# Řešení sady 2

Úvod do programování 2

Tomáš Kühr

# Fibonacciho čísla

```
long FibRek(unsigned int n) {
    if (n <= 1) return n;
    return FibRek(n - 2) + FibRek(n - 1);
}

long FibIter(unsigned int n){
    if (n <= 1) return n;
    else {
        long predPred = 0;
        long pred = 1;
        while (n-1 > 0){
            long aktual = predPred + pred;
            predPred = pred;
            pred = aktual;
            n--;
        }
    return pred;
    }
}
```

# Půlení intervalu

```
int puleni(int cisla[], int a, int b, int hledane){
    if (a > b) return -1; // nenalezeno
    else {
        int stred = (a + b) / 2;
        if (cisla[stred] == hledane) return stred;
        else if (cisla[stred] > hledane)
            return puleni(cisla, a, stred-1, hledane);
        else
            return puleni(cisla, stred+1, b, hledane);
    }
}
```

# Mapování

```
double *map(double(*fce)(double), double *vstup, int delka){
   int i;
   double *vysledek;
   vysledek = (double *)malloc(delka * sizeof(double));
   if (vysledek != NULL){
      for (i = 0; i < delka; i++){
         vysledek[i] = fce(vstup[i]);
      }
   }

   return vysledek;
}
```

# Mapování pole funkcí

```
double **map(double(*fce[])(double), double *vstup, int pocet, int delka){
   double **vysledek;
   int i, j, l;
   vysledek = (double **)malloc((pocet+1) *sizeof(double *));
   if (vysledek == NULL) return NULL;
   for (i = 0; i <= pocet; i++){
      vysledek[i] = (double *)malloc(delka * sizeof(double));
      if (vysledek[i] == NULL){ // uklid
         for (l = 0; l < i; l++) free(vysledek[l]);
         free(vysledek); return NULL;
      }
      for (j = 0; j < delka; j++){
         if (i == 0) vysledek[i][j] = vstup[j];
         else vysledek[i][j] = fce[i - 1](vstup[j]);
      }
   }
return vysledek;
}
```

# Akumulátor

```
double akumulator(double(*fce)(double, double),
                  double cisla[], int pocet){
   int i;
   double vysledek = cisla[0];
   for (i = 1; i < pocet; i++){
      vysledek = fce(vysledek, cisla[i]);
   }
   return vysledek;
}
```

# Průměr

```
long double prumer(char* format, ...){
    int pocet = 0;
    long double vysledek = 0.0;
    va_list zasobnik;
    va_start(zasobnik, format);
    while (format[pocet]){
        switch (format[pocet]){
            case 'i': vysledek += va_arg(zasobnik, int); break;
            case 'd': vysledek += va_arg(zasobnik, double); break;
            case 'l': vysledek += va_arg(zasobnik, long double); break;
        }
        pocet++;
    }
    va_end(zasobnik);
    return vysledek/pocet;
}
```

# Suma komplexních čísel

```
typedef struct {
    double realna;
    double imaginarni;
} komplexni;

komplexni suma(int pocet, ...){
    int i;
    komplexni vysledek = { 0.0, 0.0 };
    va_list zasobnik;
    va_start(zasobnik, pocet);
    for (i = 0; i < pocet; i++){
        komplexni data= va_arg(zasobnik, komplexni);
        vysledek.realna += data.realna;
        vysledek.imaginarni += data.imaginarni;
    }
    va_end(zasobnik);
    return vysledek;
}
```