Opakování pointerů, polí a funkcí

- **Pointer**
  - Pointer má velikost 8 / 4 bytů podle architektury
  - Přirazení pointeru
    - ```
      char c = 'a';
      char *p;
      p = &c;
      ```
  - **Dereferenční operátor ***
    - Získáme hodnotu z pointeru
  - **Referenční operátor &**
    - Získáme adresu z proměnné
  - **Pointer na pointer**
    - ```
      char **pp;
      pp = &p;
      ```
  - **Dereference pointeru na pointer**
    - ```
      char a = **pp; //a = 'c'
      ```

- **Pole**
  - ```
    char array[3] = {'a','b','c'};
    char *p = array;
    char a = p[0];   // a
    char a = *p;    // a

    char b = p[1];  // b
    char b = *(p+1); // b
    ```
  - **Řetězce**
    - ```
      char str[5] = "abc" // [a, b, c, \0, neznáma hodnota]
      ```
  - **Vícerozměrná pole**
    - ```
      float matrix [3, 2] // r - řádky, s - sloupce
      // [(0, 0), (0,1)]
      // [(1, 0), (1,1)]
      // [(2, 0), (2,1)]
      ```
    - **Reálně jsou řádky zapsané do paměti za sebou**
    - **Alokování paměti a jednorozměrná adresace**
      - ```
        float *matrix;
        matrix = malloc(r*s*sizeof(float))
        // podmínkou zjistit, jestli se paměť vytvořila
        for (řádky r){
            for (sloupce s){
                matrix[r*S + s]          // S - počet sloupců
            }
        }
        free (matrix);
        ```
    - **Alokování paměti a vícerozměrná adresace (neodporučeno)**
        ```

```
float **matrix;
matrix = malloc(R*sizeof(float*))   // R – počet řádků
for (R){
    *(matrix + r) = malloc(S*sizeof(float))
}
// Kontrola, jestli malloc vyšel
// Strašně moc příkazů free
```

- **Funkce**
  - **Vstupy a výstupy**

```
int x = 10;
int y = 11;
void fn (int z, int *p){
    z = 20;
    *p = 30;
}
fn(x, &y);
// x = 10;
// y = 30;
```

    - Používaní ukazatele ve funkci

```
int *p;
void fn (int **pa){
    *pa = malloc(sizeof(int));
    **pa = 10;
}
fn(&p);
//*p = 10;
free(p);
```