Pointers and Memory Allocation (Continued)

ESC101: Fundamentals of Computing

Nisheeth

So far about pointers..

What is a pointer - An address in memory

How to declare a pointer variable - type * ptrName;

Every pointer variable has a data type

type * (not type) is data type of above pointer ptrName

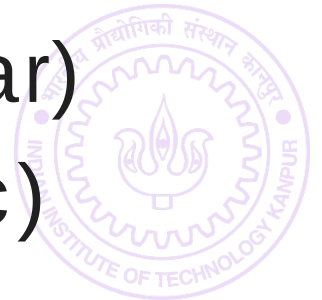
After declaration, can use *ptrName to dereference

Pointer arithmetic. Can add/subtract to go forward/back

Pointers and arrays (array name is pointer to first element)

Pointers and strings (string name is pointer to first char)

Memory allocation functions (malloc, calloc, realloc)



Reminder: Some basics about arrays and pointers

Consider an array int arr[6] = {2,4,1,3,5,7};

arr (name of the array) is the same as &arr[0]

Address of the i-th element is arr+i or &arr[i]

Value of the i-th element is *(arr+i) or arr[i]

All of the above is true for any type of array

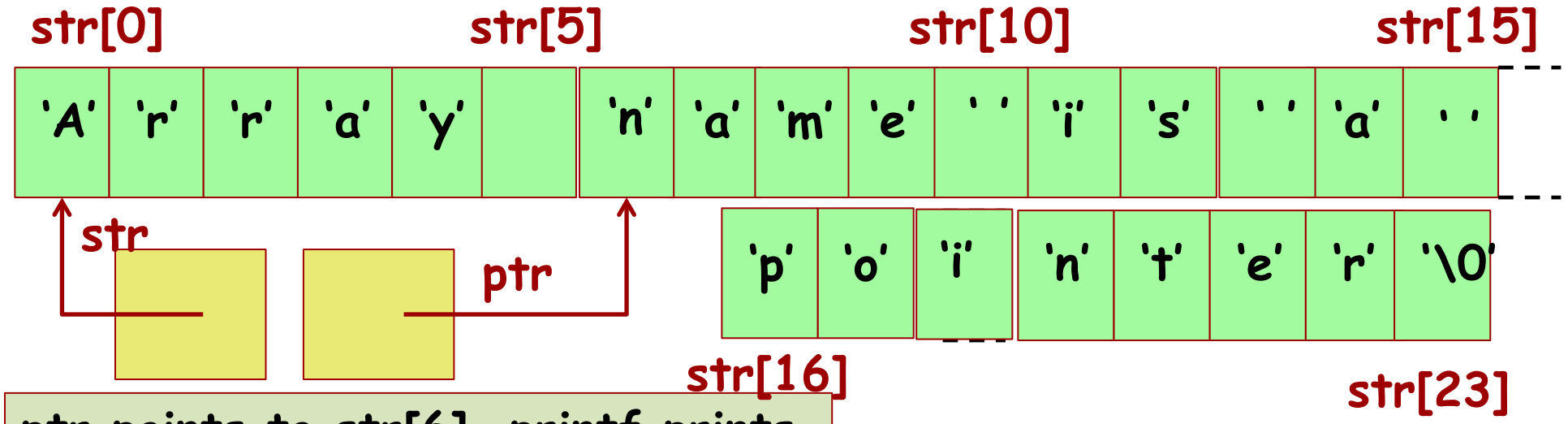
String's name is the pointer to the first character of string (so string pointer is of type char *)

String's name is used directly by scanf to read the full string (Without &)

String's name is used directly by printf to print the full string

Pointers and strings: A simple example


```
char str[] ="Array name is a pointer";
char *ptr = str + 6; /*initialize*/
printf("%s",ptr);
```



ptr points to str[6]. printf prints the string starting from str[6].

