Programownie I
Wykład 5

dr inż. Adam Zielonka

Instytut Matematyki,
Politechnika Śląska

Gliwice 16.11.2018

Średnia arytmetyczna i średnia ważona

Niech $x_1, x_2, \ldots, x_n$ będą dowolnymi liczbami.

Średnią arytmetyczną

nazywamy liczbę $\overline{x}$:

$$\overline{x} = \frac{x_1 + x_2 + \ldots + x_n}{n} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Ponadto, niech liczby $w_1, w_2, \ldots, w_n$ będą dodatnimi wagami odpowiadającymi powyższym liczbom.

Średnią ważoną

nazywamy liczbę $\overline{x}_w$:

$$\overline{x}_w = \frac{w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n}{w_1 + w_2 + \ldots + w_n} = \sum_{i=1}^{n} \frac{w_i \cdot x_i}{w_i}$$

Wariancja i odchylenie standardowe

Wariancją

nazywamy liczbę σ^2:

$$\sigma^2 = \frac{(x_1 - \overline{x})^2 + (x_2 - \overline{x})^2 + \ldots + (x_n - \overline{x})^2}{n} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

Odchyleniem standardowym

nazywamy liczbę σ:

$$\sigma = \sqrt{\frac{(x_1 - \overline{x})^2 + (x_2 - \overline{x})^2 + \ldots + (x_n - \overline{x})^2}{n}} = \sqrt{\sigma^2}$$

Przykład

Niech dane będą liczby $3, 7, 6, 9, 15$ oraz wagi:$1, 3, 3, 3, 10$

$$\begin{aligned}
\overline{x} &= \frac{3+7+6+9+15}{5} = \frac{40}{5} = 8 \\
\overline{x}_w &= \frac{1\cdot 3+3\cdot 7+3\cdot 6+3\cdot 9+10\cdot 15}{1+3+3+3+10} = \frac{219}{20} = 10.95 \\
\sigma^2 &= \frac{(3-8)^2+(7-8)^2+(6-8)^2+(9-8)^2+(15-8)^2}{5} \\
&= \frac{25+1+4+1+49}{5} = \frac{80}{5} = 16 \\
\sigma &= \sqrt{16} = 4.
\end{aligned}$$

Implementacja

Średnia arytmetyczna

```
double sredniaAryt(double x[], int n)
{
    double srednia = 0;
    for (int i = 0; i < n; i++)
         srednia += x[i];
    return srednia / n;
}
int main()
{
    double x[]={3, 7, 6, 9, 15};
    int n=5;
    double srednia = sredniaAryt (x , n );
    cout << "srednia =" << srednia << endl;
    system("PAUSE");
}
```

Implementacja

Średnia ważona

```
double sredniaWazona(double x[], double w[], int n)
{
    double suma = 0;
    double sumaWag = 0;
    for (int i = 0; i < n; i++)
    {
        suma += x[i] * w[i];
        sumaWag += w[i];
    }
    return suma / sumaWag;
}
```

Implementacja

Średnia ważona

```
int main()
{
    double x[]={3, 7, 6, 9, 15};
    double w[]={1, 3, 3, 3, 10};
    int n=5;
    double srednia = sredniaWazona (x , w , n );
    cout << "srednia wazona =" << srednia << endl;
    system("PAUSE");
}
```

Implementacja

Wariancja

```
double wariancja(double x[], int n)
{
    double suma {};
    double srednia = sredniaArytmetyczna(x, n);
    for (int i = 0; i < n; i++)
        suma +=((x[i] - srednia)*(x[i]-srednia));
    return suma / n;
}
```

Odchylenie standardowe

```
#include<cmath>
double odchylenieStandardowe(double x[], int n)
{
    return sqrt(wariancja(x,n));
}
```

Wskaźniki

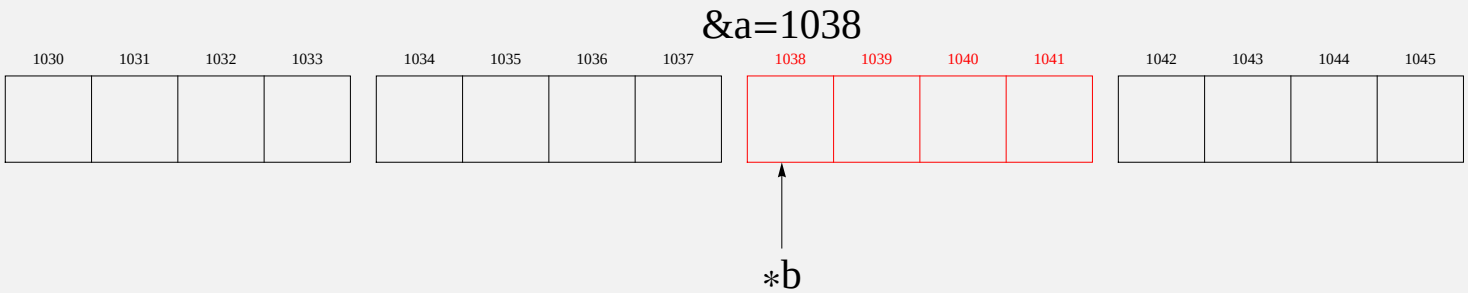
```
int a=453;//zajmuje 4 kolejne bajty pamięci
```

