Instytut Matematyki i Informatyki
Wydział Podstawowych Problemów Techniki
Politechnika Wrocławska

Wstęp do Informatyki i Programowania

Jacek Cichoń

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

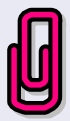
Spis treści I

Spis treści II

Spis treści III

Materiały

Literatura

1. J. Cichoń, P. Kobylański, *Notatki ze Wstępu do Informatyki*, strona WWW kursu
2. B. W. Kernighan, D. M. Ritchie, *Język ANSI C*, Wydawnictwa Naukowo-Techniczne, Warszawa, 2003
3. A. Aho, J. Ullman, *Wykłady z informatyki z przykładami w języku C*, Helion, 2003
4. D. Harrell, *Rzecz o istocie informatyki. Algorytmika*, Państwowe Wydawnictwo Naukowe, Warszawa, 2001
5. *Ściąga z języka C*: 

Zaczynamy

Język C

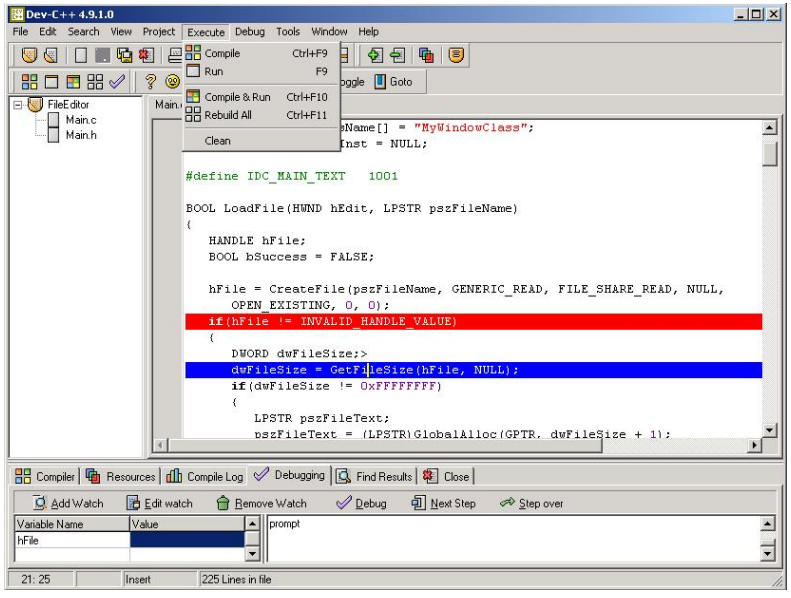
Na wykładzie będziemy programować w języku C.

Środowisko Dev C++

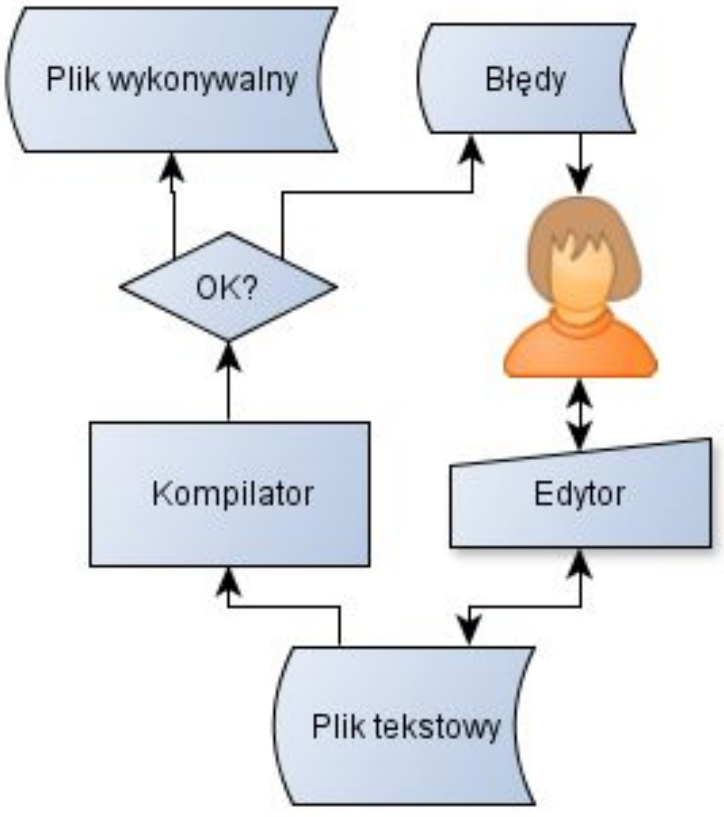
Znajdziesz je na stronie:

http://www.bloodshed.net/devcpp.html

Po załadowaniu na swój komputer pliku devcpp_setup.exe uruchom go i stosuj wszystkie domyślne opcje.



IDE



- programy języka C zapisujemy w plikach tekstowych
- kompilator tłumaczy plik tekstowy na kod maszynowy
- Zintegrowane środowisko programistyczne (Integrated Development Environment, IDE): aplikacja lub zespół aplikacji służących do tworzenia, modyfikowania, testowania i konserwacji oprogramowania.

