

PODSTAWY PROGRAMOWANIA

dr hab. Mariusz Mészka

Akademia Górniczo-Hutnicza w Krakowie

<http://home.agh.edu.pl/~meszka>

Program wykładu

1. Podstawowe pojęcia i definicje. Metody zapisu algorytmów.
2. Język programowania C: wprowadzenie, typy i rozmiary danych, operatory i wyrażenia.
3. Język programowania C: składnia języka – instrukcje, struktura programu.
4. Język programowania C: funkcje, preprocesor, klasy pamięci.
5. Język programowania C: operacje na wskaźnikach i adresach, tablice dynamiczne.
6. Język programowania C: wejście i wyjście, pliki.
7. Język programowania C: podstawowe biblioteki i narzędzia, przetwarzanie kodu źródłowego w różnych systemach operacyjnych.

Program wykładu (cd.)

8. Maszynowa reprezentacja informacji (systemy zapisu liczb i arytmetyka).
9. Elementy teorii złożoności algorytmów: notacja, złożoność pesymistyczna i średnia, klasyfikacja problemów.
10. Rekurencja i iteracja – porównanie metod.
11. Wybrane algorytmy sortowania: proste wstawianie, proste wybieranie, bąbelkowe, szybkie, scalanie, stogowe.
12. Sortowanie w czasie liniowym: przez zliczanie i kubełkowe. Wyszukiwanie mediany.
13. Złożone struktury danych: stosy, kolejki, listy pojedynczo związane.
14. Listy podwójnie i wielokrotnie związane.
15. Drzewa – implementacje, porządki w drzewach. Drzewa przeszukiwań binarnych.

Literatura

- [1] B. Kernighan, D.M. Ritchie, Język ANSI C. Programowanie. Wydanie II , Helion 2020.
- [2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Wprowadzenie do algorytmów, PWN 2012.
- [3] N. Wirth, Algorytmy + struktury danych = programy, WNT 2002.

Podstawowe obiekty danych

- (1) stałe
- (2) zmienne

Rodzaje stałych

- (1) liczbowe
- (2) znakowe
- (3) tekstowe

Typy danych

- (1) proste
 - (a) liczbowe
 - (i) całkowite: **int**
 - (ii) zmiennopozycyjne: **float, double**
 - (b) znakowe: **char**
 - (c) logiczne
 - (d) wyliczeniowe: **enum**
- (2) złożone
 - (a) tablice
 - (b) struktury
 - (c) unie

Modyfikatory

short, long, signed, unsigned

Definicja stałej

const *typ nazwastałej* = *wartość*;

Deklaracja zmiennej

typ nazwazmiennej;

typ nazwazmiennej = *wartość*;

Operatory

(1) arytmetyczne:

+ - * / % ++ --

(2) relacyjne:

< > <= >= == !=

(3) logiczne

&& || !

(4) przypisania

= *operator* =

(5) bitowe

& | ^ << >> ~

(6) rozmiaru

sizeof(*zmienna*)

sizeof(*typ*)

(7) warunkowy

wyrażenie1 ? *wyrażenie2* : *wyrażenie3*

Instrukcje warunkowe

(1)

```
if (warunek)  
    instrukcja;
```

(2)

```
if (warunek)  
    instrukcja1;  
else  
    instrukcja2;
```

Instrukcje warunkowe cd.

(3)

```
switch (wyrażenie)
{
    case przypadek1: instrukcja1;
        break;
    case przypadek2: instrukcja2;
        break;
    :
    case przypadekN: instrukcjaN;
        break;
    default: instrukcja;
}
```

Instrukcje iteracyjne

(1)

```
while (warunek)  
    instrukcja;
```

(2)

```
do  
    instrukcja;  
while (warunek);
```

(3)

```
for (wyrażenie1; wyrażenie2; wyrażenie3)  
    instrukcja;
```

Instrukcje skoku

(1)

break;

(2)

continue;

(3)

return *wyrażenie*;

(4)

goto *etykieta*;

Blok instrukcji

```
{  
    instrukcja1;  
    instrukcja2;  
    ⋮  
    instrukcjaN;  
}
```

Instrukcja pusta

```
;
```

Funkcja to zamknięty w całość fragment kodu programu, który jest wykonywany po jego jawnym wywołaniu. Do funkcji można przekazywać parametry, funkcja może zwracać wartość.