Output: name is a pointer

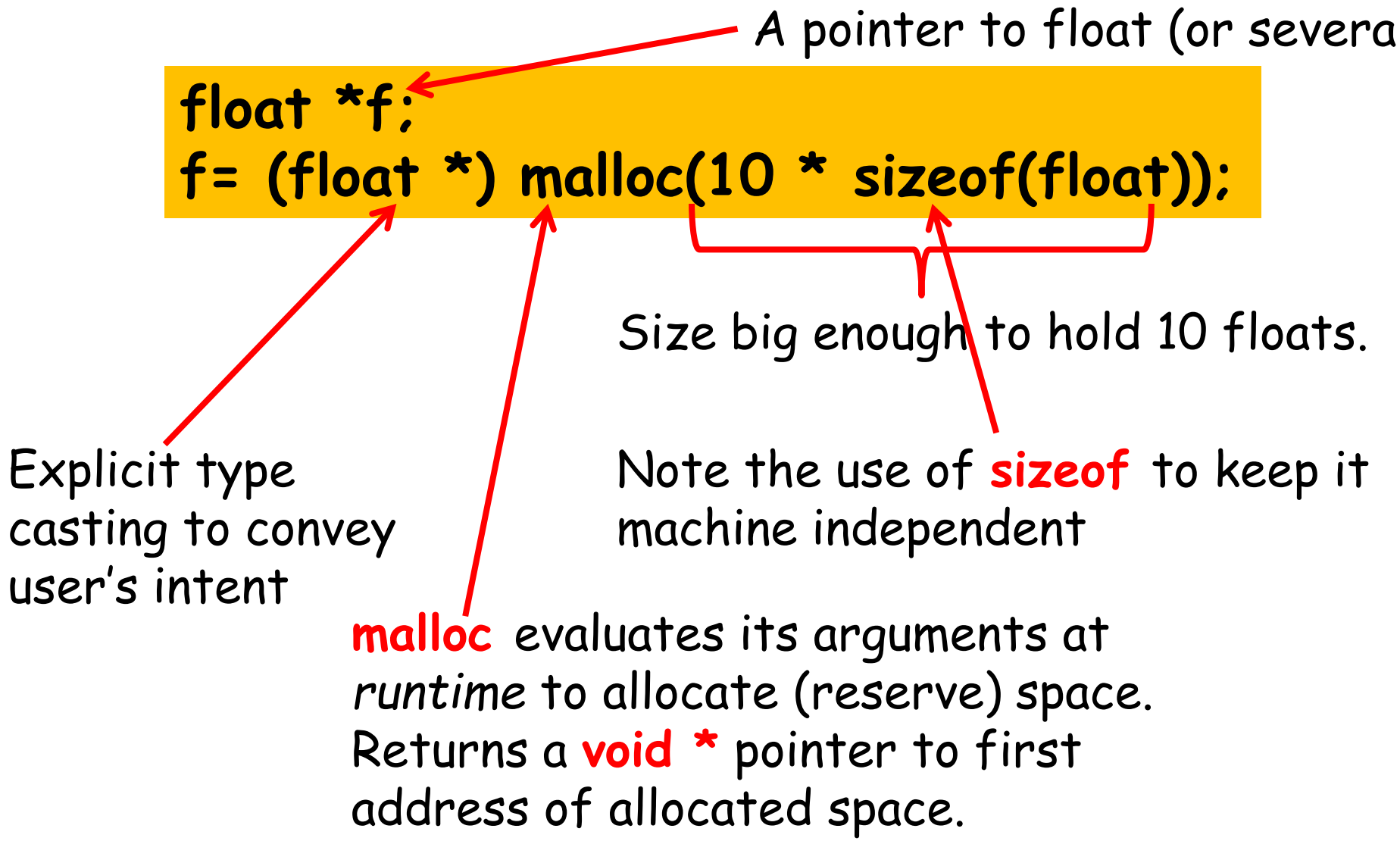
Back to memory allocation related functions

malloc calloc

free

realloc

malloc: Example



malloc: Example

Key Point: The size argument can be a variable or non-constant expression!

After memory is allocated, pointer variable behaves as if it is an **array**!

```
float *f; int n;
scanf("%d", &n);
f= (float *) malloc(n * sizeof(float));

f[0] = 0.52;
scanf("%f", &f[3]); //Overflow if n<=3
printf("%f", *f + f[0]);
```

This is because, in C, f[i] simply means *(f+i).

