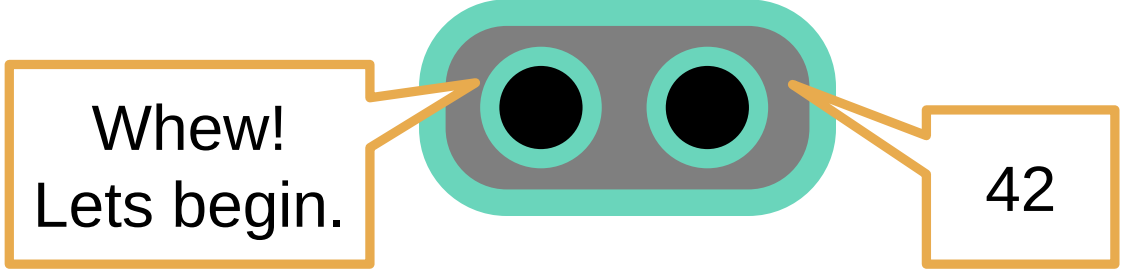
Please print the value of th int stored at the address in



a is stored at internal location 000023

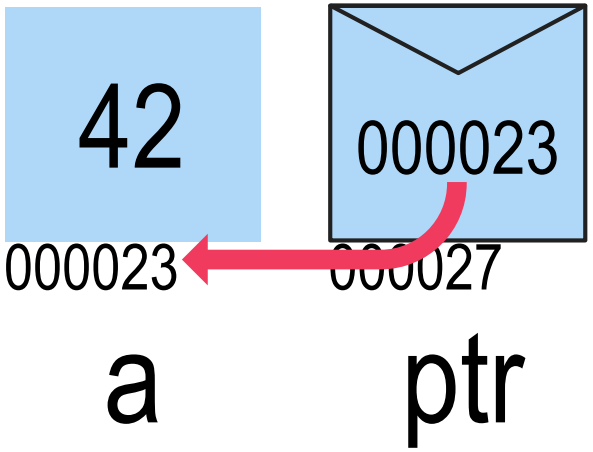
int takes 4 bytes to store

My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```



HOW WE USUALLY SPEAK TO A HUMAN

a is an int variable, value 42

ptr is a pointer that will store

Please print the value

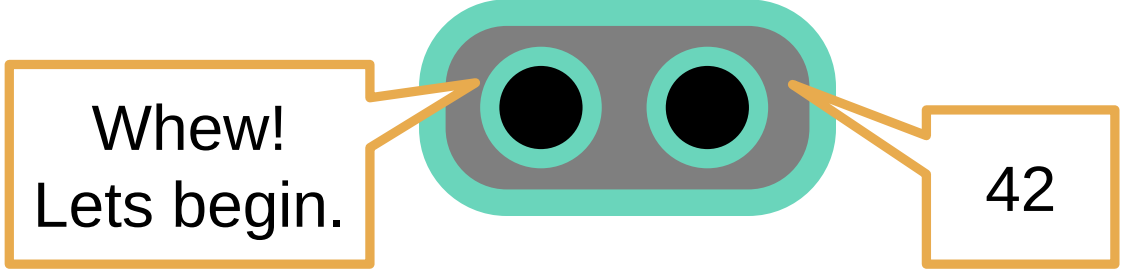
stored at the address in

Can also have pointes to char, long, float, double

a is stored at internal location 000023

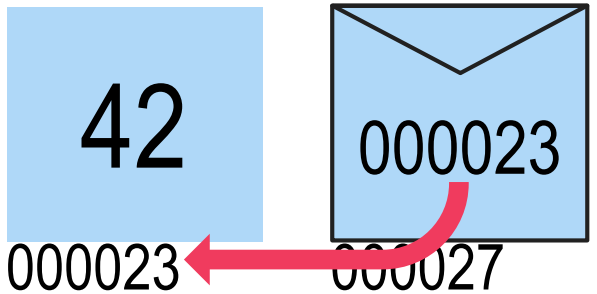
int takes 4 bytes to store

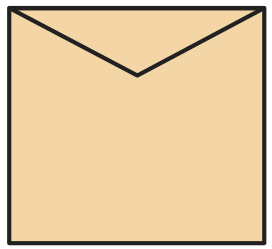
My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```





HOW WE USUALLY SPEAK TO A HUMAN

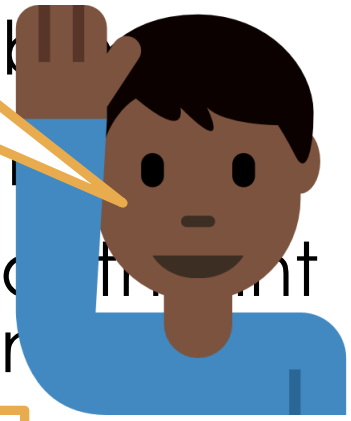
a is an int variable, value 42

ptr is a pointer that will store

Please print the value

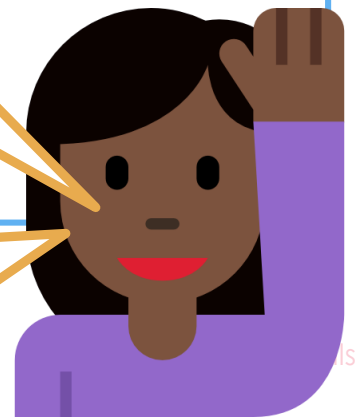
stored at the address in

Can also have pointes to char, long, float, double

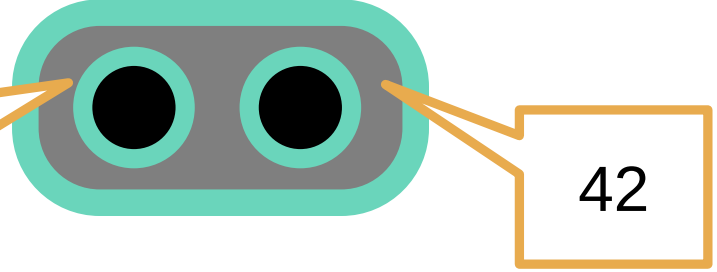


a is stored at internal location 000023

int takes 4 bytes to store

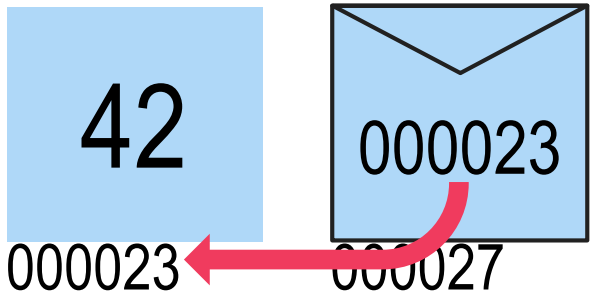


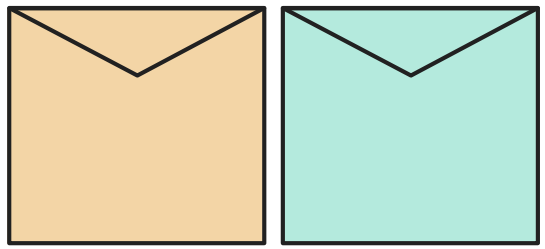
My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```