Hello World - 1

Nasz pierwszy program

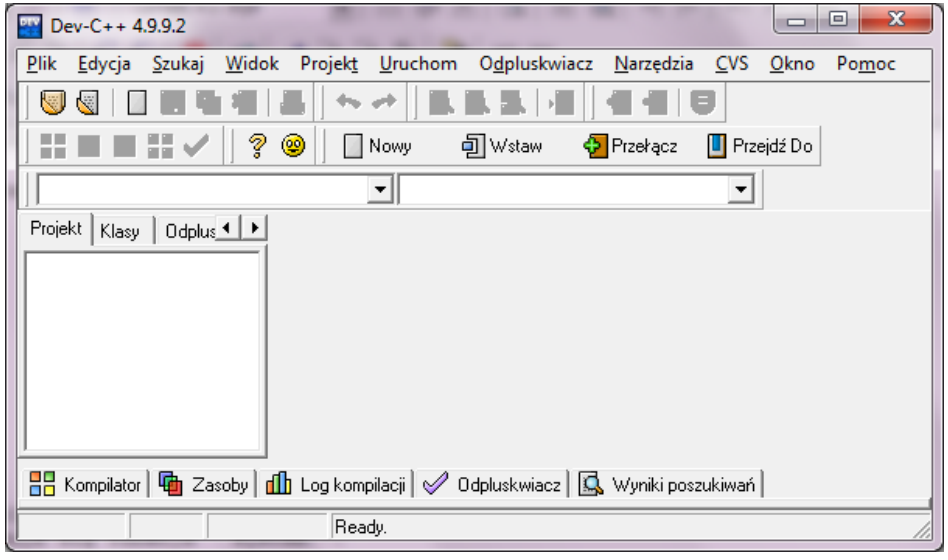
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   printf("Hello world !\n");
   system("PAUSE");
   return 0;
}
```

W bibliotece stdio znajdują sie funkcje “printf” i “scanf”. W bibliotece stdlib znajduje się funkcja “system”.

Hello World - 2

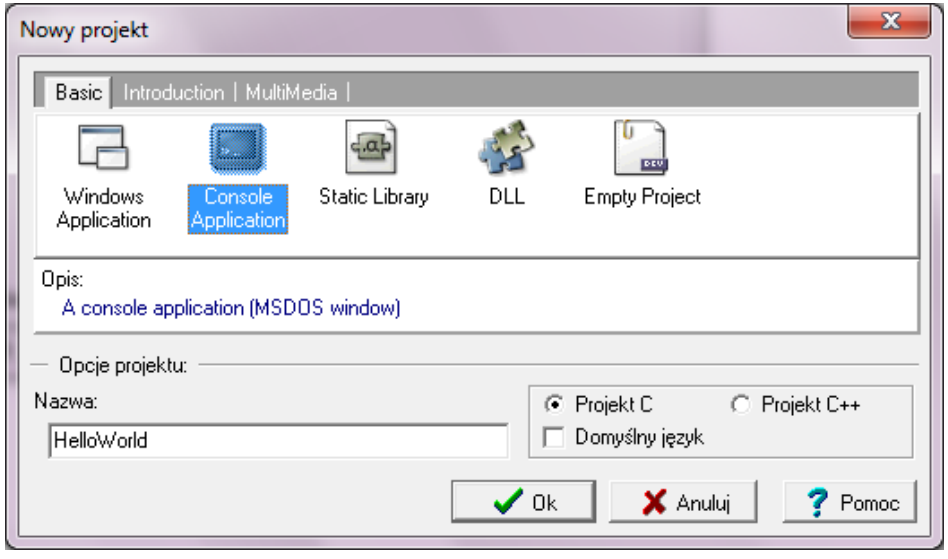
Uruchamiamy program Dev-C++. Na ekranie powinno pojawić się mniej więcej następujące okno:



Znajdź polecenie “Plik/Nowy/Projekt”.

Hello World - 3

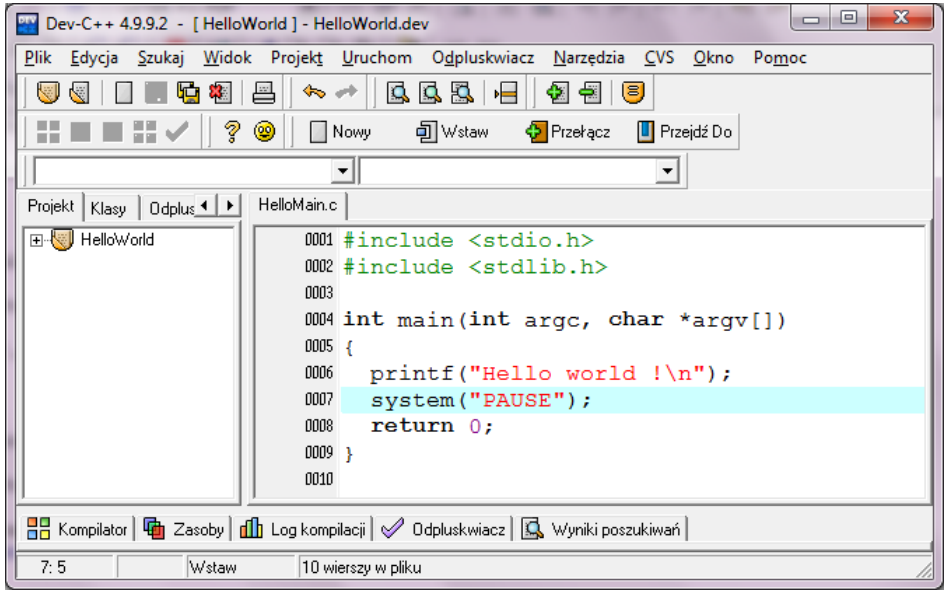
Pojawi się okno służące do ustalenia parametrów nowego projektu.



Wybierz “Aplikacja konsolowa”, zaznacz “Projekt C”, wpisz nazwę “HelloWorld”.

Hello World - 3

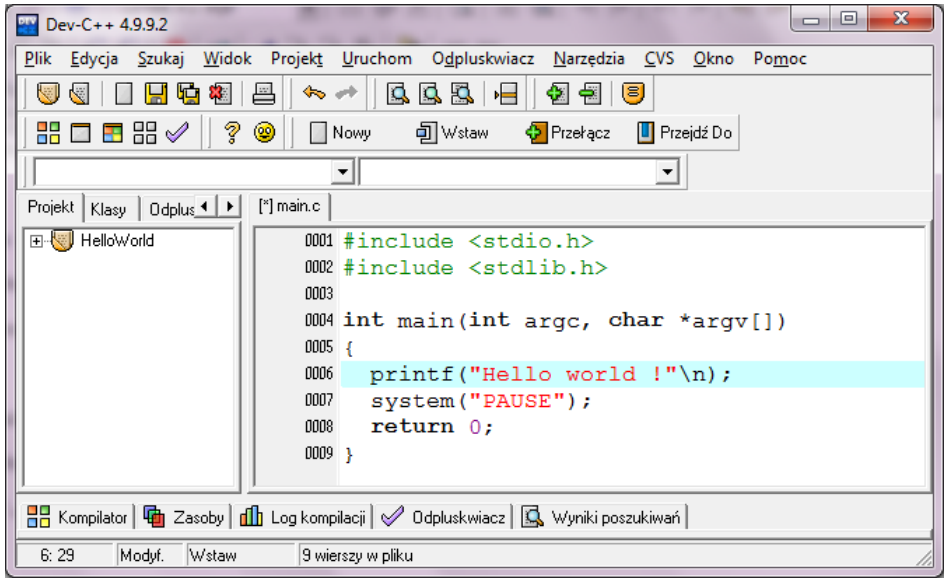
Oto efekt dotychczasowych czynności:



W panelu edytora wpisz jedną brakującą linijkę kodu, czyli:
printf("Hello world !\n”);

Hello World - 4

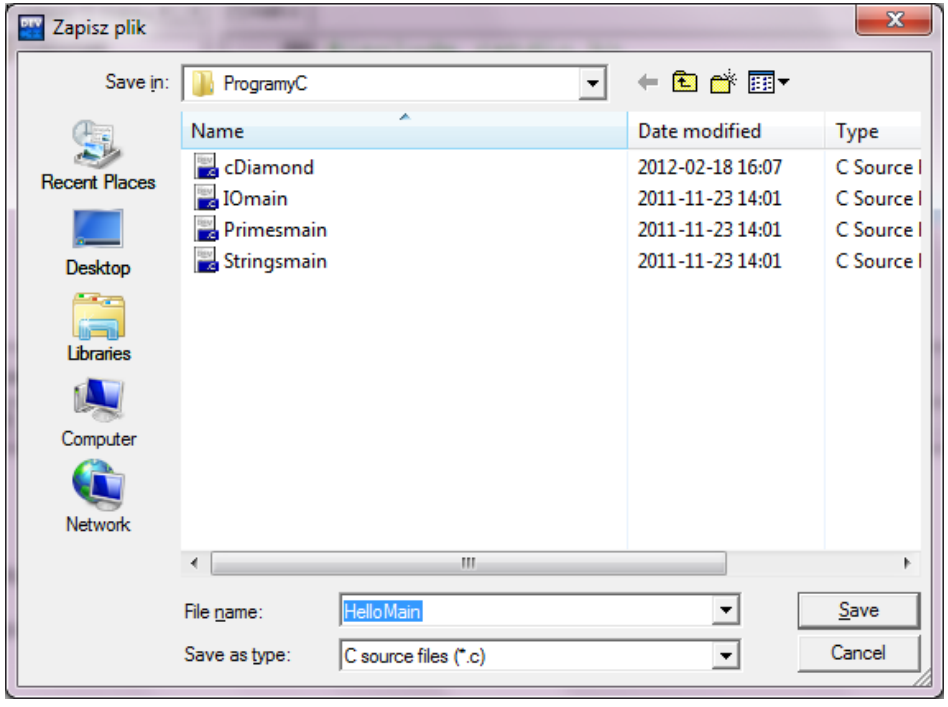
Oto spodziewany efekt dotychczasowych czynności:



Teraz znajdź polecenie “Uruchom/Kompiluj i uruchom”.

Hello World - 5

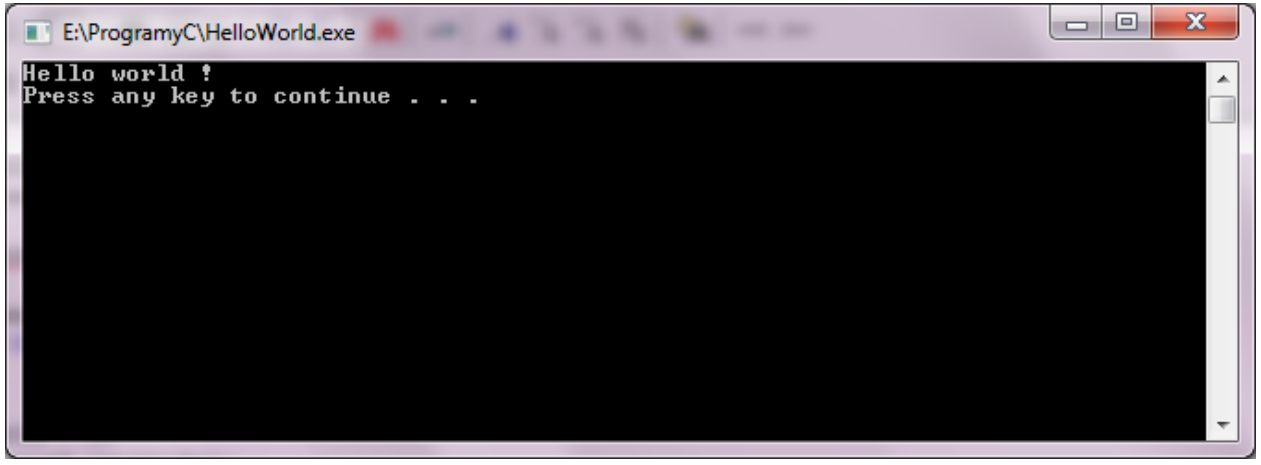
Pojawi się okno, w którego polu “File name” wprowadź nazwę “HelloMain”



Po wprowadzeniu tej nazwy naciśnij przycisk “Save”.

Hello World - 6

Na ekranie powinno pojawić się mniej więcej następujące okno:



Jeśli ci to się nie powiedzie, to przeczytaj uważnie komunikaty kompilatora. Błąd pewnie polega na braku średnika bądź na użyciu złych nawiasów.

Hello World - 6

W przyszłości nie będziemy już tak dokładnie opisywali procesu kompilacji kodu.

Zalecenia

- Powtórz samodzielnie wszystkie kroki wykonane podczas uruchamiania kodu programu “Hello World”.
- Powtarzaj samodzielnie czynności pierwszą, aż potrafisz bez korzystania z notatek napisać i uruchomić ten program.
- Korzystając z polecanych materiałów zrozum dokładnie każdy znak powyższego kodu. Możesz również zapytać się osobę prowadzącą laboratorium o detale.

Prędkość i czas spadania - I

Jeśli puścimy jakiś obiekt z wysokości h to uderzy on o ziemię po czasie

$$t = \sqrt{\frac{2G}{h}}$$

z szybkością

$$v = \sqrt{2Gh}\,.$$

CEL: napisanie programu, który po podaniu przez użytkownika wysokości h wyświetli mu na ekranie obie liczby (t oraz v).

Prędkość i czas spadania - II

Funkcje pomocnicze

