CSE 113 Structured Programming

Pointer (part 2)

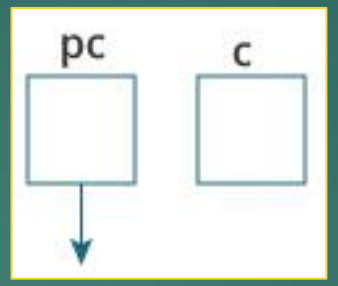
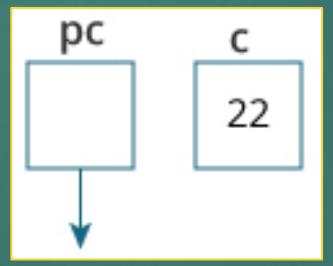
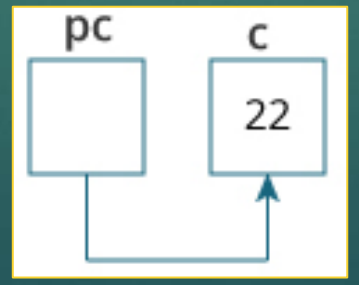
TASNIM ZAHAN

ASSISTANT PROFESSOR

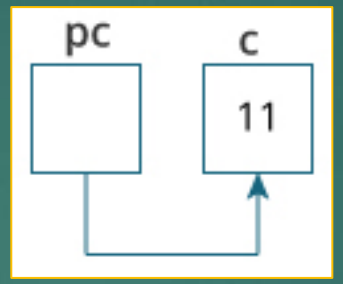
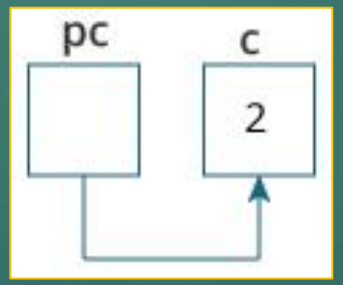
DEPT. OF COMPUTER SCIENCE & ENGINEERING

NORTH EAST UNIVERSITY BANGLADESH

Pointer review

int* pc, c;	
c = 22;	
pc = &c	

Pointer review

c = 11;	
***pc = 2;**	

Array and Pointer

```
int main() {
    int x[4];
    int i;

    for(i = 0; i < 4; ++i) {
        printf("&x[%d] = %p\n", i, &x[i]);
    }

    printf("Address of array x: %p", x);

    return 0;
}
```

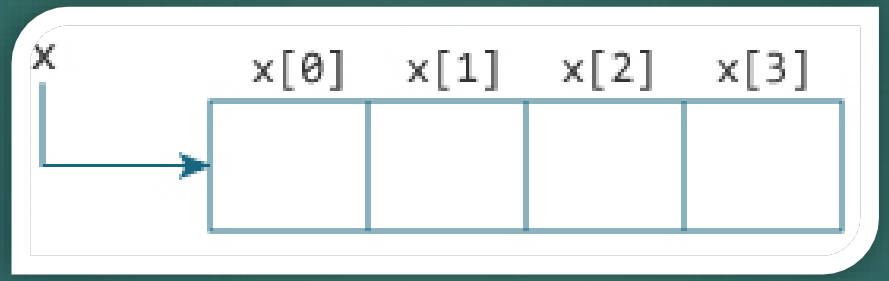
Output:

```
&x[0] = 0060FEEC
&x[1] = 0060FEF0
&x[2] = 0060FEF4
&x[3] = 0060FEF8
Address of array x: 0060FEEC
```

Notice that, the address of **&x[0]** and **x** is the same. It's because the variable name ***x points to the first element of the array.***

Array and Pointer

- &x[0] is equivalent to x [***address of array***]
- x[0] is equivalent to *x [***value of x[0]***]



Array and Pointer

```
#include <stdio.h>
int main() {
  int i, x[6], sum = 0;

  printf("Enter 6 numbers: ");

  for(i = 0; i < 6; ++i) {
  // Equivalent to scanf("%d", &x[i]);
      scanf("%d", x+i);

  // Equivalent to sum += x[i]
      sum += *(x+i);
  }
  printf("Sum = %d", sum);
  return 0;
}
```

