C pointers advanced

1. Implement and test the C examples from the slides. These will familiarize you with the main issues related to the C pointers.

2. In the following example, we regard the task to perform one of the four basic arithmetic operations. The task is first solved using a switch-statement. Then it is shown, how the same can be done using a function pointer. Implement a C program and test both options. Run the program and analyse the output.

```
// The four arithmetic operations ... one of these functions is selected
// at runtime with a switch statement or a function pointer

float Plus(float a, float b) { return a + b; }
float Minus(float a, float b) { return a - b; }
float Multiply(float a, float b) { return a*b; }
float Divide(float a, float b) { return a / b; }


// Solution with a switch-statement - <opCode> specifies which operation to
execute
void Switch(float a, float b, char opCode)
{
	float result;

	// execute operation
	switch (opCode)
	{
	case '+': result = Plus(a, b); break;
	case '-': result = Minus(a, b); break;
	case '*': result = Multiply(a, b); break;
	case '/': result = Divide(a, b); break;
	}

	printf("Switch Result: %f \n", result);        // display result
}

// Solution with a function pointer - <pt2Func> is a function pointer and points to
// a function which takes two floats and returns a float. The function pointer
// "specifies" which operation shall be executed.
void Switch_With_Function_Pointer(float a, float b, float(*pt2Func)(float,
float))
{
	float result = pt2Func(a, b);    // call using function pointer

	printf("Function Pointer Result: %f \n", result);  // display result

}
```

3. Try the following code to understand Generic Pointers.

```c
#include <stdio.h>
int main()
{
	int i;
	char c;
	void *the_data;

	i = 6;
	c = 'a';

	the_data = &i;
	printf("the_data points to the integer value %d\n",
		*(int*)the_data);

	the_data = &c;
	printf("the_data now points to the character %c\n",
		*(char*)the_data);

	return 0;
}
```

4. The following C code implements a `strcpy` like function for strings. Test the code and study how the pointers arithmetic is used.

   - Using the same approach implement and test a copy function for integers having the following signature:

```c
int *intcpy_like(int *dst, const int *src, int nr);

/* Pointer to intcpy-like function */

int *(*intcpy_ptr)(int *dst, const int *src, int nr);
```

where the *nr* represent the number of integers that has to be copied from the source(src) to the destination (*dst*).

   - Following the same approach and using the generic pointer (void *) implement and test a generic copy functions for the following types: `char, int, float, double:`

```c
void *generic_cpy_like(void *dst, const void *src, char type, int nr);

/* Pointer to generic_cpy-like function */

void *(*generic_cpy_ptr)(void *dst, const void *src, char type, int nr);
```

where the *type* indicates the type of the destination and the source, and *nr* represents the number of elements that has to be copied.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* An ordinary function declaration, for reference */

char *strcpy_like(char *dst, const char *src);

/* Pointer to strcpy-like function */
```

```
char *(*strcpy_ptr)(char *dst, const char *src);

int main()
{
	char src[] = "This is a string 2INC0.";
	char *dst = (char *) malloc(strlen(src)); // allocate memory 
for the destination string

	/* Set the value of "strcpy_ptr" to be "strcpy". */
	strcpy_ptr = strcpy_like;
	/* This works too */
	strcpy_ptr = &strcpy_like;

	(*strcpy_ptr) (dst, src);
	printf("dst = %s\n", dst);
}

char * strcpy_like(char * dst, const char * src)
{
	char *ptr = dst;

	while (*dst++ = *src++);
	return ptr;
}
```

5. C structure can be accessed in 2 ways in a C program. They are,

   1. Using normal structure variable
   2. Using pointer variable

Dot(.) operator is used to access the data using normal structure variable and arrow (->) is used to access the data using pointer variable. We are showing here how to access structure data using pointer variable in below C program. In this program, “record1” is normal structure variable and “ptr” is pointer structure variable. As you know, Dot(.) operator is used to access the data using normal structure variable and the dereference pointer operator (->) is used to access data using pointer variable. Implemement and run the program analysing the output.

```
#include <stdio.h>


struct student
{
   int id;
   char name[30];
   float percentage;
};

int main()
{

   struct student record1 = { 1, "John", 90.5 };
   struct student *ptr;

   ptr = &record1;
```

```
    // access the members of hte structure using:
    // the pointer dereference operator (->) and
    // the dot(.) operator

    printf("Records of STUDENT1: \n");
    printf("  Id is: %d, %d\n", ptr->id, (*ptr).id);
    printf("  Name is: %s, %s\n", ptr->name, (*ptr).name);
    printf("  Percentage is: %f, %f\n\n", ptr->percentage, (*ptr).percentage);

    return 0;
}
```