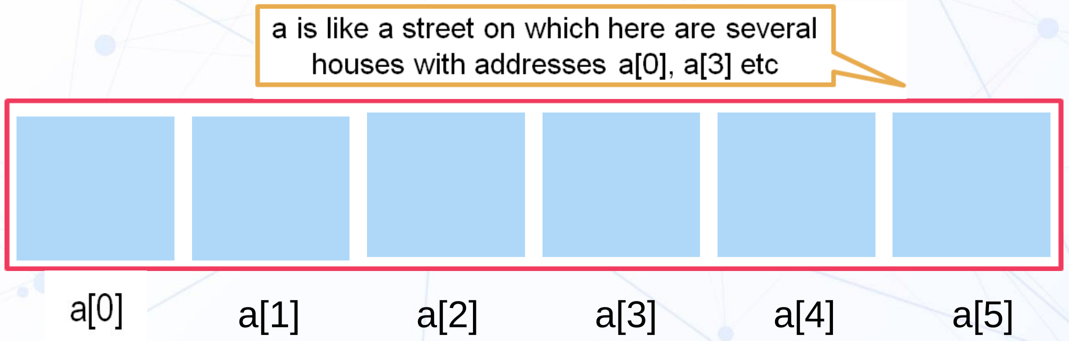
Arrays

The English word array means “objects in a line” or “an ordered series or arrangement” – *The soldiers standing in an array impressed the visiting head of the state on 26 Jan.*

In C, an array is a sequence of variables can have array of ints, longs, floats, doubles or characters.

Array subscript

```
int a[6],b=5;
a[b-4]=4;
a[4]=3;
a[-1]=6;
printf(“%d”, a[a[1]]);
a[6]=4;
```


a is like a street on which here are several houses with addresses a[0], a[3] etc
a[0]
a[1]
a[2]
a[3]
a[4]
a[5]

Can use **integer** expressions as array subscripts **not float/double**

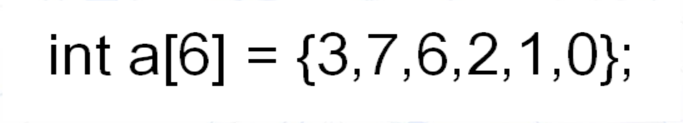
Robotics Club IITKanpur

a[0] to a[5] are just integer variables. Use them as you did any other integer variable – **first variable is a[0] not a[1]**

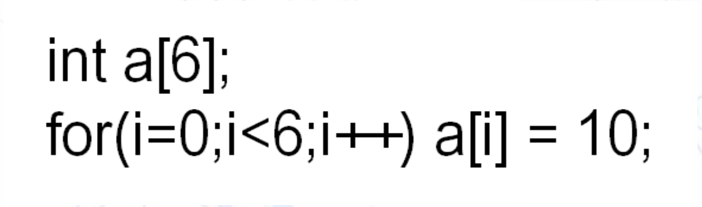
a = 564; does not make sense – a isn't a single int variable.

If you want to give values to whole array

Can do it at the time of declaring the array itself



```
int a[6] = {3,7,6,2,1,0};
```

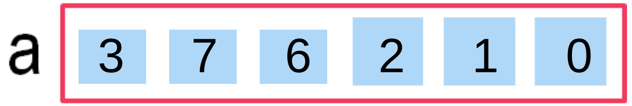


```
int a[6];
for(i=0;i<6;i++) a[i] = 10;
```

Initializing arrays

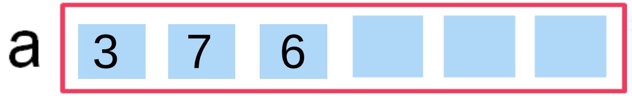
Can be initialized at time of declaration itself

```
int a[6] = {3,7,6,2,1,0};
```



Can be partly initialized as well

```
int a[6] = {3,7,6};
```



However, after declaration done, have to be initialized one by one!

```
int a[6];
a = {3,7,6,2,1,0};
```

✗

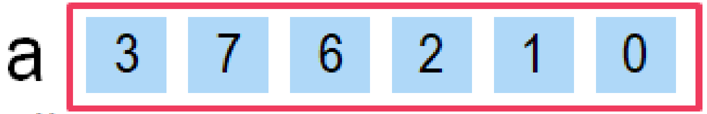
```
int a[6];
a[2] = 6;
```

✓

More on initializing arrays

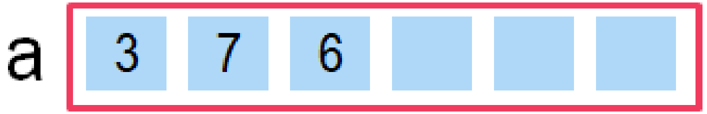
Can be initialized at time of declaration itself

```
int a[6] = {3,7,6,2,1,0};
```