Definicja funkcji

```
typzwracanejwartości nazwafunkcji(deklaracjeparametrów)  
{  
  ciąg deklaracji i instrukcji  
  return wyrażenie;  
}
```

Struktura programu

dyrektywy preprocesora

definicje i deklaracje obiektów zewnętrznych

definicje funkcji

Dyrektywy preprocesora

#include <*nazwapliku*>

#include "*nazwapliku*"

#define *identyfikator* *ciągznaków*

#undef *identyfikator*

#if *wyrażenie1* ... **#endif**

Standardowe wejście i wyjście

int getchar(void)

int putchar(int)

int printf(char *format, *argument1*, *argument2*, ...)

int scanf(char *format, *wskaźnik1*, *wskaźnik2*, ...)

format:

%<flaga><szerokość><.precyzja><modyfikator>typ

typ:

d,i **int**
u **unsigned int**
o **int** w formacie ósemkowym
x **int** w formacie szesnastkowym
o **int** w formacie ósemkowym
c **char**
s **char***
f,e **double**

Wskaźniki i adresy

Wskaźnik to zmienna, która zawiera adres innej zmiennej.

(1) operator referencji zwracający adres zmiennej:

& zmienna

(2) operator deferencji, którego argumentem jest adres:

** wskaźnik*

Alokacja pamięci

void *malloc(size_t *liczba bajtów*)

void *calloc(size_t *liczba elementów*, size_t *rozmiar*)

void free(void **wskaźnik*)

Pliki

```
FILE *wskaźnikplikowy;
```

```
FILE *fopen(char *nazwapliku, char *tryb)
```

```
int fclose(FILE *wskaźnikplikowy)
```

```
int main(int argc, char * argv[])
```

Złożoność czasowa pesymistyczna algorytmu

$$T(n) = \max\{t(d) : d \in \mathcal{D}_n\}$$

przy oznaczeniach:

n - rozmiar danych wejściowych

$\mathcal{D}_n$ - zbiór danych wejściowych rozmiaru n

$t(d)$ - czas wykonania obliczeń dla zestawu danych wejściowych d ,
wyrażony liczbą operacji elementarnych lub dominujących

Złożoność czasowa średnia algorytmu

$$A(n) = \sum_{d \in \mathcal{D}_n} p(d) t(d)$$

gdzie:

$p(d)$ - prawdopodobieństwo z jakim zestaw danych d może
pojawić się na wejściu

Złożoność pamięciowa algorytmu

$$S(n) = \max\{s(d) : d \in \mathcal{D}_n\}$$

gdzie:

$s(d)$ - liczba komórek pamięci wykorzystywanych podczas obliczeń dla zestawu danych wejściowych d

Sortowanie elementów w tablicy

Wejście: Tablica T zawierająca n liczb.

Wyjście: Rozmieszczenie elementów w T w porządku niemalejącym.

```
void proste_wstawianie(int n,int *T)
```

```
{  
    int i,j,x;  
    for (i=1;i<n;i++)  
        {  
            x=T[i];  
            j=i-1;  
            while (j>=0 && x<T[j])  
                {  
                    T[j+1]=T[j];  
                    j--;  
                }  
            T[j+1]=x;  
        }  
}
```

