Problem Solving using Pointers

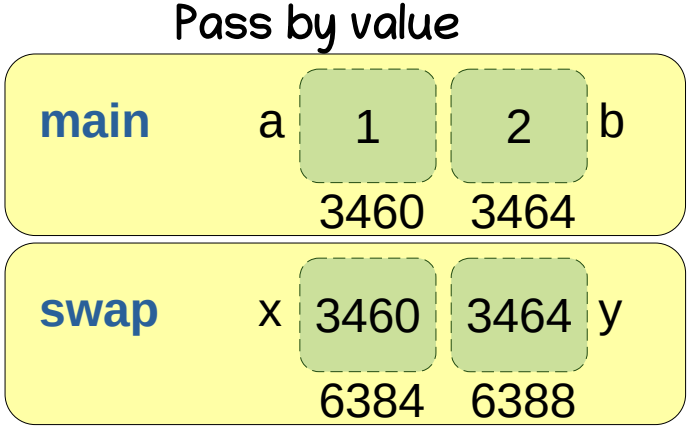
Rupesh Nasre.

Mentor TAs: Ahmed, Vivek, Vimala, Akash,
Kankan, Rahul, Swati, Akshay, Ashok, Keshav

IIT Madras
May 2022

Pointers

- Regular variables store values. Pointers' values are addresses of other variables.
- Pointers form a new (derived) data type in C.
- Pointers point to variables of a certain type.
- A pointer may contain an address which is either garbage or no longer valid.
- NULL indicates a special pointer, which is 0.
  - Similar to \0 for strings.



Pointer Dereference

```
int main() {
        int x = 1, y = 2;
        int *ptr;

        ptr = &x;
        *ptr += 10;
        printf("x=%d, y=%d\n", x, y);
```

Since s is changing, will str also change?

```
void stringlower(char *s) {
        while (*s) *s = tolower(*s), ++s;
}
int main() {
        char str[] = "Hello World.";
        char *strptr = str;

        puts(str);
        while (*strptr != '\0') {
                *strptr = toupper(*strptr);
                ++strptr;
        }
        puts(str);

        stringlower(str);
        puts(str);
}
```

Hello World.
HELLO WORLD.
hello world.

Find the behavior.

```
void fun(int *p, int *xptr) {
	*p = 10;
	*xptr = 20;
}
int main() {
	int *p, x;
	fun(p, &x);
	printf("*p = %d, x = %d\n", *p, x);
}
```

```
void fun(int **pptr, int *xptr) {
	pptr = &xptr;
	*xptr = 20;
}
int main() {
	int *p, x;
	fun(&p, &x);
	printf("*p = %d, x = %d\n", *p, x);
}
```

```
void fun(int *pptr, int *xptr) {
	pptr = xptr;
	*xptr = 20;
}
int main() {
	int *p = NULL, x;
	fun(p, &x);
	printf("*p = %d, x = %d\n", *p, x);
}
```

```
void fun(int **pptr, int *xptr) {
	*pptr = xptr;
	*xptr = 20;
}
int main() {
	int *p = NULL, x;
	fun(&p, &x);
	printf("*p = %d, x = %d\n", *p, x);
}
```

Array of Pointers

- Each entry in the array is a pointer.

sizeof(arr) == 12 or 24

```
int main() {
    int *arr[3], brr[10] = {1, 2, 3};
    int x, y;




}
```

Does this access resemble a 2D matrix?

1	2	3
4	5	6
7	8	9

int arr[3][3]