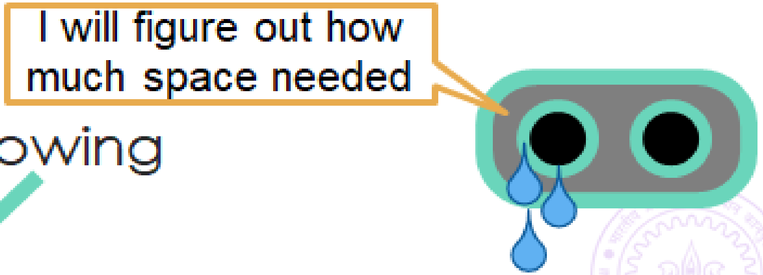
Can be partly initialized as well

```
int a[6] = {3,7,6};
```



Over initialization may crash

```
int a[6] = {1,2,3,4,5,6,7,8,9};
```



Better way to initialize is the following

```
int a[] = {1,2,3,4,5,6,7,8,9};
```

✓

Warning: uninitialized arrays contain garbage, not zeros

Highly compiler dependent feature

Robotics Club IITKanpur

More about arrays

Arrays can not be copied like normal variables

```
int a[3] = {1,2,3}, b[3];
b = a;
```

❌

Will result in an error

Arrays can also not be checked for equality directly

```
int a[3] = {1,2,3}, b[3] = {4,5,6};
if(b == a) printf("Equal");
```

❌

Will not result in error but b == a will always be false

Reason will be clear in a couple of weeks

ESC101: Fundamentals of Computing

Robotics Club IITKanpur

Operator Name	Symbol/Sign	Associativity
Brackets (**array subscript**), Post increment/decrement	(), **[]** ++, --	Left
Unary negation, Pre increment/decrement, NOT	-, ++, --, !	Right
Multiplication/division/ remainder	*, /, %	Left
Addition/subtraction	+, -	Left
Relational	<, <=, >, >=	Left
Relational	==, !=	Left
AND	&&	Left
OR	||	Left
Ternary Conditional	? :	Right
Assignment, Compound assignment	=, +=, -=, *=, /=, %=	Right

Very important!!

Robotics Club IITKanpur

Character Arrays

All things we learnt about int/float arrays apply here too

However, much more exciting things can be done here

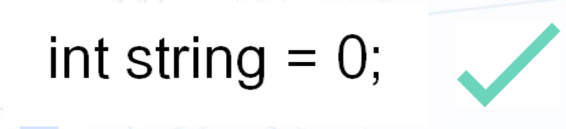
Char arrays also called *strings (well … almost all of them)*

English word *string* means a thread or a collection of items put together. *The pearls were strung together.*

In C, string implies a character array (well … almost)

Note: string is **not a datatype** in C

Word string is **not a keyword** in C

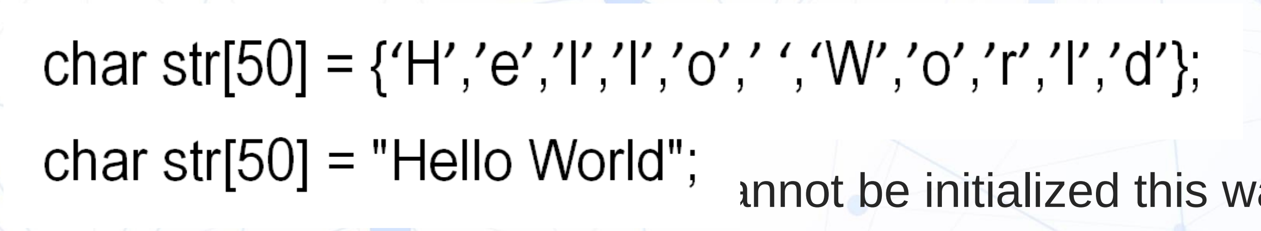


```
int string = 0;
```

✓

Declaring and using strings

Can be initialized at time of declaration



```
char str[50] = {'H','e','l','l','o',' ','W','o','r','l','d'};
char str[50] = "Hello World";
```

annot be initialized this way

after declaration is done. “Space” is also a character.