```
#define G 9.81

float Czas(float h)
{
  return sqrt(2*h/G);
}

float Predkosc(float h)
{
  return sqrt(2*h*G);
}
```

1. sqrt jest funkcją, która zwraca pierwiastek kwadratowy
2. funkcja ta znajduje się w bibliotece math
3. słowo float oznacza liczbę rzeczywistą (później omówimy pewne szczegóły)
4. w pierwszej linijce zdefiniowaliśmy stałą

Prędkość i czas spadania - III

Funkcja main

```
int main(int argc, char *argv[])
{
  float H;

  printf("Podaj wysokosc w metrach: ");
  scanf("%f",&H);
  printf("Wysokosc : %f [m]\n",H);
  printf("Czas     : %f [sec]\n",Czas(H));
  printf("Predkosc : %f [m/sec]\n",Predkosc(H));
  printf("\n");

  system("PAUSE");
  return 0;
}
```

1. Funkcja printf
   - służy do wyświetlenia napisu
   - jej pierwszy parametr jest łańcuchem formatującym
   - "%f" oznacza miejsce do wstawienia wartości zmiennej typu float
2. Funkcja scanf
   - służy do wczytywania
   - zwróć uwagę na znak &

Prędkość i czas spadania - IV

Całość - część I

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define G 9.81

float Czas(float h)
{
  return sqrt(2*h/G);
}

float Predkosc(float h)
{
  return sqrt(2*h*G);
}
```

Całość - część 2

```
int main(int argc, char *argv[])
{
  float H;

  printf("Podaj wysokosc w metrach: ");
  scanf("%f",&H);
  printf("Wysokosc : %f [m]\n",H);
  printf("Czas     : %f [sec]\n",Czas(H));
  printf("Predkosc : %f [m/sec]\n",Predkosc(H));
  printf("\n");

  system("PAUSE");
  return 0;
}
```

Zmienne

Podstawowe typy zmiennych

1. char: znak
2. int: liczba całkowita
3. long: długa liczba całkowita
4. short: krótka liczba całkowita
5. float: liczba rzeczywista
6. double: dluga liczba rzeczywista
7. file: plik
8. void: pusty

Pętla FOR - I

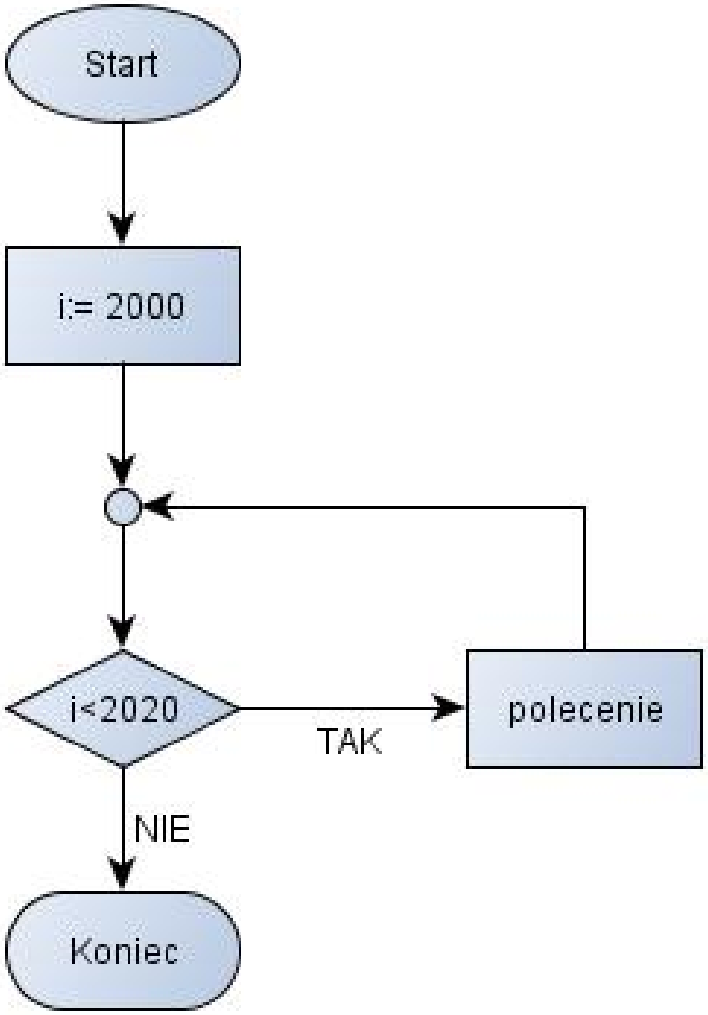
Czy rok jest przestepny ?

```
int Przestepny(int r)
{
    return ((r % 4 == 0)&&(r % 100 != 0))||(r % 400 ==0);
}

int main(int argc, char *argv[])
{
  for (int i = 2000;i<2020;i++)
    printf("%d %d\n",i,Przestepny(i));
  return 0;
}
```

1. == oznacza równość, != oznacza negację równości
2. && oznacza koniunkcję
3. || oznacza alternatywę
4. konstrukcję for omówimy za chwilę

Pętla FOR - II




```
for (int i = 2000;i<2020;i++)
{
  polecenie;
}
```

1. Podstaw $i := 2000$
2. Jeśli $i < 2020$ to idź do (3); jeśli nie to idź do (5)
3. Wykonaj polecenie
4. Zmień $i := i + 1$ i idź do (2)
5. Zakończ pętlę.

Pętla FOR - III

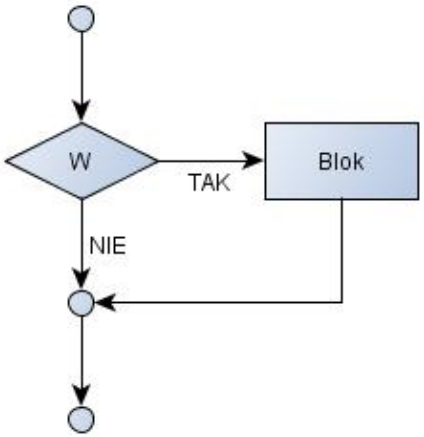
CEL: obliczenie liczby

$$S = \sum_{k=1}^{100} \frac{1}{k^2}$$

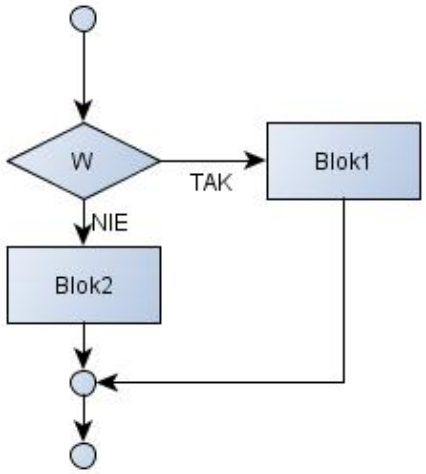
```
1   int main(int argc, char *argv[])
2   {
3     int     k;
4     double suma=0;
5
6     for (k=1;k<=100;k++)
7       suma += 1.0/(k*k);
8
9     printf("Suma = %f\n",suma);
10
11    return 0;
12  }
```

- Linię (7) można zastąpić suma= suma+1.0/(k*k);
- Uwaga: pomysł zastąpienia (7) kodem suma += 1/(k*k); nie jest zbyt mądry !!!