$$T(n) = \Theta(n^2)$$

```
void wstawianie_polowkowe(int n,int *T)
```

```
{  
    int i,j,l,m,p,x;  
    for (i=1;i<n;i++)  
        {  
            x=T[i];  
            l=0;  
            p=i-1;  
            while (l<=p)  
                {  
                    m=(l+p)/2;  
                    if (x<T[m])  
                        p=m-1;  
                    else  
                        l=m+1;  
                }  
            for (j=i-1;j>=l;j--)  
                T[j+1]=T[j];  
            T[l]=x;  
        }  
}
```

$$T(n) = \Theta(n^2)$$

```
void proste_wybieranie(int n,int *T)
```

```
{  
    int i,j,k,x;  
    for (i=0;i<n-1;i++)  
        {  
            x=T[i];  
            k=i;  
            for (j=i+1;j<n;j++)  
                if (T[j]<x)  
                    {  
                        x=T[j];  
                        k=j;  
                    }  
            T[k]=T[i];  
            T[i]=x;  
        }  
}
```

$$T(n) = \Theta(n^2)$$

```
void sortowanie_babelkowe(int n,int *T)
```

```
{  
    int i,j,x;  
    for (i=1;i<n;i++)  
        {  
            for (j=n-1;j>=i;j--)  
                if (T[j-1]>T[j])  
                    {  
                        x=T[j];  
                        T[j]=T[j-1];  
                        T[j-1]=x;  
                    }  
        }  
}
```

$$T(n) = \Theta(n^2)$$

```
void scalanie(int n,int *T,int *S,int l,int p)
```

```
{  
    int i,j,k,m;  
    if (l==p)  
        return;  
    m=(l+p)/2;  
    scalanie(n,T,S,l,m);  
    scalanie(n,T,S,m+1,p);  
    i=l;  
    j=m+1;  
    k=l;  
    while (i<=m && j<=p)  
        if (T[i]<=T[j])  
            S[k++]=T[i++];  
        else  
            S[k++]=T[j++];  
    while (i<=m)  
        S[k++]=T[i++];  
    while (j<=p)  
        S[k++]=T[j++];  
    for (i=l;i<=p;i++)  
        T[i]=S[i];  
}
```

```
void sortowanie_przez_scalanie(int n,int *T)
```

```
{  
    int S[n];  
    scalanie(n,T,S,0,n-1);  
}
```

$$T(n) = \Theta(n \log n)$$

```
void przesiewanie(int n,int *T,int l,int p)
```

```
{  
    int i,j,x;  
    i=l;  
    j=2*i+1;  
    x=T[i];  
    while (j<=p)  
    {  
        if (j<p && T[j]<T[j+1])  
            j=j+1;  
        if (x>=T[j])  
            break;  
        T[i]=T[j];  
        i=j;  
        j=2*i+1;  
    }  
    T[i]=x;  
}
```

```
void buduj_kopiec(int n,int *T)
```

```
{  
    int l;  
    for (l=n/2-1;l>=0;l--)  
        przesiewanie(n,T,l,n-1);  
}
```

```
void sortowanie_stogowe(int n,int *T)
```

```
{  
    int p,x;  
    buduj_kopiec(n,T);  
    for (p=n-1;p>0;p--)  
        {  
            x=T[0];  
            T[0]=T[p];  
            T[p]=x;  
            przesiewanie(n,T,0,p-1);  
        }  
}
```

$$T(n) = O(n \log n)$$

```
void sortuj(int n,int *T,int l,int p)
```

```
{
    int i,j,x,y;
    i=l;
    j=p;
    x=T[(l+p)/2];
    while (i<=j)
    {
        while T[i]<x)
            i++;
        while T[j]>x)
            j--;
        if (i<=j)
        {
            y=T[i];
            T[i]=T[j];
            T[j]=y;
            i++;
            j--;
        }
    }
    if (l<j)
        sortuj(n,T,l,j)
    if (i<p)
        sortuj(n,T,i,p)
}
```

```
void sortowanie_szybkie(int n,int *T)
```

```
{  
    sortuj(n,T,0,n-1);  
}
```

$$T(n) = \Theta(n^2)$$

$$A(n) = \Theta(n \log n)$$