HOW WE USUALLY SPEAK TO A HUMAN

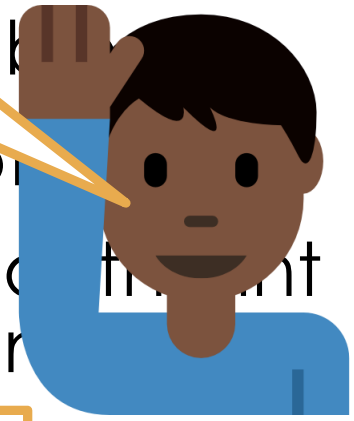
a is an int variable, value 42

ptr is a pointer that will store

Please print the value

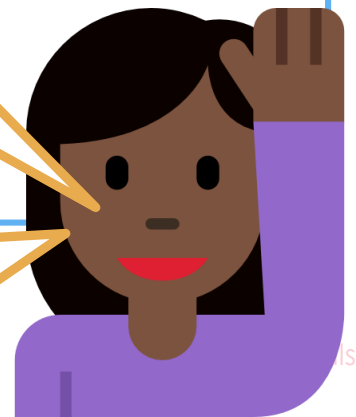
stored at the address in

Can also have pointes to char, long, float, double

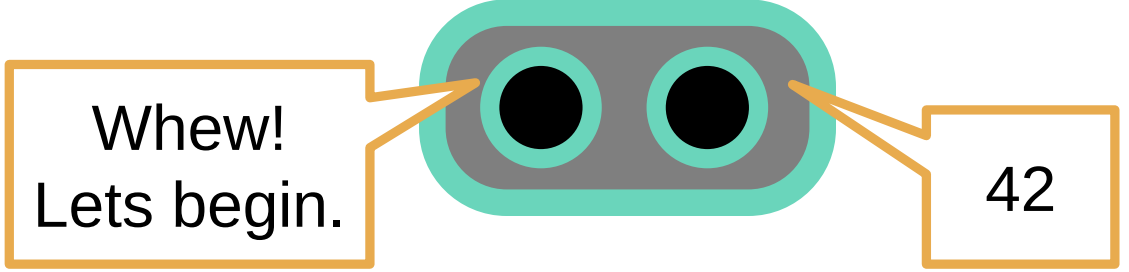


a is stored at internal location 000023

int takes 4 bytes to store

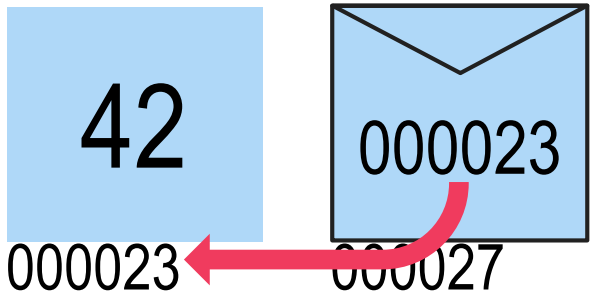


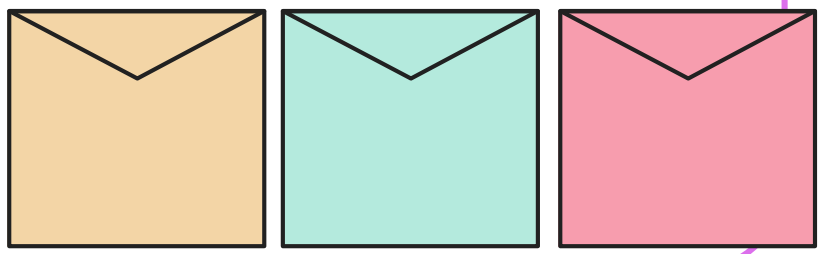
My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```





HOW WE USUALLY SPEAK TO A HUMAN

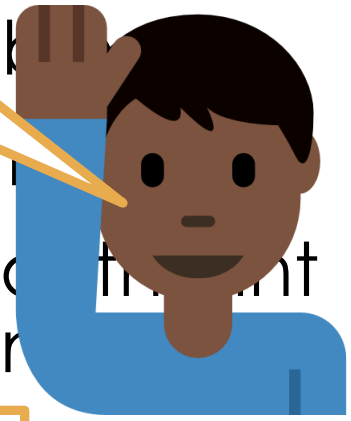
a is an int variable, value 42

ptr is a pointer that will store

Please print the value

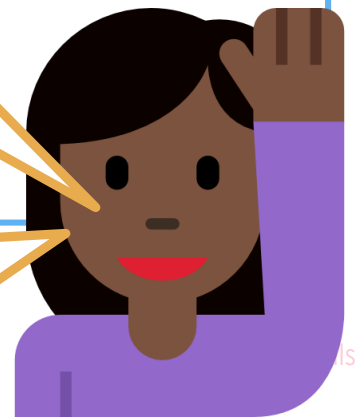
stored at the address in

Can also have pointes to char, long, float, double

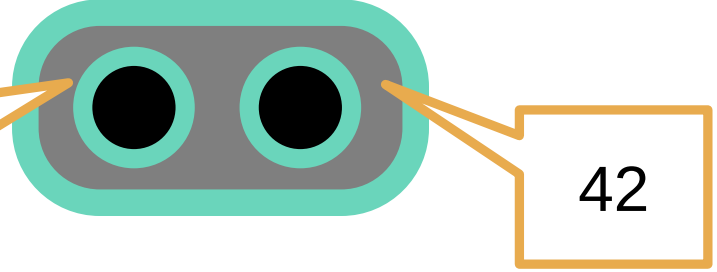


a is stored at internal location 000023

int takes 4 bytes to store

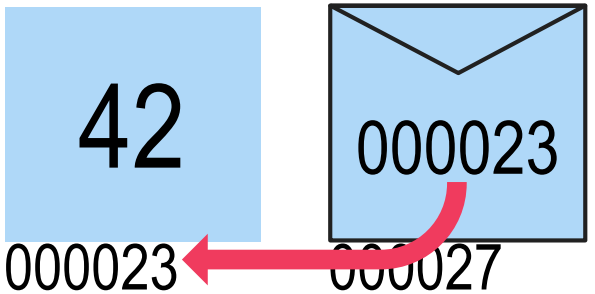


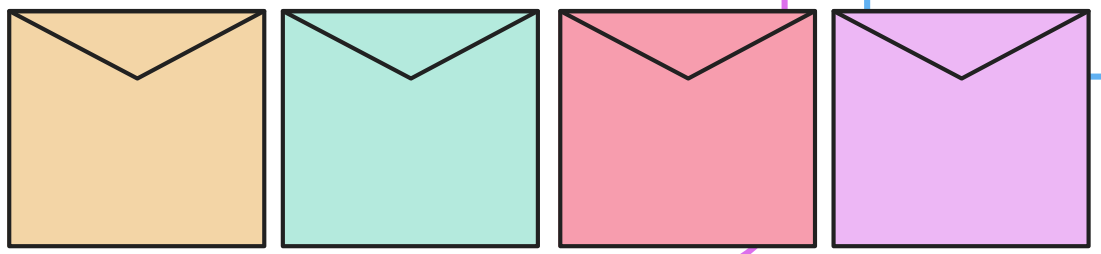
My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```





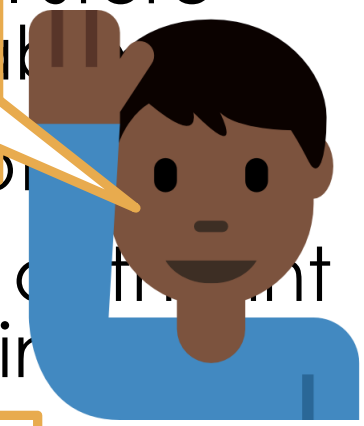
HOW WE USUALLY SPEAK TO A HUMAN

a is an int variable, value 42

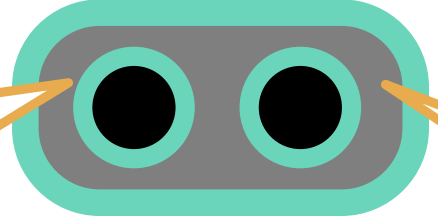
ptr is a pointer that will store

Please print the value

stored at the address in

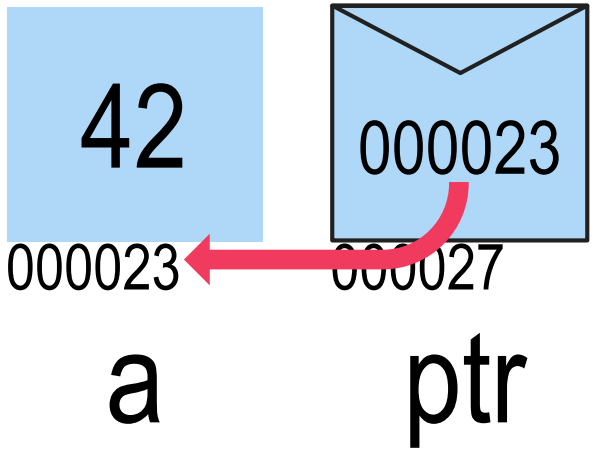


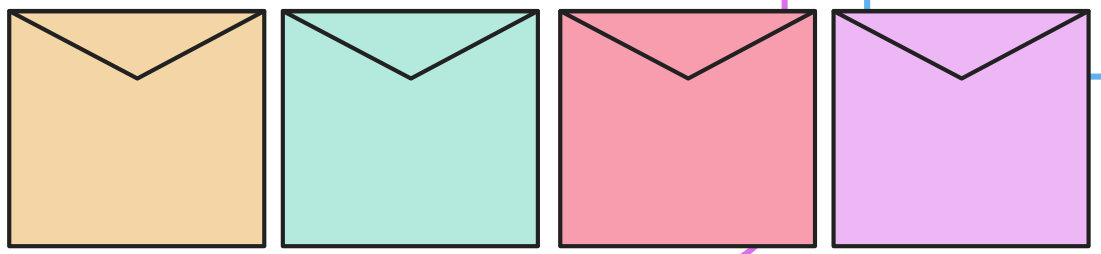
My first pointer



HOW WE MUST SPEAK TO MR. COMPILER

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```





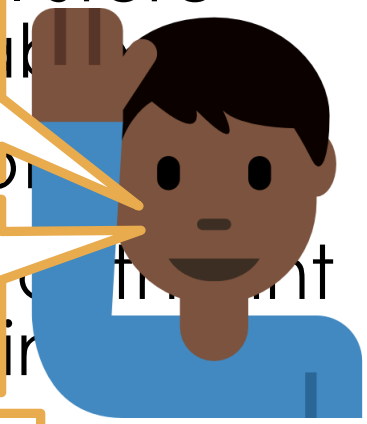
HOW WE USUALLY SPEAK TO A HUMAN

a is an int variable, value 42

ptr is a pointer that will store

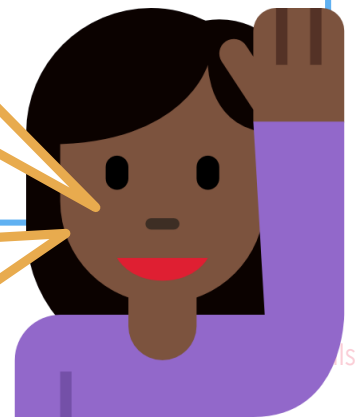
Can also have pointes to char, long, float, double