Array and Pointer

```
#include <stdio.h>
int main() {
  int x[5] = {11, 22, 33, 44, 55};
  int* ptr;

  ptr = &x[2];

  printf("%d \n", *ptr);
  printf("%d \n", *(ptr+1));
  printf("%d", *(ptr-1))

  return 0;
}
```

```
#include <stdio.h>
int main() {
  int x[5] = {11, 22, 33, 44, 55};
  int* ptr;

  ptr = &x[1];
  *ptr += 10
  *(ptr+1) += 10

  printf("%d \n", *ptr);
  printf("%d \n", *(ptr+1));
  printf("%d", *(ptr-1))

  return 0;
}
```

Pointer and Function (Call by reference)

```c
#include <stdio.h>
void swap(int *n1, int *n2);

int main()
{
    int num1 = 5, num2 = 10;

    // address of num1 and num2 is passed
    swap( &num1, &num2);

    printf("num1 = %d\n", num1);
    printf("num2 = %d", num2);
    return 0;
}
```

```c
void swap(int* n1, int* n2)
{
    int temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
}
```

Pointer and Function

```c
#include <stdio.h>

void addOne(int* ptr) {
    (*ptr)++; // adding 1 to *ptr
}

int main()
{
    int* p, i = 10;
    p = &i;

    addOne(p);

    printf("%d", *p); // 11
    return 0;
}
```

```c
#include <stdio.h>
void addOne(int* ptr) {
    (*ptr)++;
}

int main()
{
    int* p, i = 10;
    p = &i;

    for(i=0; i<10; i++){
        addOne(p);
        printf("%d\n", *p);
    }
    return 0;
}
```

Dynamic Memory Allocation

Library functions: <stdlib.h>

- **malloc():** reserves a block of memory of the specified number of bytes.

```
ptr = (castType*) malloc(size);
```

```
ptr = (float*) malloc(100 * sizeof(float));
```

- **calloc():** allocates memory and initializes all bits to zero

```
ptr = (float*) calloc(25, sizeof(float));
```

- **realloc():** change the size of previously allocated memory

```
ptr = realloc(ptr, new_size);
```

- **free():** used to release dynamically allocated memory

```
free(ptr);
```

malloc()

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n, i, *ptr, sum = 0;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    ptr = (int*) malloc(n * sizeof(int));
    if(ptr == NULL)
    {
        printf("Error! memory not allocated.");
        exit(0);
    }
    printf("Enter elements: ");
    for(i = 0; i < n; i++)
    {
        scanf("%d", ptr + i);
        sum += *(ptr + i);
    }

    printf("Sum = %d", sum);

    // deallocating the memory
    free(ptr);

    return 0;
}
```

calloc()

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

int main()
{
    int n, i, *ptr, sum = 0;
    printf("Enter number of elements: ");
    scanf("%d", &n);

    ptr = (int*) calloc(n, sizeof(int));
    printf("Enter elements: ");
    for(i = 0; i < n; ++i)
    {
        scanf("%d", ptr + i);
        sum += *(ptr + i);
    }

    printf("Sum = %d", sum);
    free(ptr);
    return 0;
}
```

realloc()

```
int main()
{
    int *ptr, i , sz1, sz2;
    printf("Enter size: ");
    scanf("%d", &sz1);

    ptr = (int*) malloc(sz1 * sizeof(int));
    printf("Addresses of previously allocated memory:\n");
    for(i = 0; i < sz1; ++i)
         printf("%u\n",ptr + i);

    printf("\nEnter the new size: ");
    scanf("%d", &sz2);

    ptr = realloc(ptr, sz2 * sizeof(int));

    printf("Addresses of newly allocated memory:\n");
    for(i = 0; i < sz2; ++i)
         printf("%u\n", ptr + i);

    free(ptr);
    return 0;
}
```

realoc()

- Output:

```
Enter size: 2
Addresses of previously allocated memory:
26855472
26855476

Enter the new size: 5
Addresses of newly allocated memory:
26855472
26855476
26855480
26855484
```