Konstrukcje warunkowe




```
if (W)
{
   Block;
}
```




```
if (W)
   Blok1;
else
   Blok2;
```

Konwencja

1. TRUE = wartość $\neq 0$
2. FALSE = wartość 0

Pętla WHILE - I (Hipoteza Collatz'a)

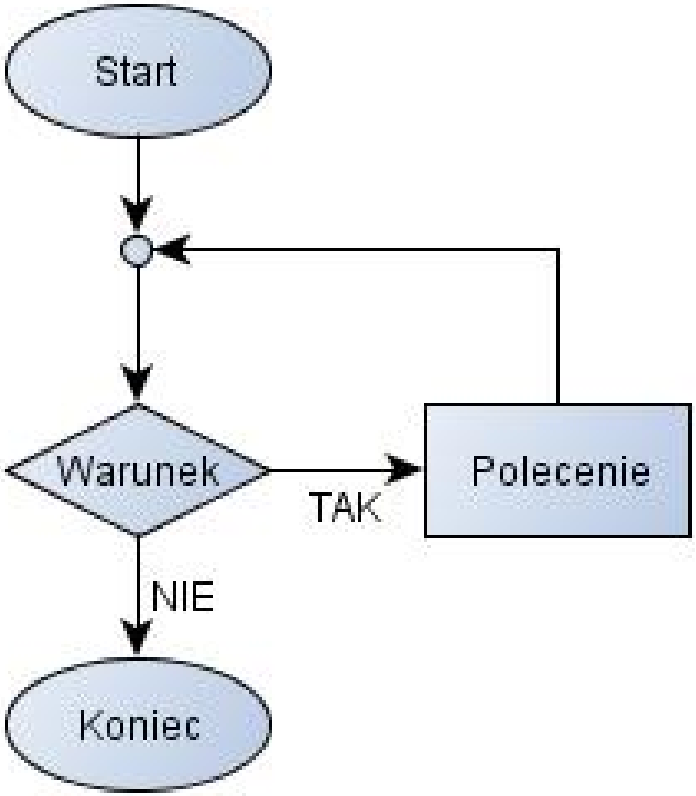
```
unsigned int D2OrM3(unsigned int x)
{
    unsigned int L=0;
    while (x > 1)  //x jest nieparzyste
    {
        L++;
        if (x%2)
          x = 3*x+1;
        else
          x/= 2; //to samo co x= x/2
    } //koniec petli while
    return L;
}

int main(int argc, char *argv[]) {
  for (int i=1;i<=100;i++)
    printf("%3d : %d\n",i,D2OrM3(i));

  return 0;
}
```

Do tej pory nie wiadomo, czy dla każdej liczby naturalnej n funkcja D2OrM3 zatrzymuje się po skończonej liczbie kroków !!!

Pętla WHILE - II (interpretacja)




```
while (Warunek)
  Polecenie;
```

1. Jeśli Warunek jest prawdziwy, to idź do (2); jesli nie, to idź do (4)
2. Wykonaj Polecenie
3. Idź do (1)
4. Zakończ petlę

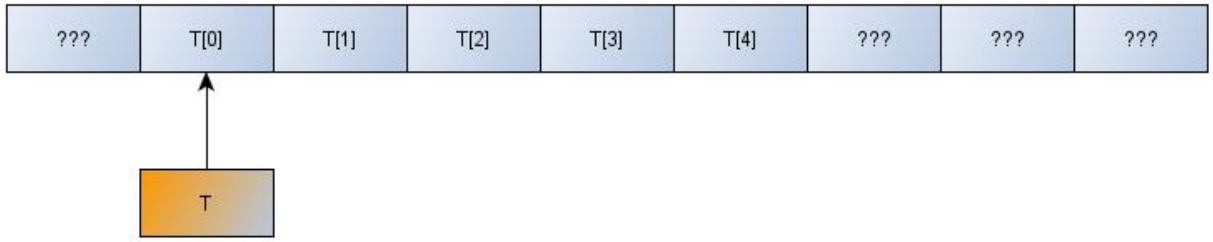
Zamiana zmiennych

```
void SwapInt(int *a, int *b)
{
    int r;
    r= *a;
   *a= *b;
   *b=  r;
}

int main(int argc, char *argv[])
{
  int x,y;
  x = 2;
  y = 5;
  printf("%d %d\n",x,y);
  SwapInt(&x,&y);
  printf("%d %d\n",x,y);
  return 0;
}
```

- Parametrami funkcji SwapInt są wskaźniki do zmiennych a i b
- Wewnątrz funkcji pracujemy na kopiach parametrów prostych !!!

Tablice



1. **int T[100];** : tablica 100 liczb typu int
2. **float T[200];** : tablica 200 liczb typu float
3. elementy tablicy numerowane są od zera
4. do $i$tego elementów tablicy odwołujemy się za pomocą konstrukcji $T[i]$

Tablice - liczby Fibbonacciego

```
1  #define MAX 20
2
3  void Fibbonacci(int F[],int rozmiar)
4  {
5      int i;
6      F[0]=1;
7      F[1]=1;
8      for(i=2;i<rozmiar;i++)
9        F[i]= F[i-1]+F[i-2];
10 }
11
12 int main(int argc, char *argv[])
13 {
14   int F[MAX]; //deklaracja tablicy
15   Fibbonacci(F,MAX);
16   for (int i=0;i<MAX;i++)
17     printf("%d ",F[i]);
18   printf("\n");
19   return 0;
20 }
```

- Tablice są przekazywane przez wskaźniki
- W linii 14 zadeklarowaliśmy tablicę 20 liczb całkowitych
- Elementy tablicy numerujemy od zera !

Łańcuchy znaków

```
1  void strrev(char s[]){
2      int  L,P;
3      char crob;       //zmienna robocza
4      L= 0;            //numer pierwszego znak lancucha
5      P= strlen(s)-1; //numer ostatniego znaku lancucha
6      while (L<P) {
7        crob= s[P];    //zamiana s[L] z s[P]
8        s[P]= s[L];
9        s[L]= crob;
10       L++;
11       P--;
12     }
13 }
14 int main(int argc, char *argv[]){
15   char str [256];
16
17   printf ("Podaj lancuch: ");
18   gets (str);
19   strrev(str);
20   printf ("Odwrocony     : %s\n",str);
21   system("PAUSE");
22   return 0;
23 }
```

Język C traktuje łańcuchy nonszalancko: są to tablice znaków zakończone znakiem char(0)

Struktura bajtu

Bajt = 8 bitów.

$$(b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0)_{(2)} = \sum_{i=0}^{7} b_i 2^i$$

$$(00000000)_{(2)} = 0$$

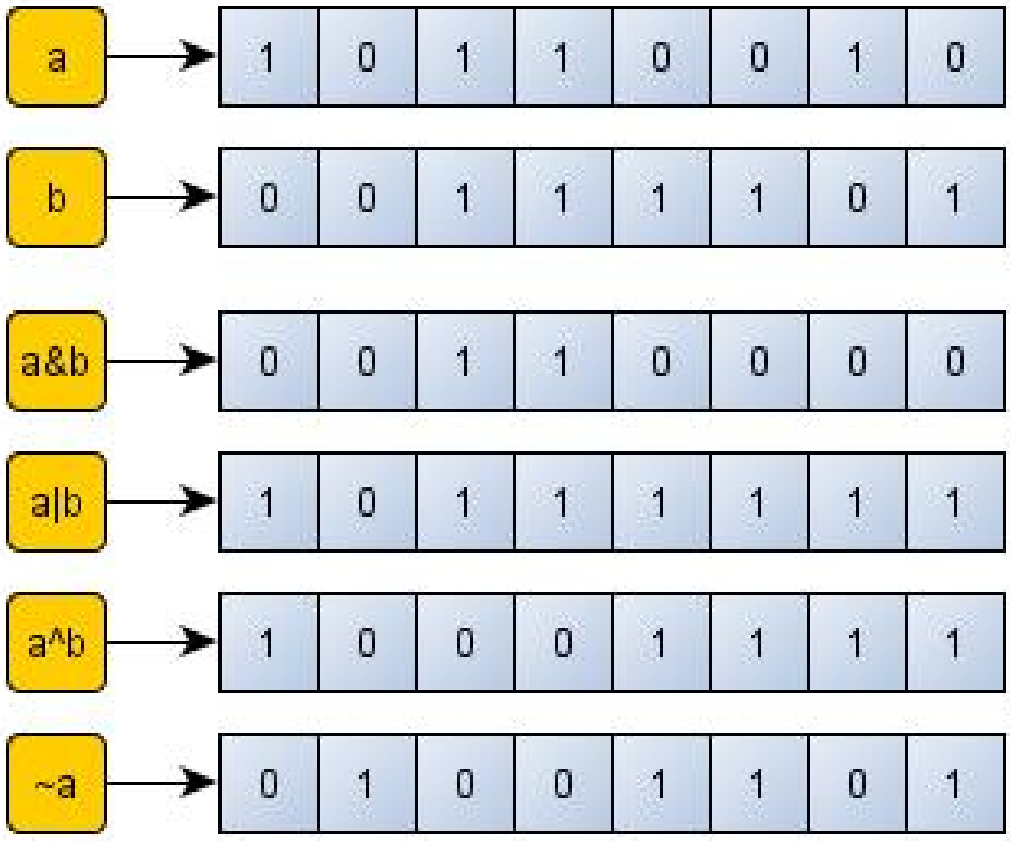
$$(11111111)_{(2)} = 1 + 2^2 + \ldots + 2^7 = \frac{2^8 - 1}{2 - 1} = 257$$

Operacje bitowe

```
/* Zapalenie k-tego bitu liczby a */
unsigned ZapalBit(unsigned a, int k)
{
  return a|(1<<k);
}
/* Zgaszenie k-tego bitu liczby a */
unsigned ZgasBit(unsigned a, int k){
  return a &(~(1<<k));
}
/* Sprawdzenie, czy k-ty bit liczby a jest zapalony*/
int Zapalony(unsigned a, int k){
  return a&(1<<k);
}
/* Liczba zapalonych bitów w a*/
int card(unsigned a) {
  int L =0;
  while (a)
  {
    L++;
    a&= a-1; //czyli a= a & (a-1);
  }
  return L;
}
```

1. &: bitowa koniunkcja
2. $|$: biowa alternatywa
3. $\sim$: bitowa negacja
4. $\wedge$: bitowe XOR
5. $a << k$: przesunięcie a o k bitów w lewo
6. $a >> k$: przesunięcie a o k bitów w prawo

Operacje bitowe - II



Najważniejsze biblioteki języka C

Biblioteki

1. <*stdio.h*>: podstawowe funkcje wejścia i wyjścia
2. <*stdlib.h*>: konwersja między typami numerycznymi, generator liczb pseudo-losowych, alokowanie pamięci
3. <*math.h*>: podstawowe funkcje matematyczne
4. <*string.h*>: funkcje obsługi łańcuchów
5. <*time.h*>: funkcje obsługi daty i czasu
6. ...