All these envelope-like boxes take 8 bytes



a is stored at internal location 000023

int takes 4 bytes to store



Pointers with printf and scanf

Pointers with printf and scanf

Pointers contain addresses, so to print the address itself, use the %ld format since addresses are 8 byte long

Pointers with printf and scanf

Pointers contain addresses, so to print the address itself, use the %ld format since addresses are 8 byte long

To print value at an address given by a pointer, first dereference the pointer using * operator

Pointers with printf and scanf

Pointers contain addresses, so to print the address itself, use the %ld format since addresses are 8 byte long

To print value at an address given by a pointer, first dereference the pointer using * operator

```
printf("%d", *ptr);
```

Pointers with printf and scanf

Pointers contain addresses, so to print the address itself, use the %ld format since addresses are 8 byte long

To print value at an address given by a pointer, first dereference the pointer using * operator

```
printf("%d", *ptr);
```

Scanf requires the address of the variable where input is to be stored. Can pass it the referenced address

Pointers with printf and scanf

Pointers contain addresses, so to print the address itself, use the %ld format since addresses are 8 byte long

To print value at an address given by a pointer, first dereference the pointer using * operator

```
printf("%d", *ptr);
```

Scanf requires the address of the variable where input is to be stored. Can pass it the referenced address

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

Pointers with printf and scanf

Pointers contain addresses, so to print the address itself, use the %ld format since addresses are 8 byte long

To print value at an address given by a pointer, first dereference the pointer using * operator

```
printf("%d", *ptr);
```

Scanf requires the address of the variable where input is to be stored. Can pass it the referenced address

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

or else pass it a pointer

Pointers with printf and scanf

Pointers contain addresses, so to print the address itself, use the %ld format since addresses are 8 byte long

To print value at an address given by a pointer, first dereference the pointer using * operator

```
printf("%d", *ptr);
```

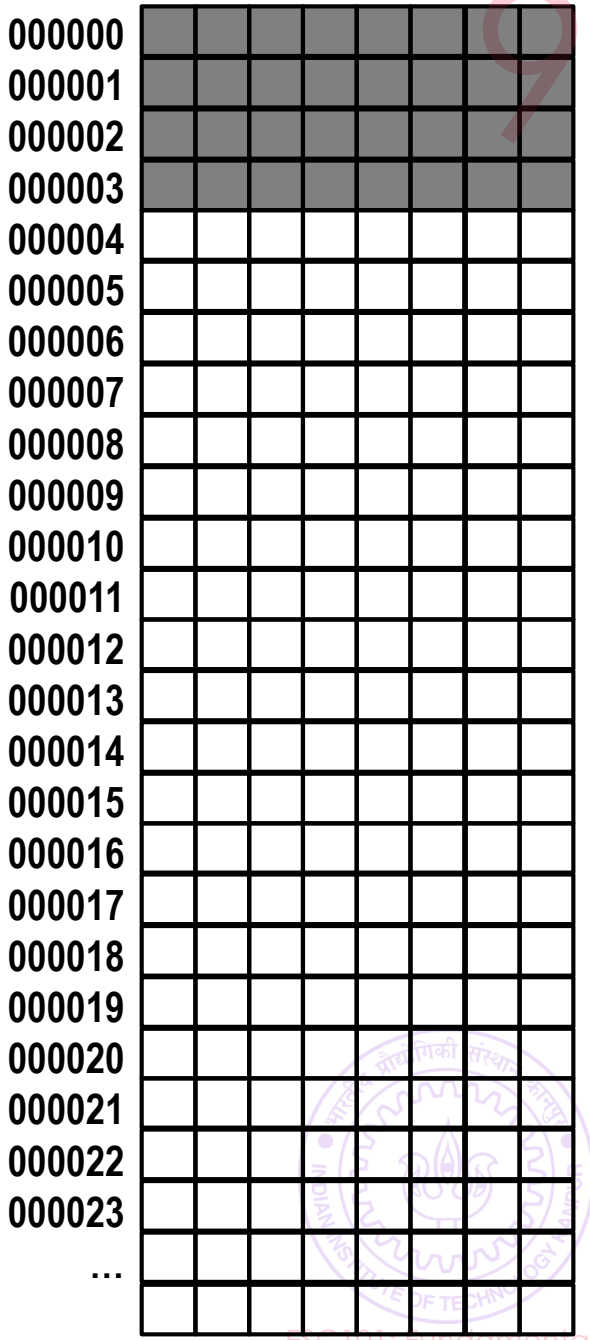
Scanf requires the address of the variable where input is to be stored. Can pass it the referenced address