Other ways: scanf (with %s), gets

To print: puts, printf (with %s)

The null character

ASCII value 0: used to signal the end of a string

Can actually print and read a null character – escape sequence \0

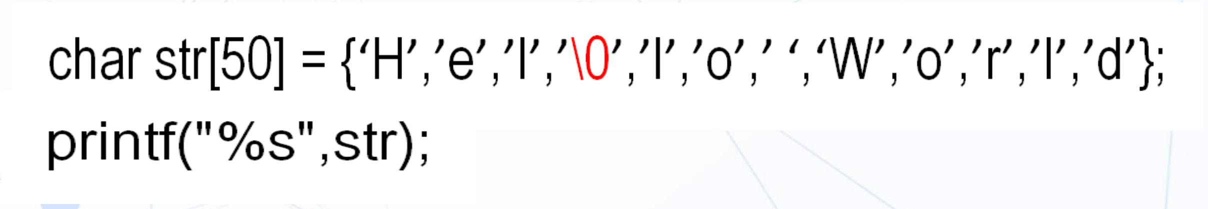
Character arrays with a null character called strings

Delimiter: a character or symbol used to signal the end of a list or end of a stream

In many questions where input is a list of numbers, -1(or 0) is delimiter

Stop reading numbers after -1(or 0) is encountered.

For strings null character is delimiter – Compiler stops reading after \0



```
char str[50] = {‘H’,’e’,’l’,’\0’,’l’,’o’,’ ‘,‘W’,’o’,’r’,’l’,’d’};
printf("%s",str);
```

How the compiler stores your variables

Compiler loves binary digits so much so they gave a cute nickname ***bit*** – short for **b**inary dig**it**

A set of 8 bits has an even cuter nickname ***byte.***

All variables, int, long, char, float, double, arrays are stored in binary format using one or more bytes.

Robotics Club IITKanpur

The size of various datatypes

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

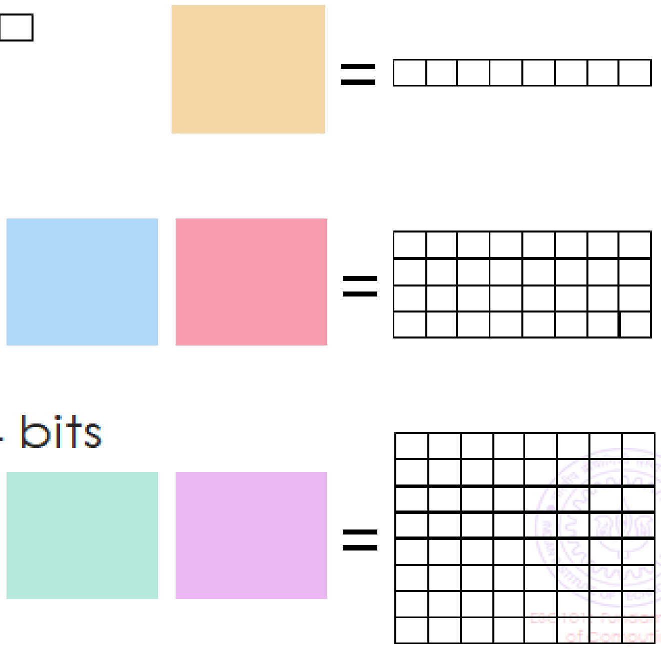
  Max value in int is $2^{(32-1)}-1$ – verify

  Max value of float discussed later

long/double takes 8 bytes = 64 bits

  Max value in long is $2^{(64-1)}-1$ – verify

  Max value of double discussed later



Robotics Club IITKanpur

The compiler 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 other uses.

Others can be used by you for variables

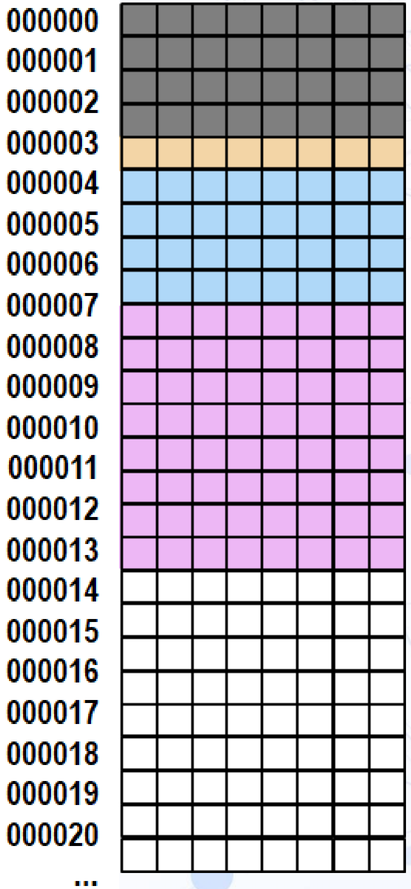
char c;

int a;

float d;