Algorytmy

Co nas czeka:

- Podstawowe algorytmy teorio-liczbowe ▸ Go
- Notacja $f = \mathrm{O}(g)$ ▸ Go
- Podstawowe algorytmy wyszukowania i sortowania ▸ Go
- Rekursja ▸ Go
- Technika Dziel i Rządź ▸ Go
- Algorytmy z nawrotami ▸ Go

oraz różne potrzebne do zrozumienia analizowanych algorytmów metody i fakty matematyczne.

Największy Wspólny Dzielnik

NWD

- $\mathrm{NWD}(a,b) = \max\{k \in \mathbf{N} : k|a \wedge k|b\}$
- jeśli $b|a$ to $\mathrm{NWD}(a,b) = b$
- jeśli $a = k \cdot b + r$ i $0 < r < b$ to $\mathrm{NWD}(a,b) = NWD(b,r)$

```
unsigned int NWD(unsigned int a, unsigned int b)
{
    unsigned int r;
    r = a%b;
    while (r)
    {
        a = b;
        b = r;
        r = a%b;  //dzielenie calkowito-liczbowe
    }
    return b;
}
```

Czy dana liczba jest pierwsza?

- Liczba $p \in \mathbf{N}$ jest pierwsza, jeśli z tego, że $k|p$ wynika, że $k = 1$ lub $k = p$
- Jeśli liczba n nie jest pierwsza, to istnieje $k \leq \sqrt{n}$ taka, że $k|n$

```
1  #define TRUE  1
2  #define FALSE 0
3
4  int IsPrime(unsigned n) {
5      unsigned int bound,i;
6      if (n < 2)
7        return FALSE;
8      bound = (unsigned int)sqrt(n); //zrzutowanie na typ unsigned int
9      for (i=2;i  <=bound;i++)
10     if (n%i==0)
11       return FALSE;
12     return TRUE;
13 }
```

Sito Erastotenesa

Generujemy tablicę $S[0,\ldots,n-1]$ taką, że $S[k] = \text{“}k \in \text{Primes}\text{”}$

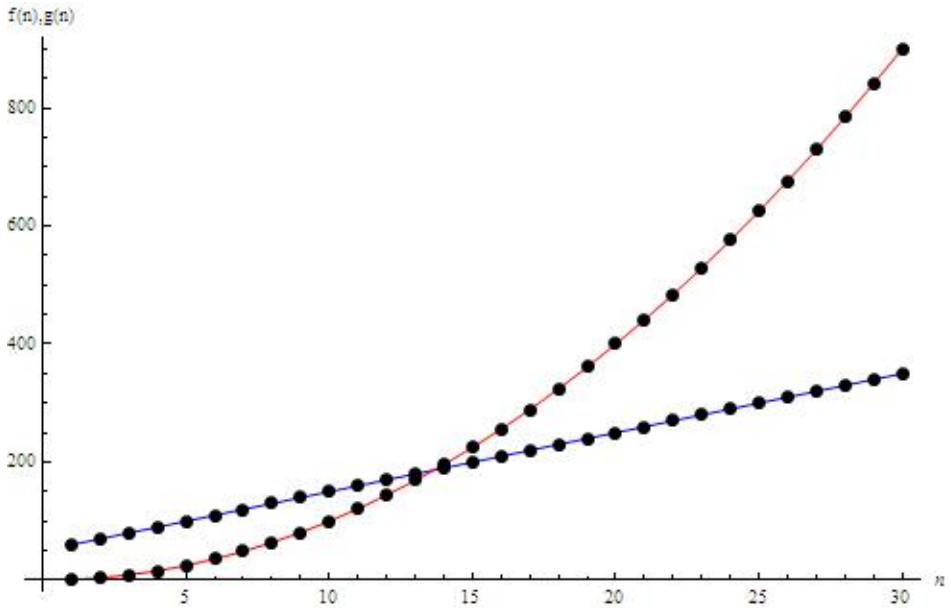
```
1  void SitoErastotenesa(int S[], int n) {
2      unsigned int bound,i,j;
3
4      bound = (unsigned int)sqrt(n);
5      S[0] = FALSE;
6      S[1] = FALSE;
7      for (i=2;i < n;i++)
8        S[i]=TRUE;
9
10     for (i=2;i <=bound;i++)
11     {
12         j=i*2;
13         while (j < n)
14         {
15             S[j]=FALSE;
16             j+=i;
17         }
18     }
19 } /*Sito Erastotenesa*/
```

Notacja Duże O - I

Definicja

Niech $f, g : \mathbf{N} \to \mathbf{R}$. Wtedy

$$f = \mathrm{O}(g) \leftrightarrow (\exists C > 0)(\exists N)(\forall n > N)\left(|f(n)| \leq C|g(n)|\right)$$



- $f(n) = 50 + 10 \cdot n$
- $g(n) = n^2$
- $f = \mathrm{O}(g)$
- $50 + 10 \cdot n = \mathrm{O}\left(n^2\right)$

Notacja Duże O - II

Z pomocą przychodzi nam Analiza Matematyczna.

Twierdzenie

Załóżmy, że $f, g : \mathbf{N} \to \mathbf{R}$. Wtedy

$$\left(\lim_{n\to\infty} \left| \frac{f(n)}{g(n)} \right| < \infty \right) \to (f = \mathrm{O}(g))$$

$$\lim_{n\to\infty} \frac{50+10n}{n^2} = \lim_{n\to\infty} \left(\frac{50}{n^2} + \frac{10}{n} \right) = 0 + 0 = 0 < \infty ,$$

więc $50 + 10n = \mathrm{O}\left(n^2\right)$.

Notacja Duże O - III

Definicja

Niech $f, g : \mathbf{N} \to \mathbf{R}$. Wtedy

1. $f = \Theta(g) \leftrightarrow f = \mathrm{O}(g) \wedge g = \mathrm{O}(f)$
2. $f = \mathrm{o}(g) \leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Twierdzenie

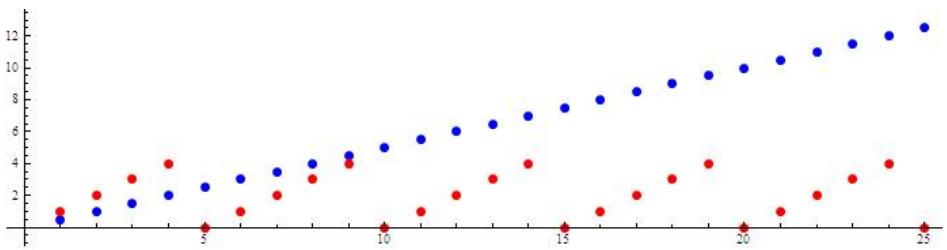
Niech $f, g : \mathbf{N} \to \mathbf{R}$. Wtedy

1. $f = \mathrm{o}(g) \to f = \mathrm{O}(g)$
2. $0 < \lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < \infty \to f = \Theta(g)$

Największy Wspólny Dzielnik - II

Twierdzenie

Załóżmy, że $a, b \in \mathbf{N}$ oraz $0 < b \leq a$. Wtedy $a \mod b < \frac{a}{2}$.



Twierdzenie

Algorytm NWD(a,b) działa w czasie $O(\max\{\log a, \log b\})$

Sortowanie przez wstawianie - I

Sortujemy tablicę $X[0,\ldots,n-1]$ liczb typu double.

```
1  void InSort(double X[],int n)
2  {
3      int i,j;
4      double m;
5
6      for (i=1;i<n;i++)
7      {
8          m = X[i];
9          j=i-1;
10         while ((j>=0)&&(X[j]>m))
11         {
12             X[j+1]=X[j];
13             j--;
14         }
15         X[j+1]=m;
16     }
17 } /*InSort*/
```

Sortowanie przez wstawianie - II

1. L_n = liczba porównań użytych dla tablicy rozmiaru n
2. $L_n \geq n - 1$
3. $L_n \leq \frac{1}{2}n(n-1)$

$$L_n \leq 1 + 2 + \ldots + (n-1) = \frac{(n-1)((n-1)+1)}{2} = \frac{(n-1)n}{2}$$

Bardzo Ważny Wzór

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Sortowanie przez wstawianie - III

L_n = liczba porównań użytych do sortowania tablicy rozmiaru n

Wniosek

$$L_n = \mathrm{O}\left(n^2\right)$$

Oznaczamy (na chwilę): $f \preceq g \leftrightarrow f = \mathrm{o}(g)$.