sizeof(arr) == 4 or 8

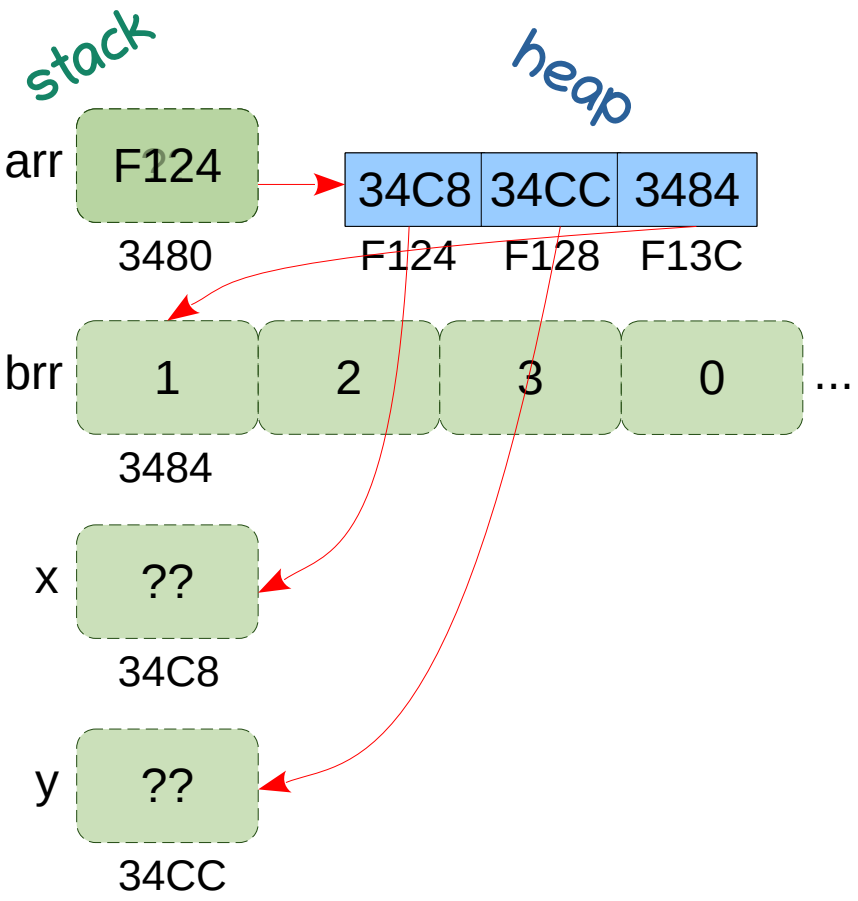
This leads to a segfault.

```
int **arr;        // pointer to a pointer

arr[0] = &x;    // type-wise, these are correct
arr[1] = &y;
arr[2] = brr;
```

```
nt main(int a, char *b[], char *c[]) {
    printf("Number of cmdline args = %d\n", a);
    for (int ii = 0; ii < a; ++ii)
        printf("\t%d: %s\n", ii, b[ii]);
    int jj = 0;
    while (c[jj] != NULL) {
        printf("%d: %s\n", jj, c[jj]);
        ++jj;
    }
}
```

Dynamic Memory Allocation




```
#include <stdlib.h>

int **arr;        // pointer to a pointer
arr = (...)malloc(3 * sizeof(int *));
arr[0] = &x;    // type-wise, these are correct
arr[1] = &y;
arr[2] = brr;
```

Write a program to populate an array that contains pointers to all zeros in another array.

Note: Every malloc should have a corresponding free.
`free(arr);`

Returning an array

- We cannot return an array from a function.
- But we can return a pointer.

```
int *giveMeAnArray() {
        int a[] = {1, 2, 3};
        return a;
}
int main() {
        int *p = giveMeAnArray();
        for (int ii = 0; ii < N; ++ii)
                printf("%d ", p[ii]);
        printf("\n");
}
```

warning: function returns address of local variable

More importantly, the program is wrong.

```
int *giveMeAnArray() {
        int *ptr = (int *)malloc(N * sizeof(int));
        for (int ii = 0; ii < N; ++ii)
                ptr[ii] = ii;
        return ptr;
}
void freeMyArray(int *ptr) {
        free(ptr);
}
```

Memory pointed to by ptr will be available after this function.

Memory pointed to by ptr will **not** be available after this statement.

Variable	Scope	Lifetime
Dynamically allocated (malloc)	Global (via pointers)	Until deallocated (free) or 100%

Pointer Arithmetic

- Pointers get incremented or decremented based on their type.

```
int main() {
        int *p, arr[] = {1, 2, 3, 4, 5, 6};
        p = &arr[0];    // same as p = arr;

        ++p;                // p points to 2
        printf("*p = %d, p[0] = %d\n", *p, p[0]);
        printf("p[2] = %d\n", p[2]);
        printf("*(p+2) = %d\n", *(p + 2));

        p += 2;
        printf("*p = %d\n", *p);
        printf("*(p-2) = %d\n", *(p - 2));
        printf("p[-2] = %d\n", p[-2]);
}
```

```
printf("*(2+p) = %d\n", *(2 + p));
printf("2[p] = %d\n", 2[p]);
printf("*(-2+p) = %d\n", *(-2+p));
printf("-2[p] = %d\n", -2[p]);
```

```
*p = 2, p[0] = 2
p[2] = 4
*(p+2) = 4
*p = 4
*(p-2) = 2
p[-2] = 2
```

Problem: Store friends' names.

```
#define N 10









int main() {
        char *names[N];         // "" indicates end of names.
        populate(names, N); // use gets.
        print(names, N);        // use puts.
}
```

You are a secret agent, and this program has names of other secret agents in your team. Encrypt and decrypt these names.

All C Keywords

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

	Week	Problems	Tools
✓	0	Solve equations, find weighted sum.	Data types, expressions, assignments
✓	1	Find max, convert marks to grade.	Conditionals, logical expressions
✓	2	Find weighted sum for all students.	Loops
✓	3	Encrypt and decrypt a secret message.	Character arrays
✓	4	Our first game: Tic-tac-toe	2D arrays
✓	5	Making game modular, reuse.	Functions
✓	6	Find Hemachandra/Fibonacci numbers.	Recursion
✓	7	Encrypt and decrypt many messages.	Dynamic memory, pointers
	8	Maintain student records.	Aggregate data types
	9	Search and sort student records.	Searching and sorting algorithms
	A	Reduce memory wastage.	Linked lists
	B	Implement token system in banks.	Queues
	C	IRCTC-like ticket booking system	File handling
	D	Putting it all together	All the above