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

or else pass it a pointer

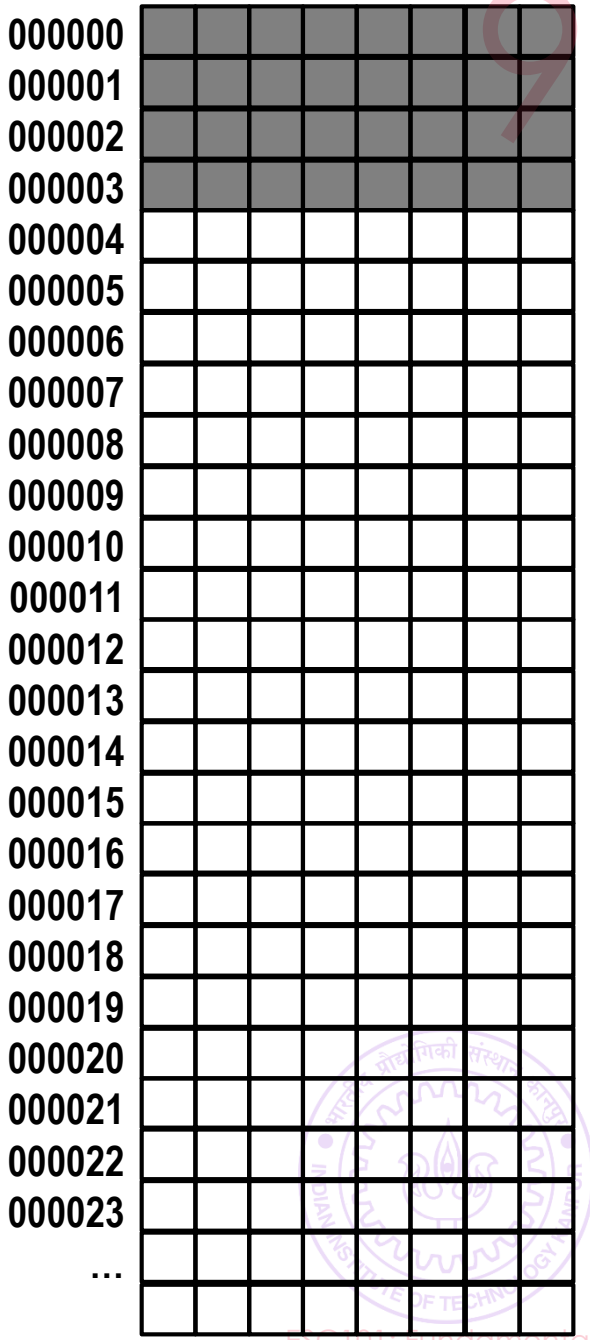
```
scanf("%d", ptr);
```

How Mr C stores arrays



How Mr C stores arrays

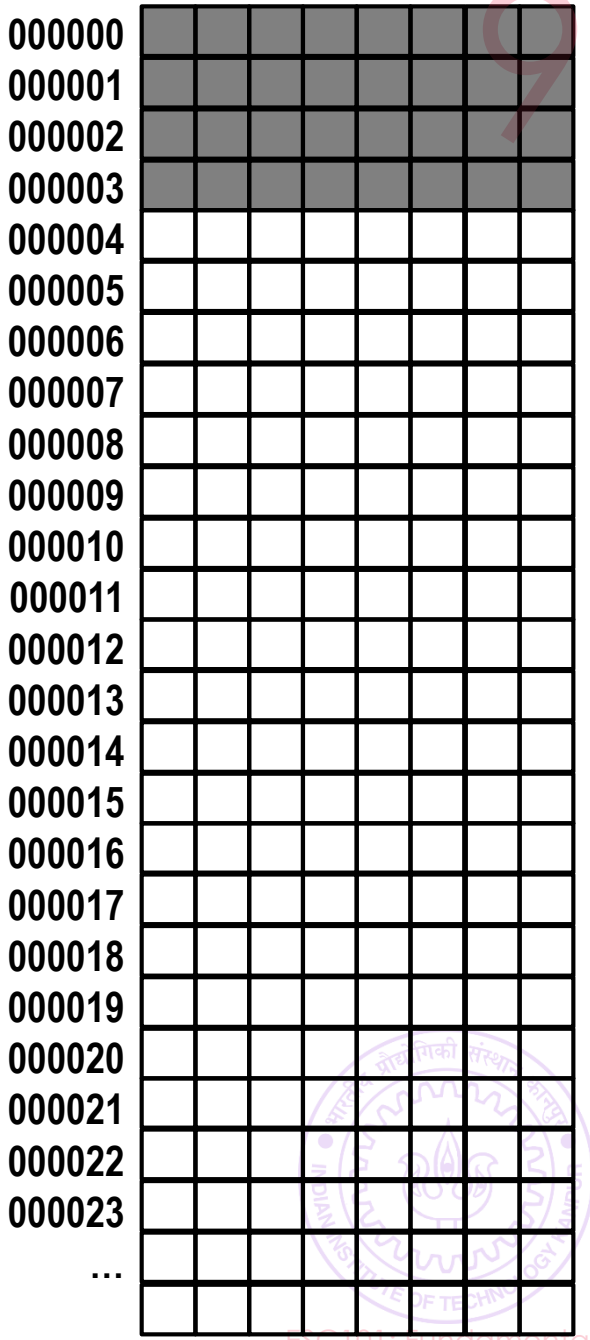
If we declare an array, a sequence of addresses get allocated