So c is stored at address 000004, a at 000005

and d at address 000009



Robotics Club IITKanpur

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 the compiler to manage stuff

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

Robotics Club IITKanpur

How are arrays stored

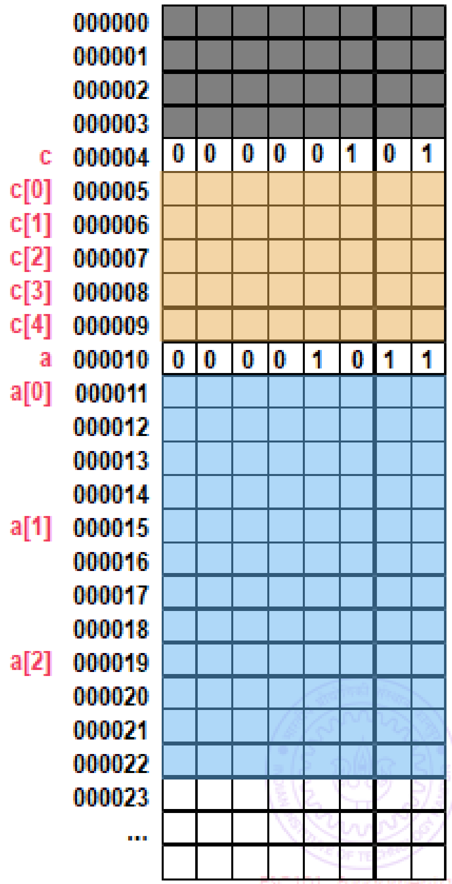
If we declare an array, a sequence of addresses get allocated

```
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

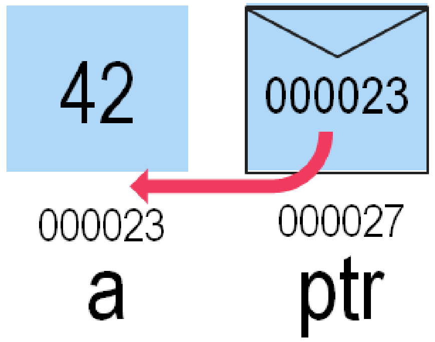
a[0] is stored at address 000011, a[1] at address 000015 (int takes 4 bytes), a[2] at address 000019, and so on



How we must speak to compiler

How we usually speak to a human

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



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

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

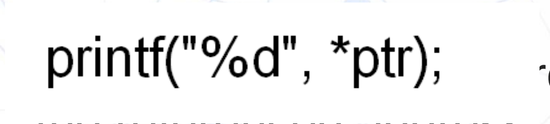
a is stored at internal location 000023
int takes 4 bytes to store

Robotics Club IITKanpur

Pointers with printf and scanf

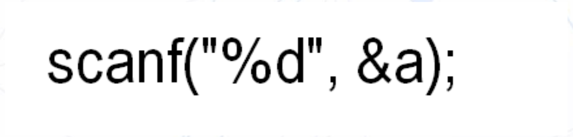
Pointers contain addresses, so to print the address itself, use the %ld format since addresses are 8 byte long

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

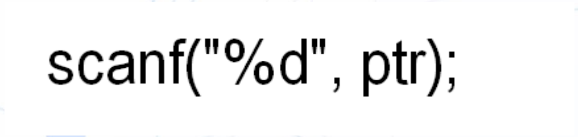