Klasy złożoności obliczeniowej

$$1 \preceq \log(n) \preceq \underbrace{n \preceq n\log n \preceq n^2 \preceq n^3}_{\textit{wielomianowe}} \preceq \ldots \underbrace{\preceq 2^n \preceq 3^n}_{\textit{wykladnicze}} \ldots$$

Wyszukiwanie w tablicy

Dane:

1. X = tablica liczb całkowitych rozmiaru n
2. a = poszukiwana liczba

CEL: Rozstrzygnięcie, czy a występuje w tablicy X

```
1  int Find(int X[], int n, int a)
2  {
3    int i;
4
5    for (i=0;i<n;i++)
6    if (X[i]==a) return TRUE;
7
8    return FALSE;
9  } /*Find*/
```

Liczba porównań: $O(n)$

Wyszukiwanie w tablicy uporządkowanej

Dane:

1. X = uporządkowana tablica liczb całkowitych rozmiaru n
2. a = poszukiwana liczba

CEL: Rozstrzygnięcie, czy a występuje w tablicy X

```
1  int BFind(int X[],int n, int a) {
2      int L,P,S;
3      L=0;
4      P=n-1;
5      while (L<=P)
6      {
7          S = L + (P-L)/2;
8          if (X[S]==a)
9            return TRUE;
10         else if (X[S]<a)
11           L = S+1;
12         else
13           P = S-1;
14     }
15     return FALSE;
16 } /* BFind */
```

Liczba porównań: $O(\log n)$

Definicje rekurencyjne

Silnia

$$0! = 1, \quad n > 0 \rightarrow n! = (n-1)! \cdot n$$

Liczby Fibbonacciego

$$F_0 = 1, \quad F_1 = 1, \quad n > 1 \rightarrow F_n = F_{n-2} + F_{n-1}$$

Funkcja Ackermana

$$A(m,n) = \begin{cases} n+1 & : \quad m = 0 \\ A(m-1,n) & : \quad m > 0 \wedge n = 0 \\ A(m-1,A(m,n-1)) & : \quad m > 0 \wedge n > 0 \end{cases}$$

Rekursja: problem wież z Hanoi - I

```
void H(int Ile, int i, int j)
{
    int k;
    if (Ile==1)
    {
        printf("(%d,%d)\n",i,j);
    }
    else
    {
        k= 6-(i+j);
        H(Ile-1,i,k);
        printf("(%d,%d)\n",i,j);
        H(Ile-1,k,j);
    }
}
```

Wynik wywołania funkcji H z parametrami (3,1,3): (1,3) (1,2) (3,2) (1,3) (2,1) (2,3) (1,3)

$$h_n = \begin{cases} 1 & : \quad n = 1 \\ 2 \cdot h_{n-1} + 1 & : \quad n > 1 \end{cases}$$

Rekursja: problem wież z Hanoi - I

```
void H(int Ile, int i, int j)
{
    int k;
    if (Ile==1)
    {
        printf("(%d,%d)\n",i,j);
    }
    else
    {
        k= 6-(i+j);
        H(Ile-1,i,k);
        printf("(%d,%d)\n",i,j);
        H(Ile-1,k,j);
    }
}
```

Wynik wywołania funkcji H z parametrami (3,1,3): (1,3) (1,2) (3,2) (1,3) (2,1) (2,3) (1,3)

$$h_n = \begin{cases} 1 & : \quad n = 1 \\ 2 \cdot h_{n-1} + 1 & : \quad n > 1 \end{cases}$$

Rekursja: problem wież z Hanoi - I

```
void H(int Ile, int i, int j)
{
    int k;
    if (Ile==1)
    {
        printf("(%d,%d)\n",i,j);
    }
    else
    {
        k= 6-(i+j);
        H(Ile-1,i,k);
        printf("(%d,%d)\n",i,j);
        H(Ile-1,k,j);
    }
}
```

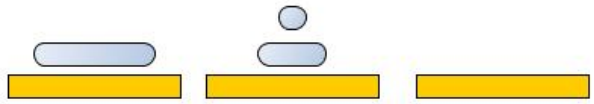
Wynik wywołania funkcji H z parametrami (3,1,3): (1,3) (1,2) (3,2) (1,3) (2,1) (2,3) (1,3)

$$h_n = \begin{cases} 1 & : \quad n = 1 \\ 2 \cdot h_{n-1} + 1 & : \quad n > 1 \end{cases}$$

Rekursja: problem wież z Hanoi - I

```
void H(int Ile, int i, int j)
{
    int k;
    if (Ile==1)
    {
        printf("(%d,%d)\n",i,j);
    }
    else
    {
        k= 6-(i+j);
        H(Ile-1,i,k);
        printf("(%d,%d)\n",i,j);
        H(Ile-1,k,j);
    }
}
```

Wynik wywołania funkcji H z parametrami (3,1,3): (1,3) (1,2) (3,2) (1,3) (2,1) (2,3) (1,3)

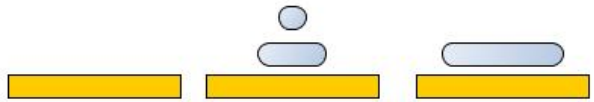


$$h_n = \begin{cases} 1 & : \quad n = 1 \\ 2 \cdot h_{n-1} + 1 & : \quad n > 1 \end{cases}$$

Rekursja: problem wież z Hanoi - I

```
void H(int Ile, int i, int j)
{
    int k;
    if (Ile==1)
    {
        printf("(%d,%d)\n",i,j);
    }
    else
    {
        k= 6-(i+j);
        H(Ile-1,i,k);
        printf("(%d,%d)\n",i,j);
        H(Ile-1,k,j);
    }
}
```

Wynik wywołania funkcji H z parametrami (3,1,3): (1,3) (1,2) (3,2) (1,3) (2,1) (2,3) (1,3)



$$h_n = \begin{cases} 1 & : \quad n = 1 \\ 2 \cdot h_{n-1} + 1 & : \quad n > 1 \end{cases}$$

Rekursja: problem wież z Hanoi - I

```
void H(int Ile, int i, int j)
{
    int k;
    if (Ile==1)
    {
        printf("(%d,%d)\n",i,j);
    }
    else
    {
        k= 6-(i+j);
        H(Ile-1,i,k);
        printf("(%d,%d)\n",i,j);
        H(Ile-1,k,j);
    }
}
```

Wynik wywołania funkcji H z parametrami (3,1,3): (1,3) (1,2) (3,2) (1,3) (2,1) (2,3) (1,3)

$$h_n = \begin{cases} 1 & : \quad n = 1 \\ 2 \cdot h_{n-1} + 1 & : \quad n > 1 \end{cases}$$

Rekursja: problem wież z Hanoi - I

```
void H(int Ile, int i, int j)
{
    int k;
    if (Ile==1)
    {
        printf("(%d,%d)\n",i,j);
    }
    else
    {
        k= 6-(i+j);
        H(Ile-1,i,k);
        printf("(%d,%d)\n",i,j);
        H(Ile-1,k,j);
    }
}
```

Wynik wywołania funkcji H z parametrami (3,1,3): (1,3) (1,2) (3,2) (1,3) (2,1) (2,3) (1,3)

$$h_n = \begin{cases} 1 & : \quad n = 1 \\ 2 \cdot h_{n-1} + 1 & : \quad n > 1 \end{cases}$$

Rekursja: problem wież z Hanoi - I

```
void H(int Ile, int i, int j)
{
    int k;
    if (Ile==1)
    {
        printf("(%d,%d)\n",i,j);
    }
    else
    {
        k= 6-(i+j);
        H(Ile-1,i,k);
        printf("(%d,%d)\n",i,j);
        H(Ile-1,k,j);
    }
}
```

Wynik wywołania funkcji H z parametrami (3,1,3): (1,3) (1,2) (3,2) (1,3) (2,1) (2,3) (1,3)

$$h_n = \begin{cases} 1 & : \quad n = 1 \\ 2 \cdot h_{n-1} + 1 & : \quad n > 1 \end{cases}$$

Rekursja: problem wież z Hanoi - II

Równanie rekurencyjne

$$h_n = \begin{cases} 1 & : \quad n = 1 \\ 2 \cdot h_{n-1} + 1 & : \quad n > 1 \end{cases}$$

1. Kolejne wartości h_n: 1,3,7,15,31,63, ...
2. Hipoteza: $h_n = 2^n - 1$
3. Dowód (indukcja matematyczna):

$$h_1 = 2^1 - 1 = 2 - 1 = 1$$

$$h_{n+1} = 2 \cdot h_n + 1 = 2(2^n - 1) + 1 = 2 \cdot 2^n - 1 = 2^{n+1} - 1 \, .$$