calloc

Similar to malloc except for zero initialization

Syntax is slightly different from malloc

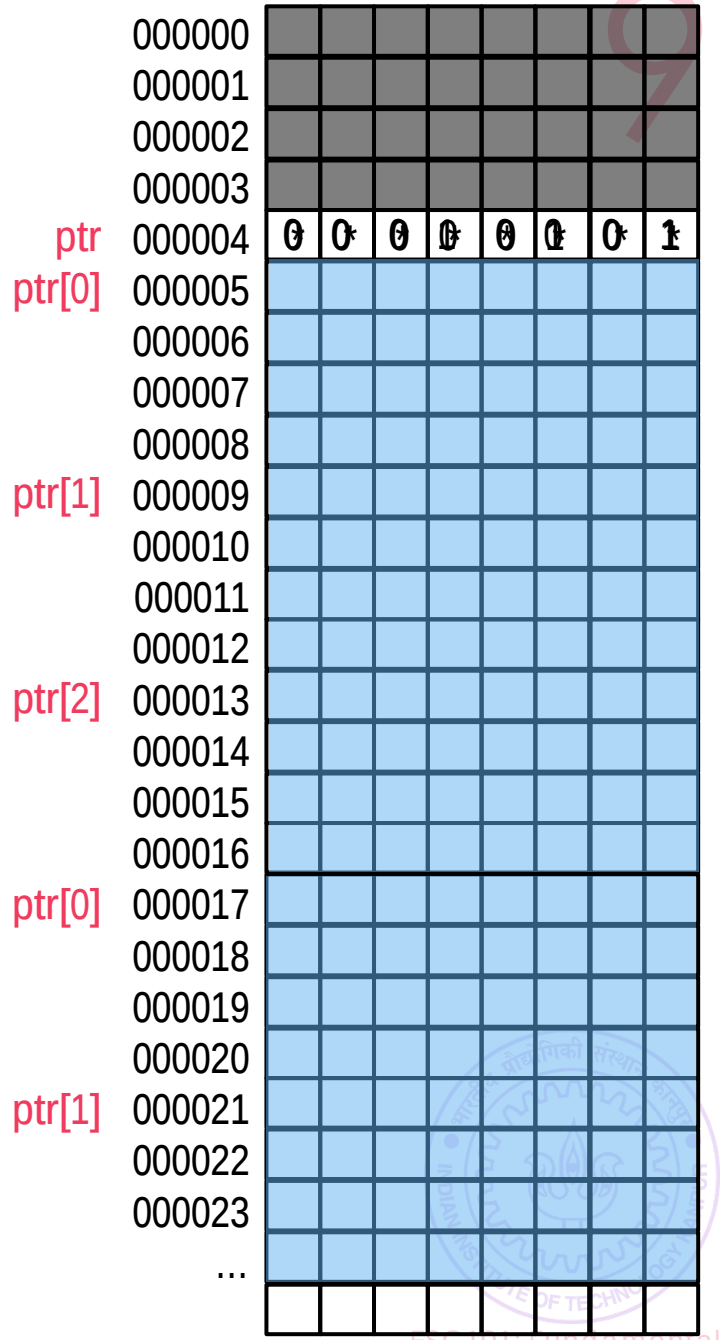
```
float *f;
f= (float *) calloc(10, sizeof(float));
```

How many elements?

Size of each element

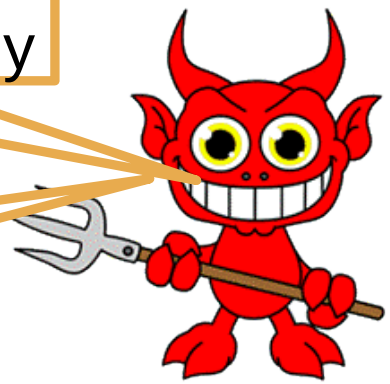
Memory leaks

Situation where memory allocated earlier becomes unusable and blocked ☹



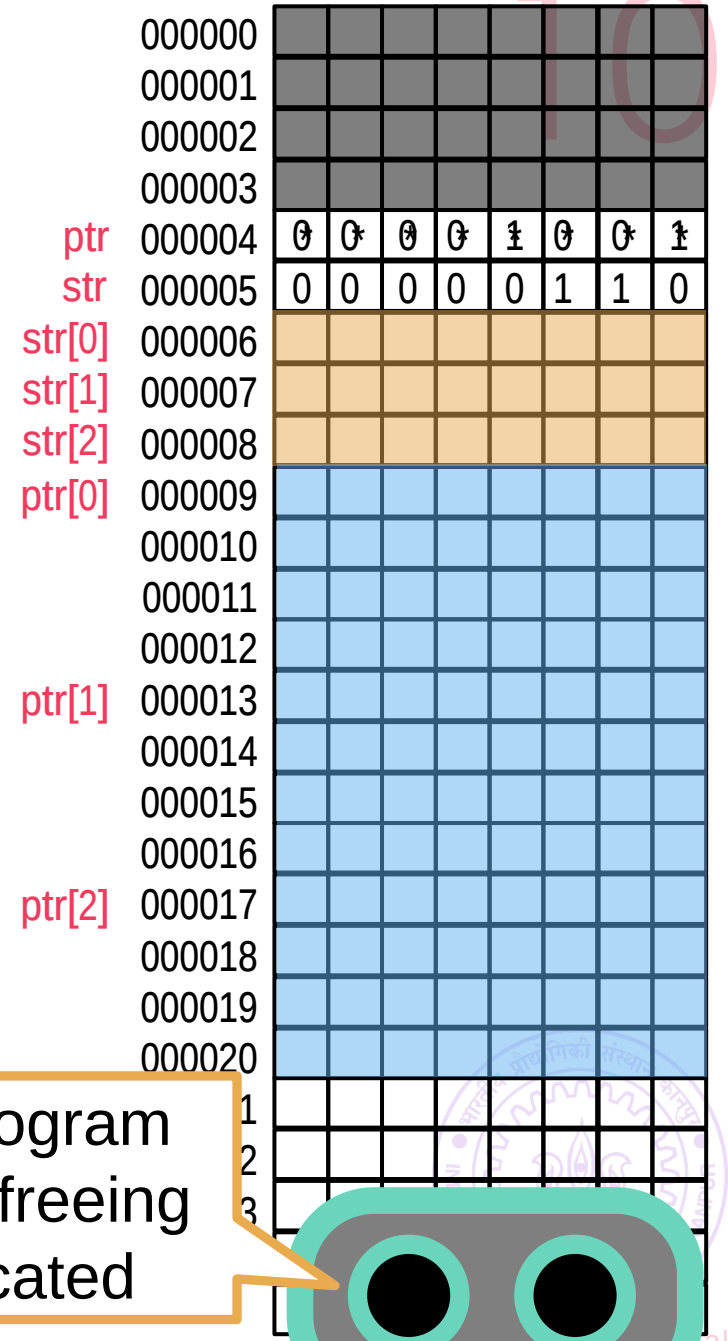
ptr will take 8 bytes to store – sorry for not drawing accurately