How Mr C stores arrays

If we declare an array, a sequence of addresses get allocated

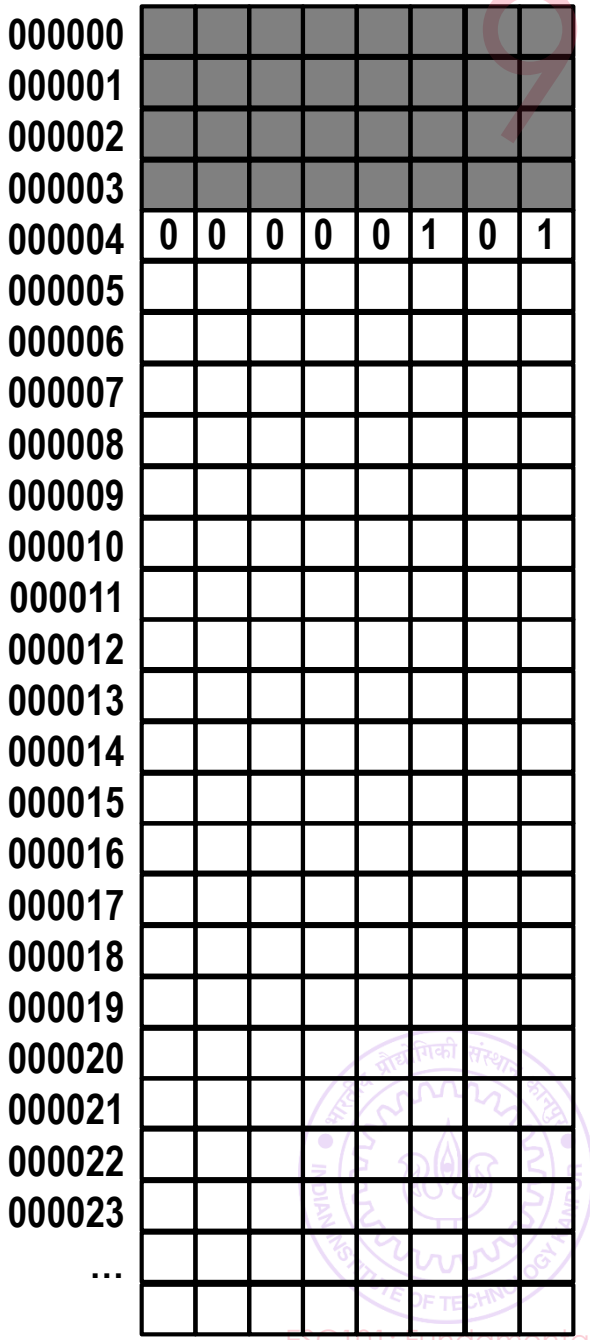
```
char c[5];
```



How Mr C stores arrays

If we declare an array, a sequence of addresses get allocated

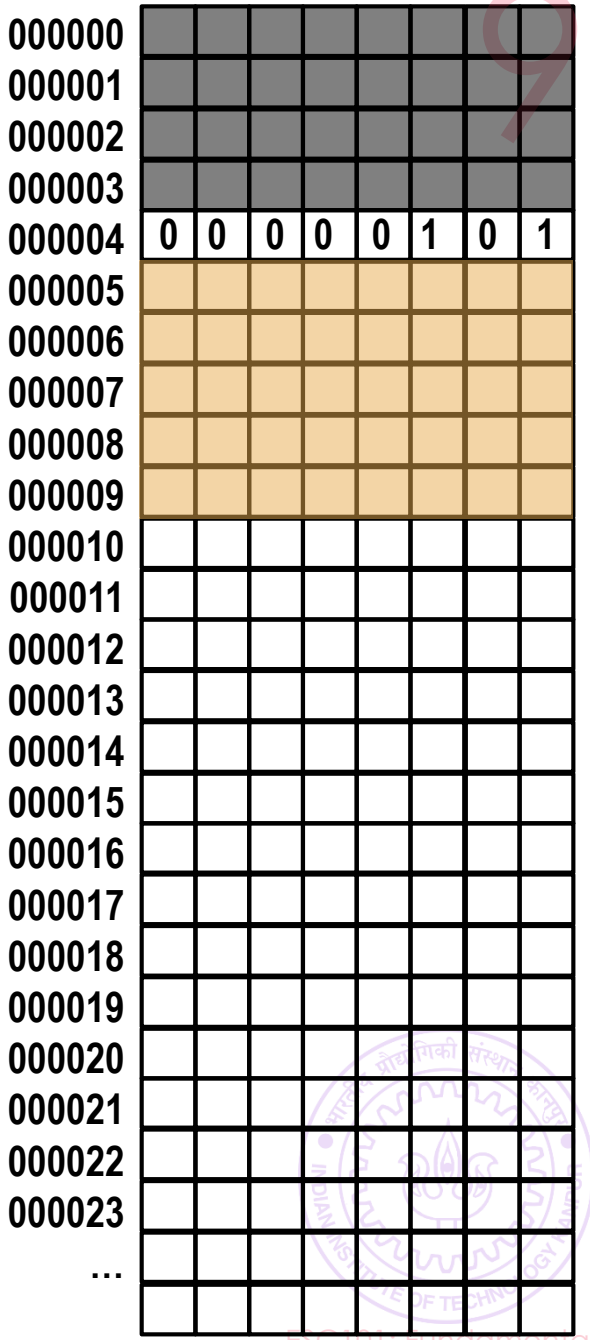
```
char c[5];
```



How Mr C stores arrays

If we declare an array, a sequence of addresses get allocated

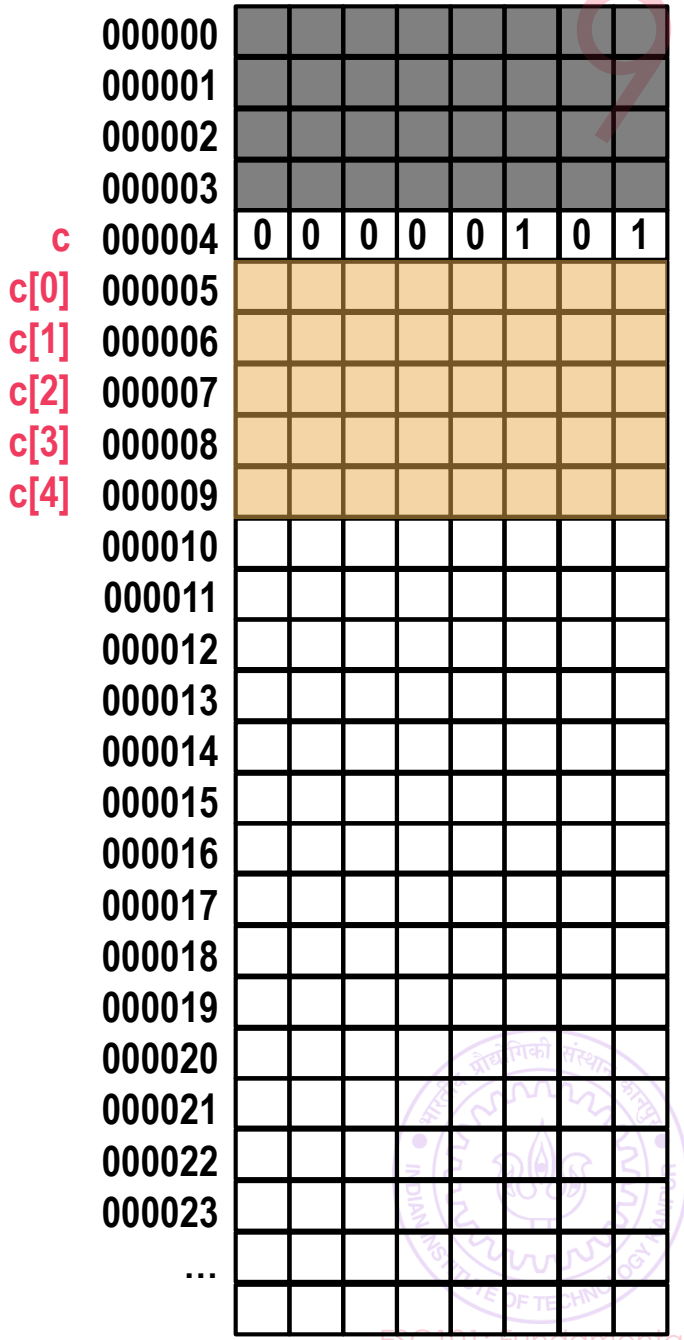
```
char c[5];
```



How Mr C stores arrays

If we declare an array, a sequence of addresses get allocated

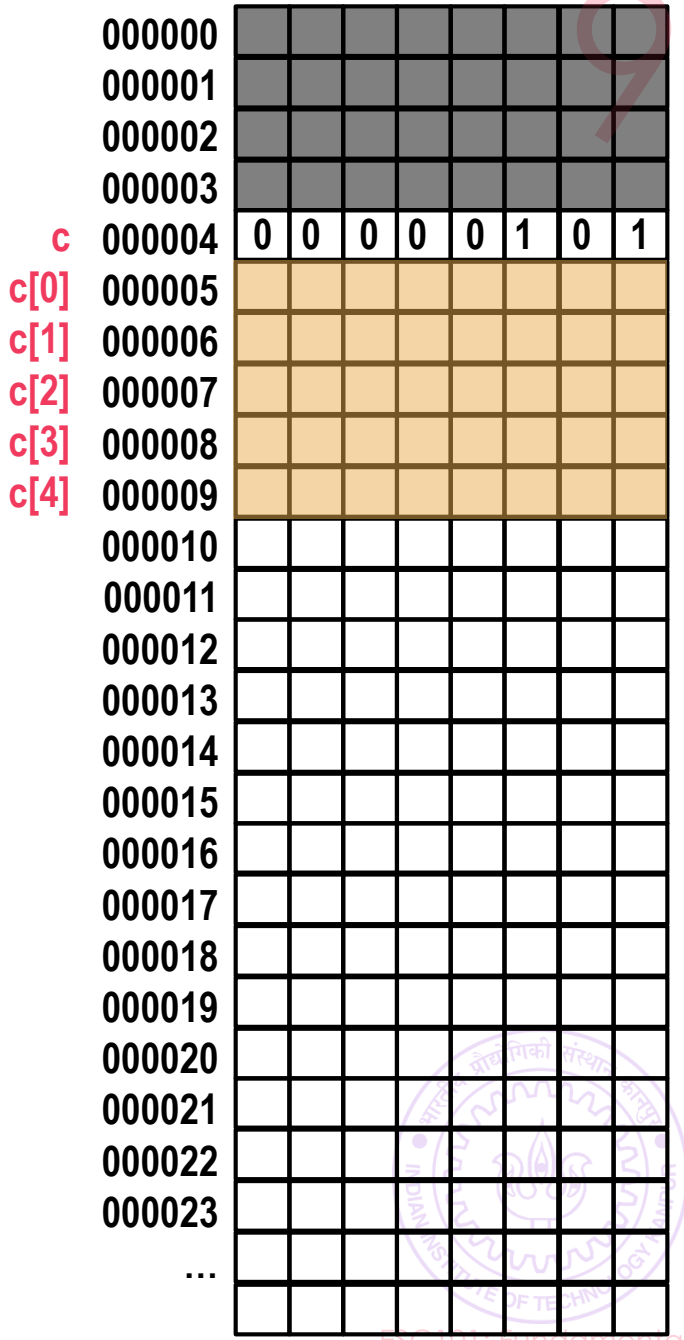
```
char c[5];
```



How Mr C stores arrays

If we declare an array, a sequence of addresses get allocated

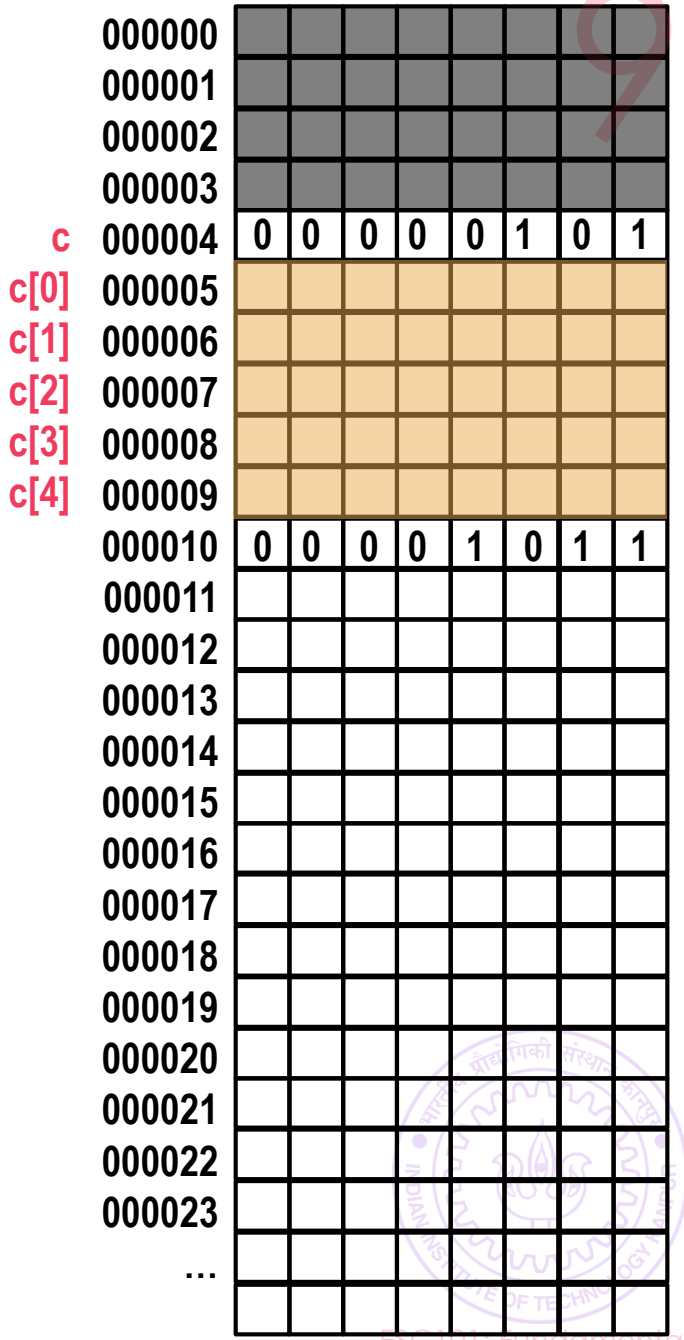