Rekursja: problem wież z Hanoi - II

Równanie rekurencyjne

$$h_n = \begin{cases} 1 & : \quad n = 1 \\ 2 \cdot h_{n-1} + 1 & : \quad n > 1 \end{cases}$$

1. Kolejne wartości h_n: 1,3,7,15,31,63, ...
2. Hipoteza: $h_n = 2^n - 1$
3. Dowód (indukcja matematyczna):

$$h_1 = 2^1 - 1 = 2 - 1 = 1$$

$$h_{n+1} = 2 \cdot h_n + 1 = 2(2^n - 1) + 1 = 2 \cdot 2^n - 1 = 2^{n+1} - 1 \, .$$

Rekursja: problem wież z Hanoi - II

Równanie rekurencyjne

$$h_n = \begin{cases} 1 & : \quad n = 1 \\ 2 \cdot h_{n-1} + 1 & : \quad n > 1 \end{cases}$$

1. Kolejne wartości h_n: 1,3,7,15,31,63, ...
2. Hipoteza: $h_n = 2^n - 1$
3. Dowód (indukcja matematyczna):

$$h_1 = 2^1 - 1 = 2 - 1 = 1$$

$$h_{n+1} = 2 \cdot h_n + 1 = 2(2^n - 1) + 1 = 2 \cdot 2^n - 1 = 2^{n+1} - 1 \, .$$

Dziel i rządź

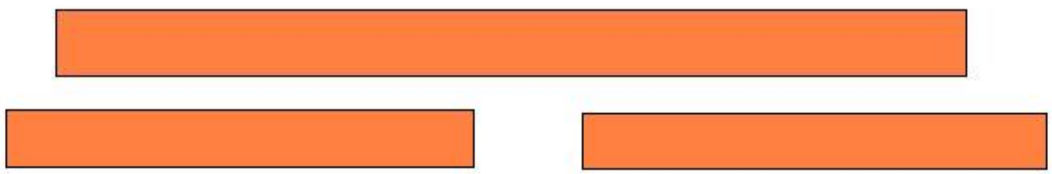
Idea

Podziel zadanie na mniejsze część, rozwiąż oddzielnie każde z podzadań i scal rozwiązania częściowe w rozwiązanie całego zadania

Dziel i rządź

Idea

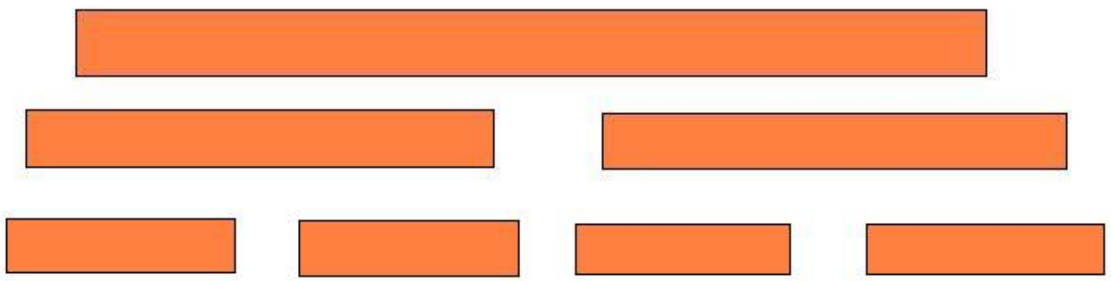
Podziel zadanie na mniejsze część, rozwiąż oddzielnie każde z podzadań i scal rozwiązania częściowe w rozwiązanie całego zadania



Dziel i rządź

Idea

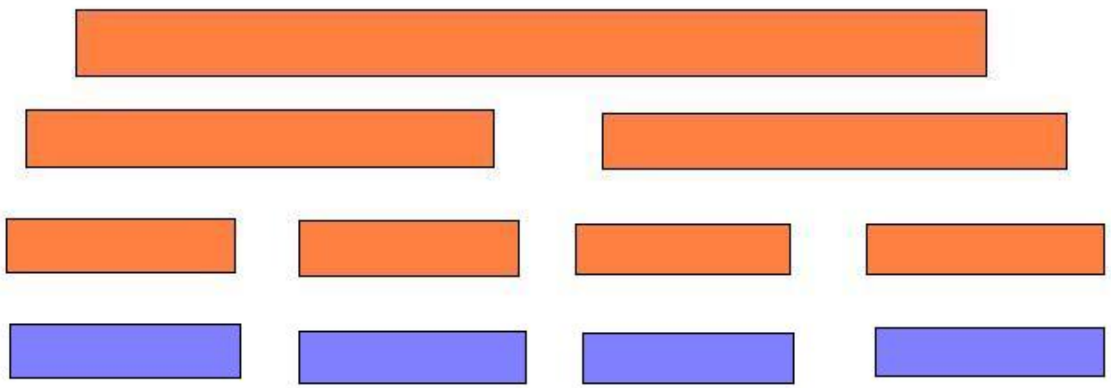
Podziel zadanie na mniejsze część, rozwiąż oddzielnie każde z podzadań i scal rozwiązania częściowe w rozwiązanie całego zadania



Dziel i rządź

Idea

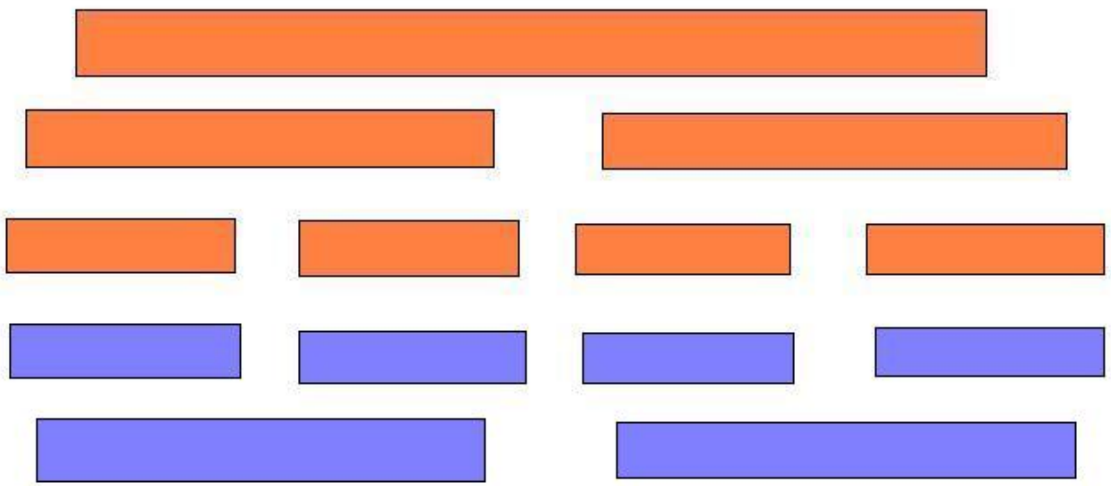
Podziel zadanie na mniejsze część, rozwiąż oddzielnie każde z podzadań i scal rozwiązania częściowe w rozwiązanie całego zadania



Dziel i rządź

Idea

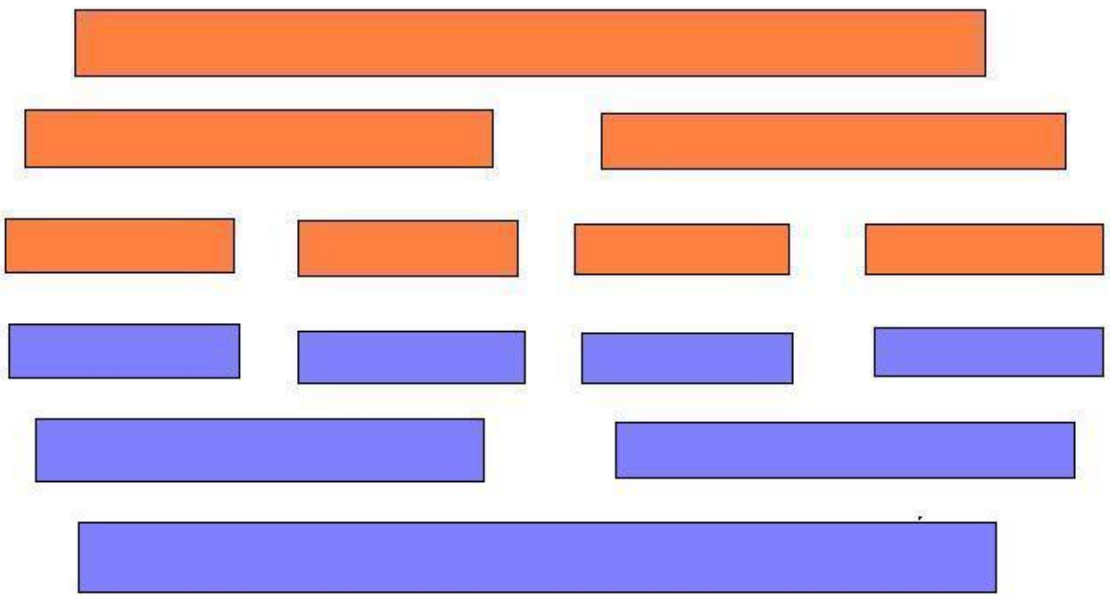
Podziel zadanie na mniejsze część, rozwiąż oddzielnie każde z podzadań i scal rozwiązania częściowe w rozwiązanie całego zadania