```
printf("%d", *ptr);
```

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



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



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

Pointer arithmetic

Array names are pointers to first element of the array

Warning: consecutive addresses only assured in arrays.

int c[10];

int a,b,ptr = c;

a, b need not be placed side-by-side (i.e. 4 bytes apart) but c[0], c[1] will always be 4 bytes apart (int takes 4 bytes)

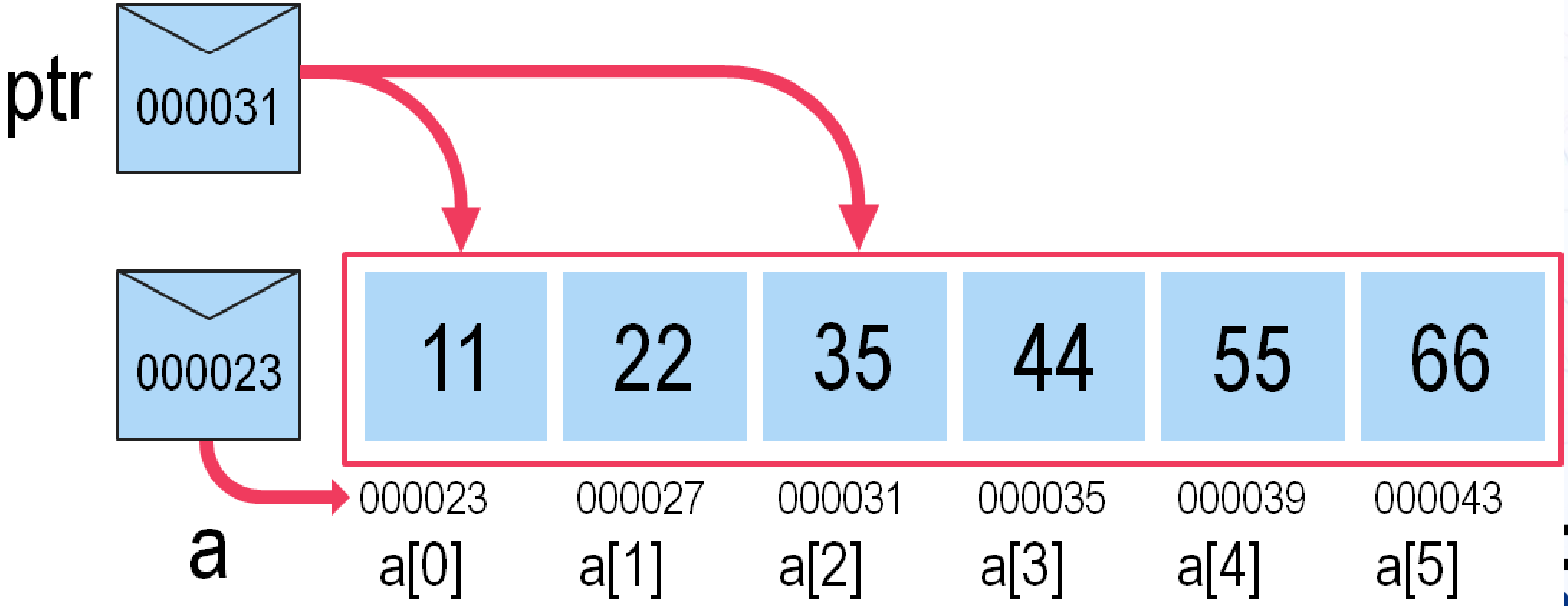
Pointer arithmetic often used to traverse (go back and forth in) arrays and calculate offsets **c[2]** and ***(c+2)** both give value of the 3rd element in c

Warning: c++ will give error, ptr++ will move pointer to c[1].

Pointers and Arrays

```
int a[6] = {11,22,33,44,55,66};
int *ptr = a;
```

```
ptr += 2;
*ptr += 2;
```



Variable-length arrays

So far we have always used arrays with constant length

int c[10];

Waste of space – often allocate much more to be “safe”. Also need to remember how much of array actually used. Rest of the array may be filled with junk (not always zeros).

In strings NULL character does this job.For other types of arrays, need to do this ourselves .

Lets us learn ways for on-demand memory allocation

The secret behind getline and other modern functions

Need to include stdlib.h for these functions malloc(), calloc(),

realloc(), free().

Robotics Club IITKanpur

Malloc - **m**emory **alloc**ation

We tell malloc how many bytes are required

malloc allocates those many **consecutive** bytes

Returns the address of (a pointer to) the first byte

Warning: allocated bytes filled with garbage

Warning: if insufficient memory, NULL pointer returned

malloc has no idea if we are allocating an array of floats or chars – returns a void* pointer – typecast it yourself

The allocated memory can be used safely as an array

Calloc – **c**ontiguous **alloc**ation

A helpful version of malloc that initializes memory to 0 .

However, slower than malloc since time spent initializing

Use this if you actually want zero initialization

Syntax a bit different – instead of total number of bytes, we need to send it two things

- length of array (number of elements in the array)
- number of bytes per element

Sends back a NULL pointer if insufficient memory – careful!

Need to typecast the pointer returned by calloc too!

Robotics Club IITKanpur

Robotics
Club IITKanpur