```
char c[5];
int a[3];
```



How Mr C stores arrays

If we declare an array, a sequence of addresses get allocated

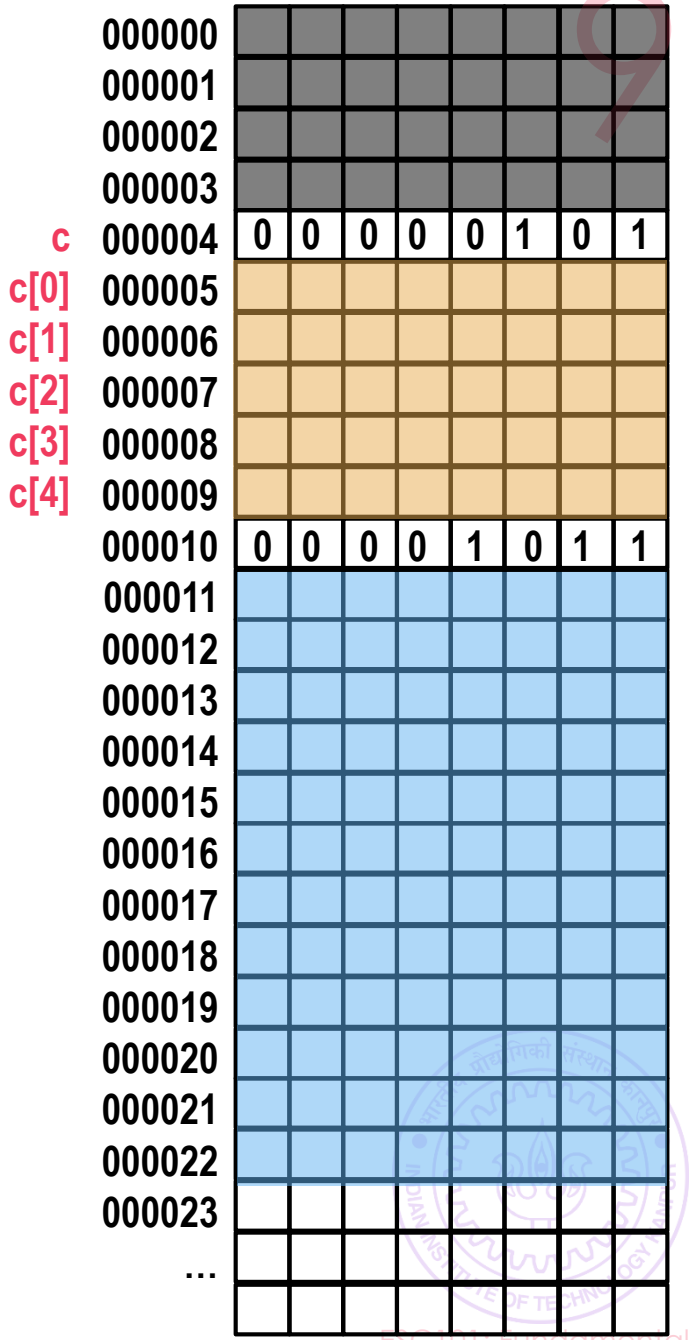
```
char c[5];
int a[3];
```



How Mr C stores arrays

If we declare an array, a sequence of addresses get allocated

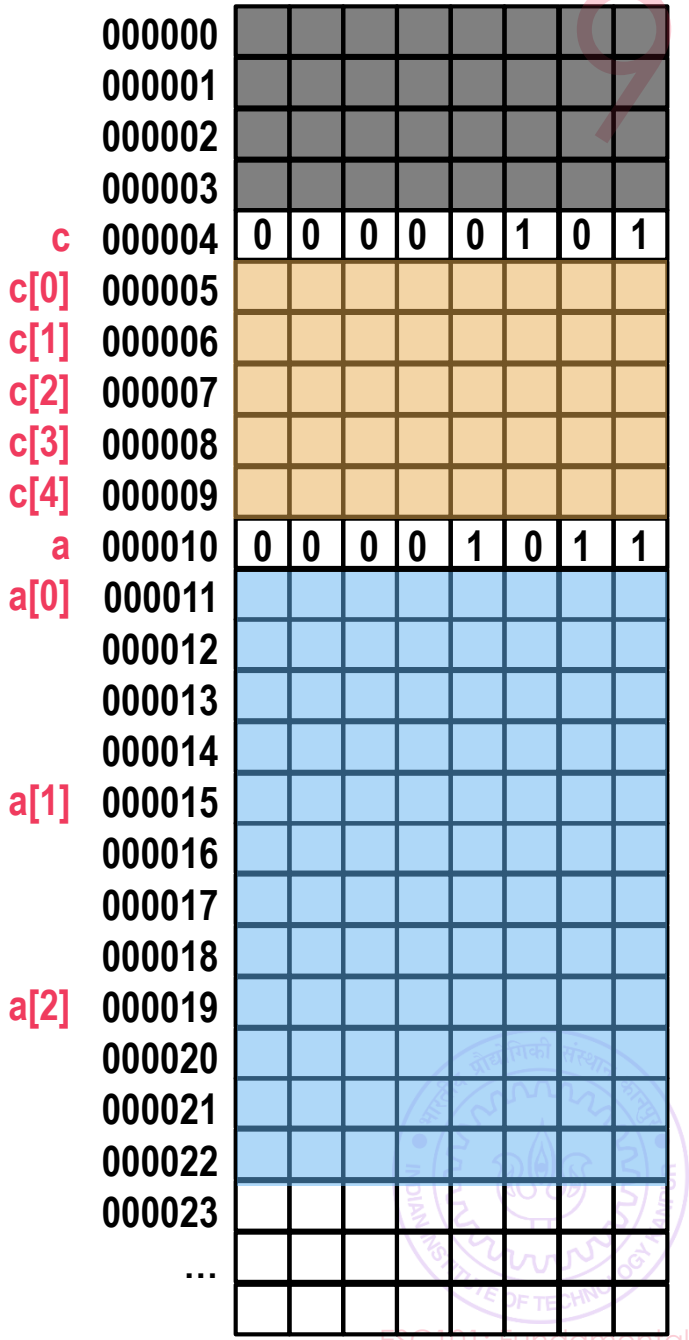
```
char c[5];
int a[3];
```



How Mr C stores arrays

If we declare an array, a sequence of addresses get allocated

```
char c[5];
int a[3];
```

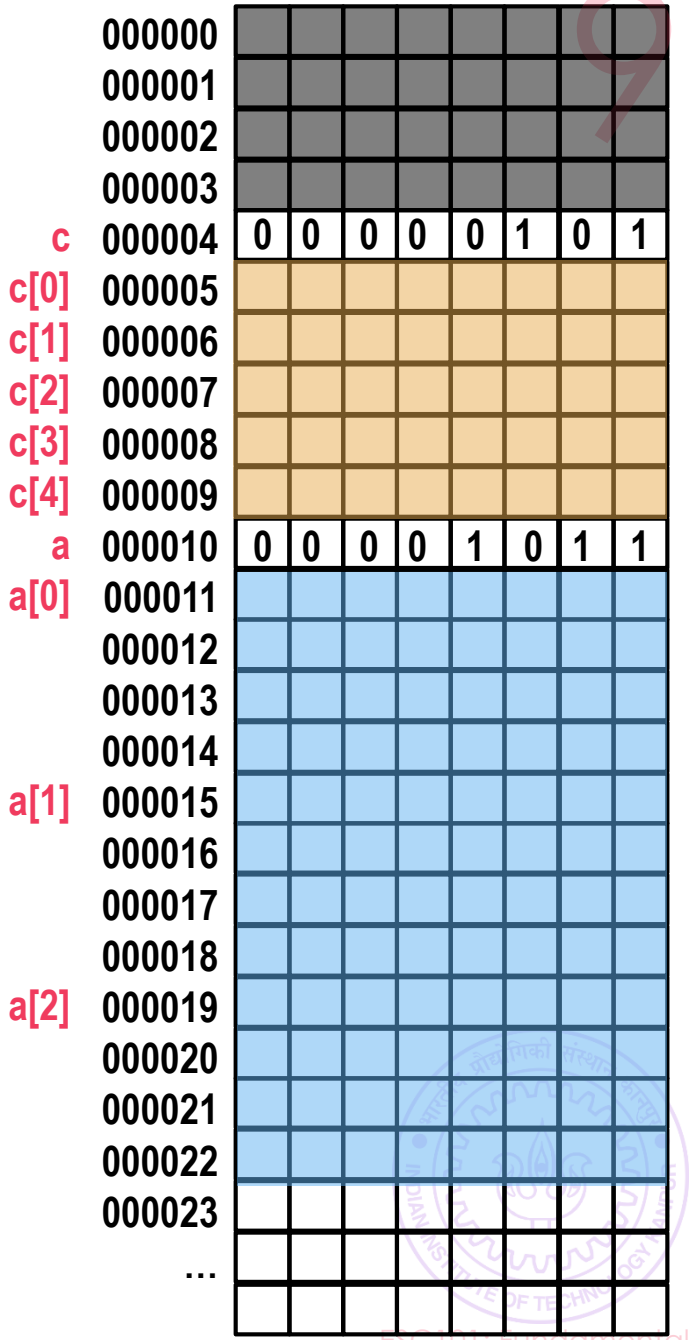


How Mr C stores arrays

If we declare an array, a sequence of addresses get allocated

```
char c[5];
int a[3];
```

c and a are actually pointers, c stores the address of c[0], a stores address of a[0]