If you keep losing memory like this, soon your program may crash!



free

Used to deallocate (fee) memory allocated using malloc/calloc/realloc



Don't use freed memory or free memory twice or free no
arrays – will caus

```
free(str); // runtime error
```

I always free all memory when a program ends. You only have to worry about freeing memory that you asked to be allocated

Library analogy for malloc/free

malloc/calloc is like borrowing a book from library

If that book unavailable, cannot use it (NULL pointer)

1000+ students in Y19 but only 50 copies of Thomas' Calculus

free is like returning a book so others can use it after you

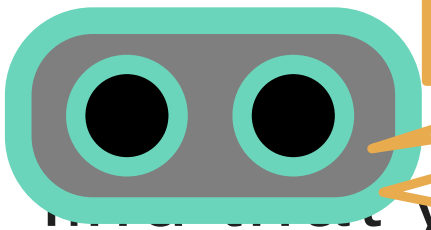
If you keep issuing books without returning, eventually library will stop issuing books to you and impose a fine

Cannot use a book after returning it (cannot use an array variable after it has been freed)

Cannot return a book you do not have (cannot free memory that has been already freed)

Of course, if you re-issue a book you can return it again

realloc



I realize that. That is why I will copy those 100 elements to the new array of 200 elements ☺

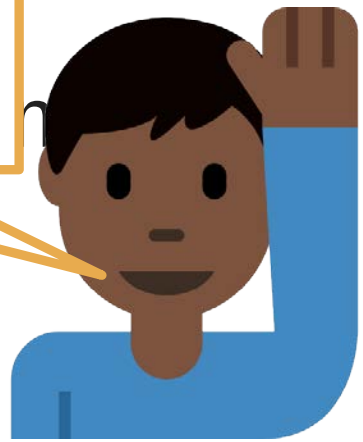
I will also free the old 100 elements – you don't have to write free() for them

If insufficient memory, I will not free old memory but just return NULL pointer

But I had so much precious data stored in those 100 elements

You are the best Mr C

A bit system-dependent. If insufficient memory, Prutor programs will simply crash



int *ptr = (int*)

Can use re

int *tmp = (in izeof(int));

if(tmp != NULL) ptr = tmp;

Don't use realloc to resize of non-malloc arrays

```
int c[100];
int *ptr = (int*)realloc(c, 200 * sizeof(int)); // Runtime error
```

Use realloc only to resize of calloc/malloc-ed arrays

getline (reading string of any length)

Read a single line of text from input (i.e. till '\n')

Uses realloc-like methods to expand array siz

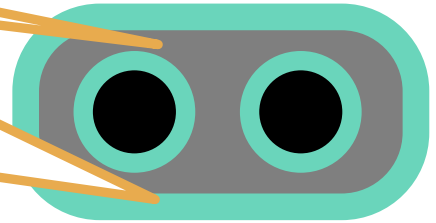
Needs a malloc-ed arra for this reason

Inception? ☺

Pointer to a pointer simply stores the address of a pointer variable

```
int len = 11; // I only expe                    ered
char *str =
getline(&st
```

printf("%ld",*ptrstr) will print address of first char in str
printf("%c",**ptrstr) will print the first char in str
printf("%s",*ptrstr) will print entire string str



If user input doesn't fit inside original array, str will contain pointer to expanded array, len will be length of new array

```
char **ptrstr = &str;
getline(ptrstr, &len, st
```

WARNING: len may be larger than length of input + 1
Get actual length of input using strlen() from string.h