&a adres pierwszej komórki pamięci, w której "przechowywana" jest wartość zmiennej a

Deklaracja wskaźnika

```
int *b = nullptr; //wskaźnik do zmiennej typu int
```

```
*b=&a; //b wskazuje na zmienną a
```



Wskaźniki

```
int a=5,b=3,c[10]{};

int *w=nullptr;

w=&a;  //wskazuje na a

b=*w;  //wyłuskanie wartości spod adresu w

*w=7;  //zmiana wartości pod adresem w na 7

w=&c[2]; //ustawienie wskaźnika na c[2]
```

Wskaźniki

Zamiana

```
...
void zamiana(int *a, int *b)
{
    int z=*a;
    *a=*b;
    *b=z;
}
int main()
{
    int a=11,b=12;
    cout << "a=" << a << " b=" << b << endl;
    zamiana( &a, &b );
    cout << "a=" << a << " b=" << b << endl;
}
```

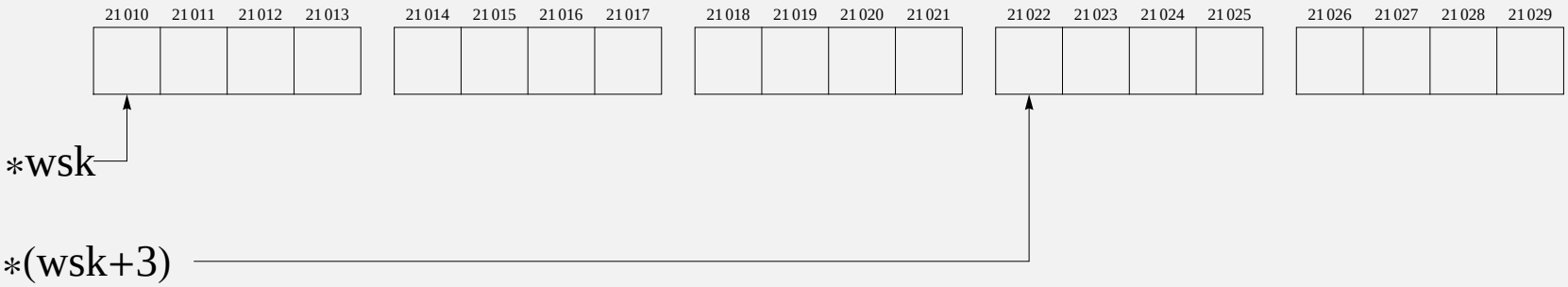
Wskaźniki a tablice

```
int tablica[] = {8, 8, 6, -9, 12};
//wskaznik do tablicy
int *wsk = &tablica[0];

cout << *wsk << endl;

cout << *(wsk+1) << endl;

cout << tablica[2] << endl;
```

`*(wsk+n)` oznacza wyłuskanie elementy typu wskaźnika znajdującego się pod adresem `wsk+n*sizeof(typ)`
W przypadku tablicy zapis ten jest równoważny `tablica[n]`

Wskaźniki a tablice

```
int tablica[5]  {};
//wskaznik do tablicy
int *wsk = &tablica[0];
```



Wskaźniki a tablice

Nazwa tablicy jest wskaźnikiem, który nie może zmieniać wartości, czyli zapamiętanego w nim adresu. Zatem zamiast:

```
int tab[] = { 10, 2, 3, 4 };
int *wsk = &tab[0];
```

można wywołać:

```
int tab[] = { 10, 2, 3, 4 };
int *wsk = tab;
```

Wskaźniki

```
int tab[] = { 10, 2, 3, 4 };
int *wsk = tab;

wsk++; //ustawienie wskaźnika na adres kolejnej
       //zmiennej typu int, czyli na wyraz tab[1]

//tab++; // błąd, ten wskaźnik nie może ulec zmianie

cout << *(wsk+1); //wypisze 3

cout << *(tab+1); //wypisze 2

cout << *++wsk;
```