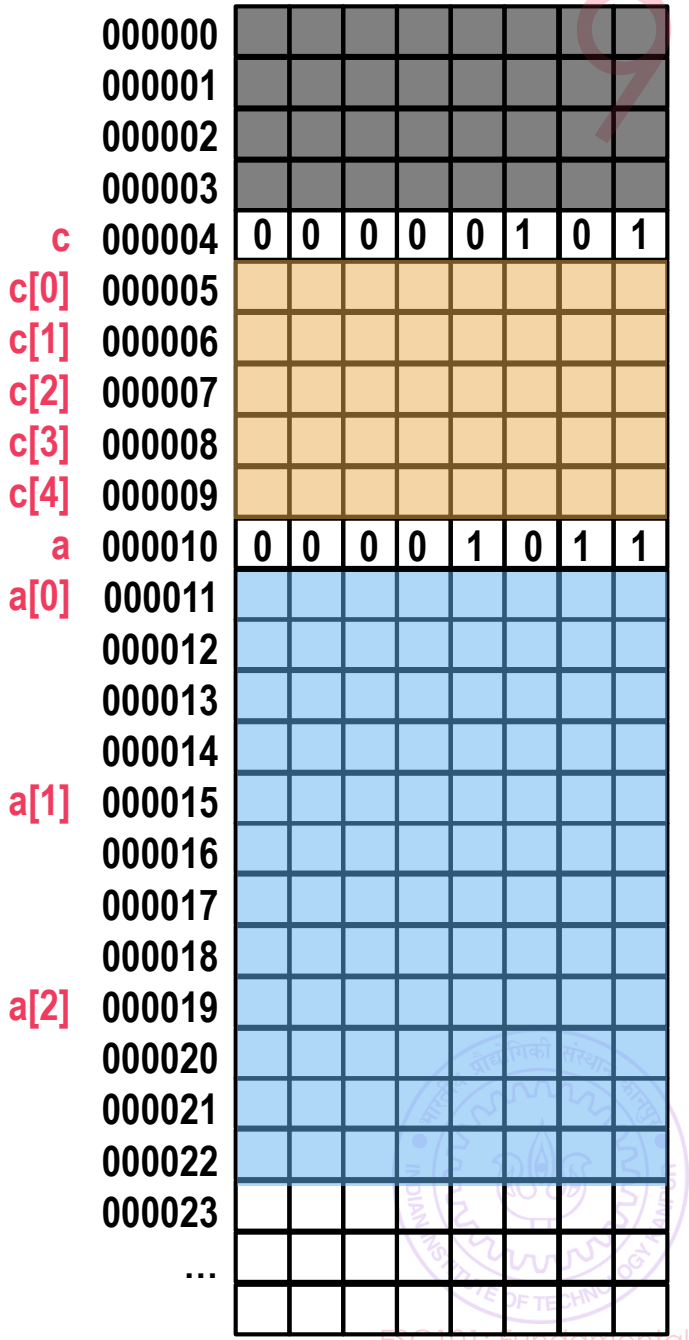
How Mr C stores arrays

If we declare an array, a sequence of addresses get allocated

```
char c[5];
int a[3];
```

c and a are actually pointers, c stores the address of c[0], a stores address of a[0]

c[0] is stored at address 000005, c[1] at address 000006, c[2] at 000007 and so on



How Mr C stores arrays

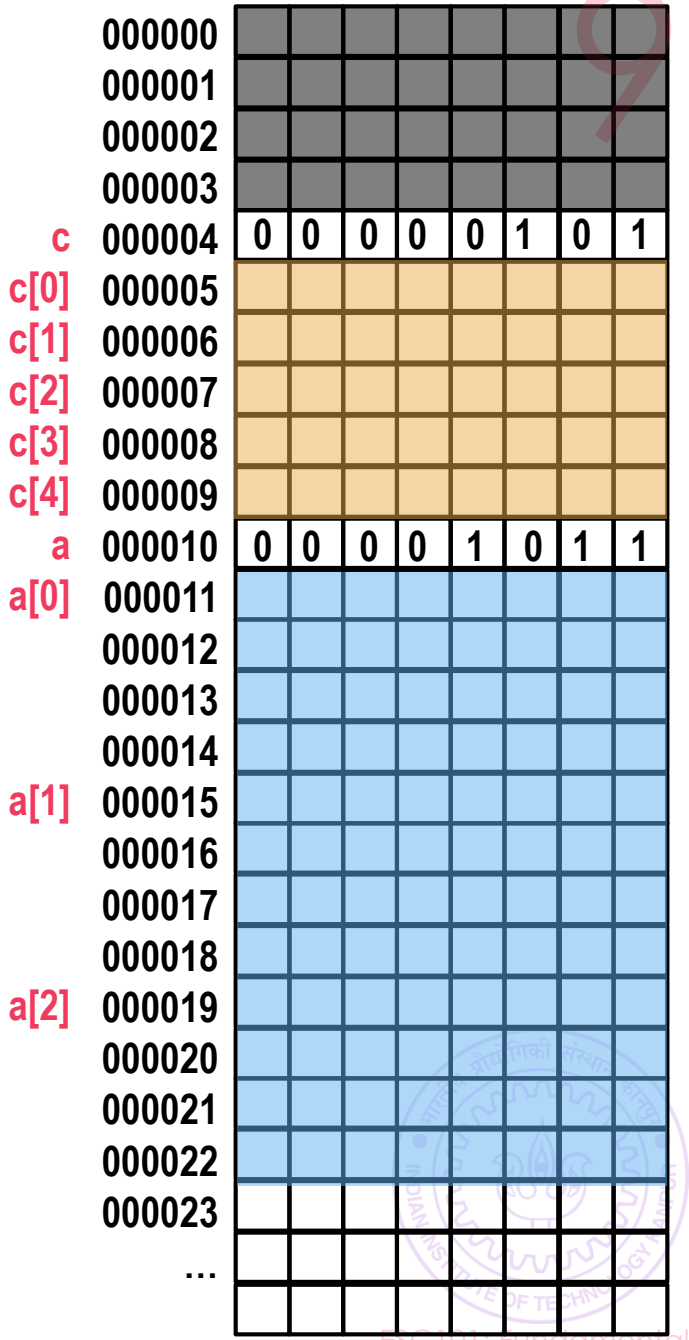
If we declare an array, a sequence of addresses get allocated

```
char c[5];
int a[3];
```

c and a are actually pointers, c stores the address of c[0], a stores address of a[0]

c[0] is stored at address 000005, c[1] at address 000006, c[2] at 000007 and so on

a[0] is stored at address 000011, a[1] at address 000015 (int takes 4 bytes), a[2] at address 000019, and so on



How C stores arrays

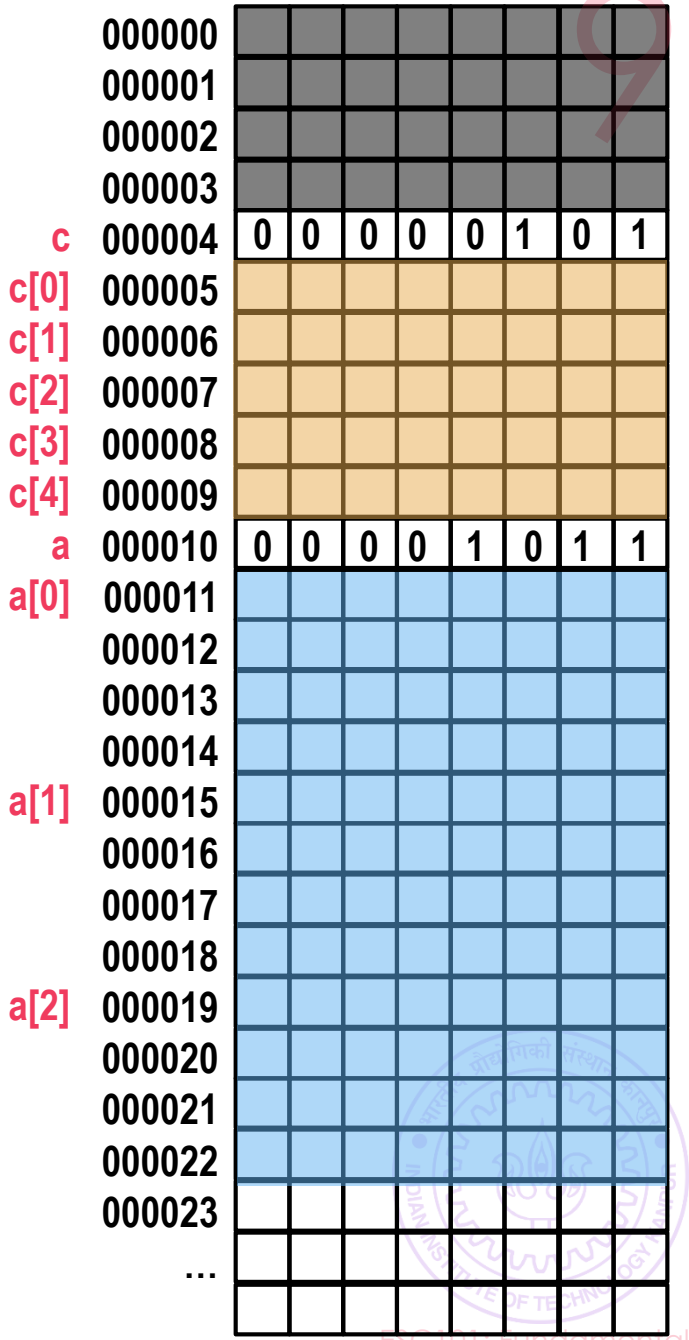
If we declare an array, a sequence of addresses get allocated

