Realizace vícerozměrných polí pomocí pole ukazatelů
double *Pole2b[2];

- v paměti zabírá lineární prostor ukazatelů (na pole doublů, které je nutné vytvořit (definovat pole, naalokovat) a přiřadit – tato pole zabírají další lineární prostory v paměti, které na sebe obecně nenavazují)
- první index vybere ukazatel, který je oindexován druhým indexem (vybere prvek v poli)
- pole má pevný počet řádků. Počet prvků v řádku je libovolný, pro jednotlivé řádky může být různý

Pozn.: následující ukázky fungují v místě definice proměnné a ve funkcích se stejným prototypem. Při předání do funkce jako jinak definované vícerozměrné pole dojde k chybě.

```
double Pole1[3] = {1,2,3}, *Pole2;
double *Pole2b[2];
Pole2 = (double*) malloc(5*sizeof(double));
Pole2[0]=4;
Pole2[1]=5;
Pole2[2]=6;
Pole2[3]=7;
Pole2[4]=8;
Pole2b[0] = Pole1;
Pole2b[1]= Pole2;
Pole2b[0][0] = *(*(Pole2b+1)+1); // oindexovat ukazatel a přistoupit. První výsledek
// je ukazatel, druhý double
if (Pole2) free(Pole2);
```

Nakreslete paměťovou mapu.

Realizace pomocí ukazatelů na pole

double (*Pole2c)[2];

- v paměti zabírá prostor na jeden ukazatel. Do toho je nutné naalokovat lineární prostor pro celé pole
- první index vybírá řádek (vypočte ho překladač díky znalosti délky řádku plynoucí z druhého indexu), druhý index vybere prvek v řádku.
- pole má pevný počet prvků v řádku. Může mít libovolný počet řádků.

```
double pole [2][3]={1,2,3,4,5,6};
double (*pp)[2]; // definice bez inicializace

double (*pp)[3]=(double(*)[3])pole; // definice s inicializací použití existujícího pole

double (*pp)[2] = NULL; // definice „bez inicializace“ na „rozumnou hodnotu“
pp = (double(*)[3]) malloc (2*3*sizeof(double));
// „inicializace“ pomocí přiřazení - alokace nového pole

pp[0][0] = 0;
pp[0][1] = 01;
pp[0][2] = 2;
pp[1][0] = 10;
pp[1][1] = 11;
*(*(pp+1)+1) = 11; // předchozí řádek pomocí ukazatelů
pp[1][2] = 12;
if (pp) free (pp);
```

Nakreslete paměťovou mapu.

Pole ve funkci a volání

```
void funkce_správně(double(*pp)[2])
{
	pp[1][1] = 11;
	*(*(pp + 1) + 1) = 11; // předchozí řádek pomocí ukazatelů
}

void funkce_spatne(double *pp[2])
{
	pp[1][1] = 11;
	*(*(pp + 1) + 1) = 11; // předchozí řádek pomocí ukazatelů
}
// část kódu s voláním předchozích funkcí
double(*pp)[2] = NULL;
pp = (double(*)[3]) malloc(2 * 3 * sizeof(double));
funkce_správně(pp);
funkce_spatne(pp);
if (pp) free(pp);
```

Realizace pomocí ukazatele na ukazatel

double **Pole2d

- v paměti zabírá místo na jeden ukazatel. Je nutné naalokovat (a sem uložit) pole ukazatelů (lineární prostor – lze si ho představit jako vertikální pole, každý ukazatel pak ukazuje na "řádek" matice). Do každého ukazatele je nutné naalokovat řádek prvků (double)
- první index vybere ukazatel v poli ukazatelů (ukazující na řádek), druhý index oindexuje řádek (vybere prvek v daném řádku)
- počet řádků může být proměnný. Řádky mohou být různě dlouhé, lze je měnit.

```
double **Pole2d=NULL;
Pole2d = (double **) malloc( 2 * sizeof(double*));
if(Pole2d == NULL) return chyba;

Pole2d[0]=(double*) malloc(3*sizeof(double*));
Pole2d[1]=(double*) malloc(6*sizeof(double*));

if(Pole2d [0]== NULL)
   {free(Pole2d); return chyba ;}
if(Pole2d [1]== NULL)
      { free(Pole2d[0]);
       free(Pole2d); return  chyba;}
Pole2d[1] [0]=4; Pole2d[1] [1]=5;Pole2d[1] [2]=6;
Pole2d[1] [3]=7; Pole2d[1] [4]=8; Pole2d[1] [5]=8;
Pole2d[0] [0]=*(*(Pole2d+1)+1);
Pole2d[0] [1]=2;      Pole2d[0] [2]=3;
free(Pole2d[1]);
free(Pole2d[0]);
free(Pole2d);   Pole2d = NULL;
```

Na základě předešlého příkladu (Nakreslete si jeho paměťovou mapu.) napište kód na vytvoření obdélníkového pole (nejdříve v main, potom přesuňte do funkcí). Napište části kódu pro alokaci a uvolnění pole daných rozměrů (ve funkci s rozměry jako parametrem, pro práci použijte cykly). Proměnné vytvořeného pole inicializujte na nulu.
Nakreslete paměťovou mapu.