Przekazywanie tablic do funkcji

```
void drukuj( int *tab, int n )
{
    for(int i=0 ; i < n ; i++ )
        cout << tab[i] << ' ';

    cout << endl;
}
```

```
int dlugosc( char *s )
{
    int n=0;
    while( *s++ ) n++;
    return n;
}
```

Przykład

```
void sortuj(int *tab1, int n)
{
    int *tab = tab1;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (*tab > *(tab + 1))
                zamiana(tab, tab + 1);
            tab++;
        }
        tab = tab1;
    }
}
```

Reprezentacja binarna liczby całkowitej nieujemnej

Niech $x \in N \cup \{0\}$. Reprezentacją binarną tej liczby w postaci n-bitowego słowa nazywamy ciąg: $a_{n-1}, \ldots, a_1, a_0$ taki, że;

$$x = a_{n-1}2^{n-1} + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0,$$

gdzie $a_i \in \{0, 1\}$.

Przykład

Dla $x = 63$ i $n = 8$ mamy:

$$56 = 2^5 + 2^4 + 2^3$$

Zatem 8-bitowe słowo reprezentujące liczbę 56 ma postać:

$$00111000$$

Operatory bitowe

OR – alternatywa bitowa:

```
unsigned char a=17, b=9, c;
c=a | b; \\ c=25
```

```
a  00010001
b  00001001
c  00011001
```

AND – koniunkcja bitowa:

```
unsigned char a=17, b=9, c;
c = a & b; \\ c=1
```

```
a  00010001
b  00001001
c  00000001
```

Operatory bitowe

XOR – bitowa alternatywa wykluczająca:

```
unsigned char a=17, b=9, c;
c=a ^ b; \\ c=24
```

a	00010001
b	00001001
c	00011000

NOT – negacja:

```
unsigned char a=17, c;
c = ~ a; \\ c=238
```

a	00010001
c	11101110

Operatory bitowe

<< przesunięcie bitowe w lewo

```
unsigned char a=17, b{};
c = a << 2; \\ c=68
```

a 00010001
b 01000100

>> przesunięcie bitowe w prawo

```
unsigned char a=49, b{};
b = a >> 3; \\ b=6
```

a 00110001
c 00000110

Operatory bitowe

Operatory przypisania

```
unsigned char a{};
a|=2;   // a=a|2;
a&=3;   // a=a&3;
a^=2;   // a=a^2;
a<<=2;  // a=a<<2;
a>>=2;  // a=a>>2;
```

Przykład

```
unsigned char a=1;
for(int i=1; i<8 ;i++)
    cout << ( a << i ) << endl;
```

Przykłady

Zapalanie wybranego bitu

```
unsigned char zmienna{};
...
zmienna | = 1; // zapalenie pierwszego bitu
zmienna | = 2; // zapalenie drugiego bitu
zmienna | = 4; // zapalenie trzeciego bitu
```

Wygaszenie / przełączenie wybranego bitu

```
zmienna & = ~1  //zgaszenie pierwszego bitu
zmienna ^ = 4   //przełączenie trzeciego bitu
```

?

```
int n=13241;
cout << (n & 1 ? 'a' : 'b');
```

Przykłady

Drukowanie postaci binarnej

```
void drukujBinarnie(int n)
{
    int bity[32] {};
    int* i = &bity[31];
    for (int j = 0; j<32 ;j++)
    {
        *i-- = ( n >> j ) % 2 ? 1 : 0 ;
    }
    i = bity;
    for (int j = 0; j < 32; j++)
        cout << *i++;
    cout << endl;
}
```

Metoda dopełnienia dwójkowego

Niech $x \in Z$. Reprezentacją binarną tej liczby w postaci n-bitowego słowa nazywamy ciąg: $a_{n-1}, \ldots, a_1, a_0$ taki, że;

$$x = -a_{n-1}2^{n-1} + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0,$$

gdzie $a_i \in \{0, 1\}$.

Przykład

Dla $x = -3$ i $n = 8$ mamy:

$$-3 = -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2 * 0$$

Zatem 8-bitowe słowo reprezentujące liczbę -3 ma postać:

$$11111101$$

Metoda dopełnienia dwójkowego

Dla słów 8-bitowych

00000000 - 01111111 liczby od 0 do 127

10000000 kolejna po dodaniu 1 wynosi -128

10000001 kolejna po dodaniu 1 wynosi -127

10000010 kolejna po dodaniu 1 wynosi -126

...

11111110 reprezentuje liczbę -2

11111111 kolejna po dodaniu 1 wynosi -1

00000000 następna po dodaniu 1 wynosi 0