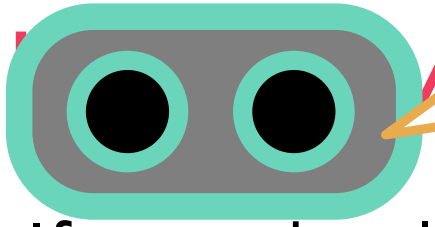
```
char c[5];
int a[3];
```

c and a are actually pointers, c stores the address of c[0], a stores address of a[0]

c[0] is stored at address 000005, c[1] at address 000006, c[2] at 000007 and so on

a[0] is stored at address 000011, a[1] at address 000015 (int takes 4 bytes), a[2] at address 000019, and so on





Actually, being pointers, c and a should themselves take 8 bytes to store the addresses

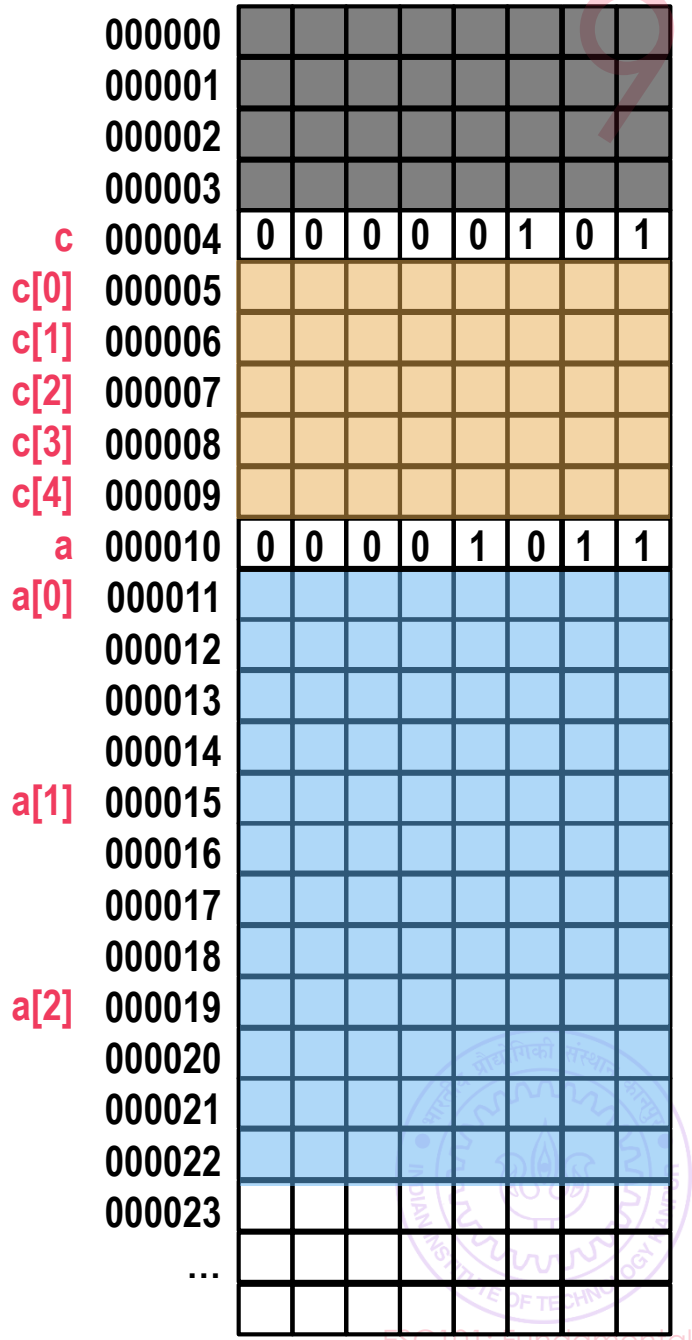
If we declare an array, a sequence of addresses get allocated

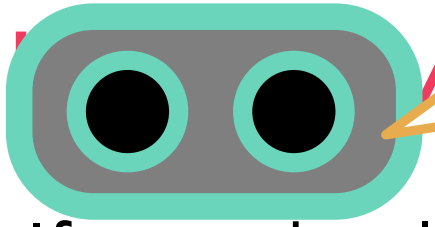
```
char c[5];
int a[3];
```

c and a are actually pointers, c stores the address of c[0], a stores address of a[0]

c[0] is stored at address 000005, c[1] at address 000006, c[2] at 000007 and so on

a[0] is stored at address 000011, a[1] at address 000015 (int takes 4 bytes), a[2] at address 000019, and so on





Actually, being pointers, c and a should themselves take 8 bytes to store the addresses

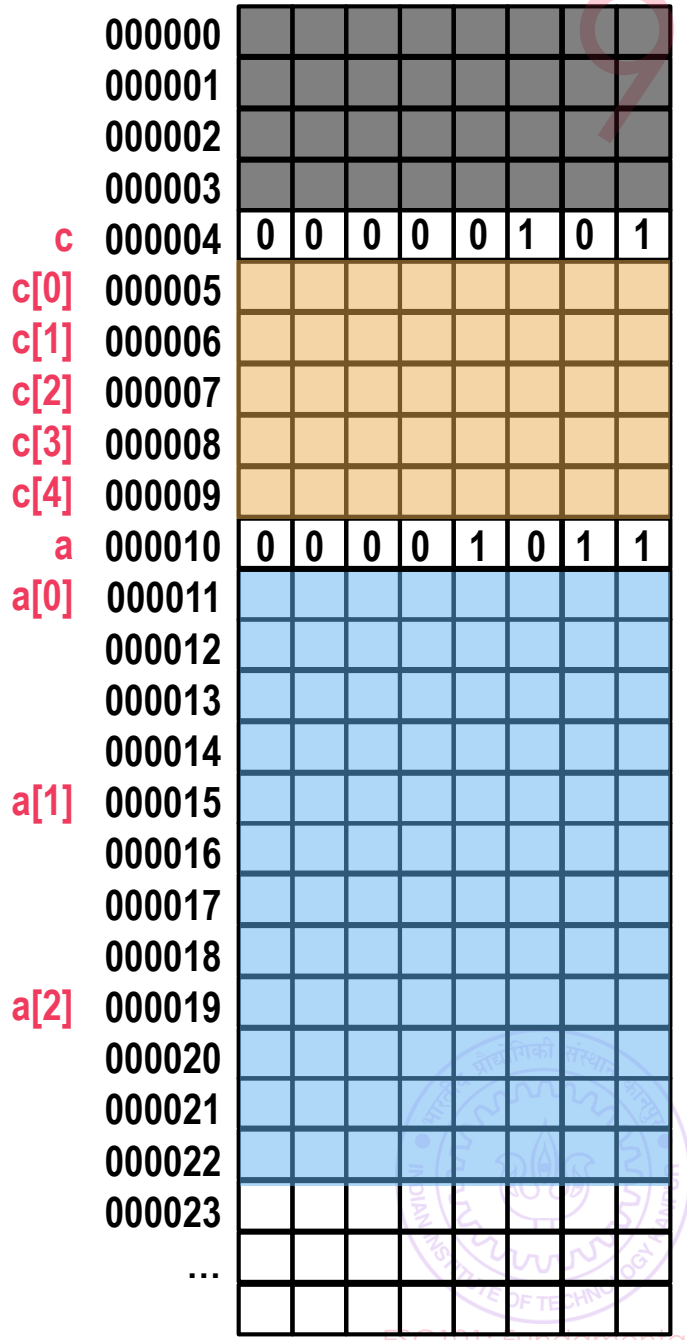
If we declare an array, a sequence of addresses get allocated

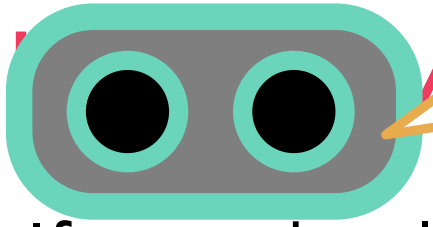
```
char c[5];
int a[3];
```

c and a are actually pointers, c stores the address of c[0], a stores address of a[0]

c[0] is stored at address 000005, c[1] at address 000006, c[2] at 000007 and so on

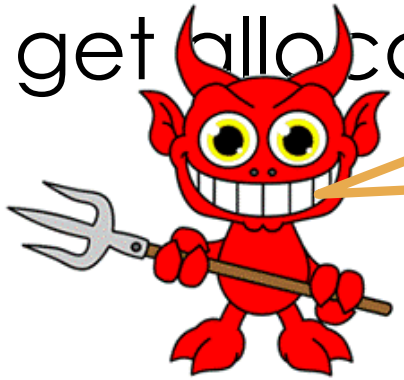
a[0] is stored at address 000011, a[1] at address 000015 (int takes 4 bytes), a[2] at address 000019, and so on





Actually, being pointers, c and a should themselves take 8 bytes to store the addresses

If we declare an array ... of addresses get allocate...



Yes, but I ran out of space space as well as ran out of patience drawing boxes

```
char c[5];
int a[3];
```

c and a are actually pointers, c stores the address of c[0], a stores address of a[0]

c[0] is stored at address 000005, c[1] at address 000006, c[2] at 000007 and so on

a[0] is stored at address 000011, a[1] at address 000015 (int takes 4 bytes), a[2] at address 000019, and so on