Dziel i rządź

Idea

Podziel zadanie na mniejsze część, rozwiąż oddzielnie każde z podzadań i scal rozwiązania częściowe w rozwiązanie całego zadania



Rekursja: Sortowanie przez scalanie - I

```
double T[MAX];
double rob[MAX];

void MS(int L,int P)
{
    int S,i,j,k;

    if (L==P) return;

    S= (L+P)/2; //punkt srodkowy
    MS(L, S); //rekursja
    MS(S+1,P); //rekursja
    i= L;
    j= S+1;
    k= 0;
    while((i<=S)&&(j<=P)) //scalanie
    {
        if (T[i]<T[j]) {rob[k]= T[i];i++;}
        else           {rob[k]= T[j];j++;}
        k++;
    }
    while (i<=S){rob[k]= T[i];i++;k++;}
    while (j<=P){rob[k]= T[j];j++;k++;}
    for (i=L;i<=P;i++) //kopiowanie
      T[i]= rob[i-L];
}
```

```
int main(int argc, char *argv[])
{
  T[0]=10;T[1]=5;T[2]= 88.8;T[3]= 12;
  MS(0,3);
  printf("%f %f %f %f\n\n",
         T[0],T[1],T[2],T[3]);
  system("PAUSE");
  return 0;
}
```

1. $L(n)$ = liczba porównań
2. $L(1) = 0$
3. $L(n) \leq 2 \cdot L(\lfloor \frac{n}{2} \rfloor) + n$
4. Jak oszacować $L(n)$?

Sortowanie przez scalanie - II

Część 1

Rozważamy funkcję $f : \mathbf{N} \to \mathbf{R}$ taką, że $f(1) = d$ oraz

$$f(n) = 2f(\left\lfloor \frac{n}{2} \right\rfloor) + b \cdot n \,.$$

Wprowadzamy pomocniczą funkcję $g(n) = f(2^n)/2^n$. Wtedy $g(0) = d$ oraz

$$g(n+1) = \frac{f(2^{n+1})}{2^{n+1}} = \frac{2f(2^n) + b2^{n+1}}{2^{n+1}} = \frac{f(2^n)}{2^n} + b = g(n) + b$$

Sortowanie przez scalanie - III

Część 2: analiza równania

Z tego, że $g(0) = d$ i $g(n+1) = g(n) + b$ wynika, że

$$g(n) = bn + d,$$

więc

$$f(2^n) = 2^n g(n) = 2^n(bn + d).$$

Niech $N = 2^n$. Wtedy $n = \log_2 N$, więc

$$f(N) = N(b \log_2 N + d)$$

Sortowanie przez scalanie - IV

TWIERDZENIE O SORTOWANIU PRZEZ SCALANIE

Dla N postaci 2^k mamy

$$N - 1 \leq L_N \leq N \log_2 N$$

Mniej dokładna wersja

Dla dowolnego N mamy

$$L_N = \mathrm{O}\left(N \log N\right)$$

Porównanie (siła ALGORYTMIKI)

1. rozmiar tablicy: $N = 10^7$
2. taktowanie procesora: $\omega = 3GH$
3. jedno porównanie: $n = 10$ taktów procesora

Sortowanie przez wstawianie

$$T = \frac{\frac{1}{2}N^2 \cdot n}{\omega} = \frac{1}{2}\frac{10^{15}}{3 \cdot 10^9}[sek] = \frac{3}{2}10^6[sek] \approx 17.4[dni]$$

kod

Sortowanie przez scalanie

$$T = \frac{N \cdot \log_2 N \cdot n}{\omega} \approx 0.56[sek]$$

kod

Przeszukiwanie z nawrotami: Problem Hetmanów

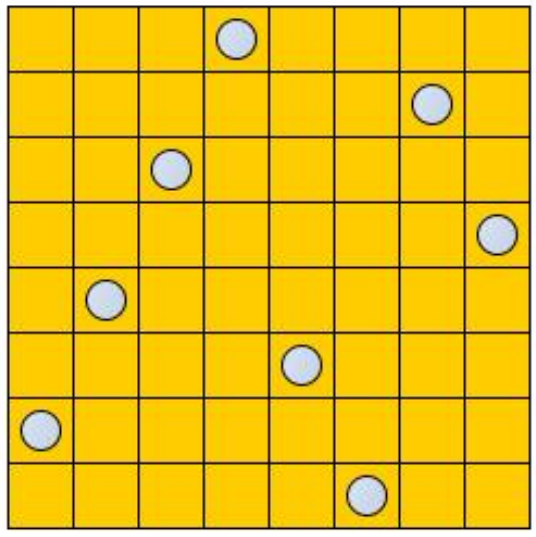
```
int OK(int H[],int L) {
   int i;

   for (i=1;i<L;i++)
   if ((H[i]==H[L]) || (abs(H[i]-H[L])==L-i))
     return 0;
   return 1;
}

void UstawHetmany(int H[],int L)
{
  int i;
  if (L>8)
  {
    for (i=1;i<=8;i++)
      printf("%d ",H[i]);
    printf("\n");
    return;
  }

  for (i=1;i<=8;i++)
  {
    H[L]=i;
    if (OK(H,L))
      UstawHetmany(H,L+1);
  }
}
```

```
int main(int argc, char *argv[])
{
   int H[9];
   UstawHetmany(H,1);
   return EXIT_SUCCESS;
}
```



Co dalej was czeka ?

Kursy związane z Algorytmiką

1. Matematyka Dyskretna
2. Kursy programowania
3. Algorytmy i Struktury Danych
4. Paradygmaty Programowania
5. Probabilistyka
6. Języki Formalne
7. Różne kursy do wyboru

Ważne fakty

Wzory

1. $1 + 2 + \ldots + n = \frac{1}{2}n(n+1)$
2. $1^2 + 2^2 + \ldots + n^2 = \frac{1}{6}n(n+1)(2n+1)$
3. $1 + q + q^2 + \ldots + q^n = \frac{1-q^{n+1}}{1-q}$ dla $q \neq 1$
4. $\binom{n}{k} = \frac{n!}{k!(n-k)!} = |\{X \subseteq \{1,\ldots,n\} : |X| = k\}|$
5. Reguła de l'Hospitala: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$
6. $\log_a b = \log_c b / \log_c a$

O stylu programowania

Ogólne zasady

1. Pisz kod dla czlowieka a nie dla maszyny
2. Stosuj jednolitą konwencję formatowania kodu
3. Stosuj tylko sensowne nazwy zmiennych (poza iteratorami)
4. Stosuj zasadę KISS (“**K**eep it **S**imple, **S**tupid”)
5. Komentuj istotne fragmenty kodu
6. Nie komentuj oczywistości
7. Dokumentuj kod (autor, data powstania, modyfikacji,...)

Złożoności obliczeniowe omawianych algorytmów

Algorytm	Złożoność
Naiwny test pierwszości	$O\left(\sqrt{n}\right)$
Wyszukiwanie w dowolnej tablicy	$O\left(n\right)$
Wyszukiwanie w tablicy uporządkowanej	$O\left(\log n\right)$
Sortowanie przez wstawianie	$O\left(n^2\right)$
Sortowanie przez scalanie	$O\left(n \log n\right)$
NWD(a,b)	$O\left(\max\{\log a, \log b\}\right)$

Wybrane zadania I

Liczby bliźniacze Liczby bliźniacze to takie dwie liczby pierwsze, których różnica wynosi 2. Napisz funkcję która dla zadanej liczby naturalnej n wyznaczy ilość par liczb bliźniaczych mniejszych od n.

Liczby zaprzyjaźnione Niech $\sigma(n)$ oznacza sumę wszystkich dzielników liczby naturalnej n mniejszych od liczby n (na przykład $\sigma(5) = 1$ oraz $\sigma(6) = 1 + 2 + 3 = 6$. Liczbę n nazywamy doskonałą jeśli $\sigma(n) = n$. Parę liczb (n, m) nazywamy zaprzyjaźnioną, jeśli $\sigma(n) = m$ oraz $\sigma(m) = n$. Znajdź wszystkie liczby doskonałe mniejsze od 1000. Wyznacz wszystkie zaprzyjaźnione pary liczb mniejszych niż 1000.

Rozwiązywanie równań liniowych w liczbach całkowitych Napisz program który dla danych liczb całkowitych a, b, c znajduje, jeśli istnieje, przynajmniej jedno rozwiązanie liczbach całkowitych równania $a \cdot X + b \cdot Y = c$.

Sito Erastotenesa Przerób algorytm wyznaczania sita Erastotenesa tak aby pracował on na pojedynczych bitach, czyli tak aby do reprezentacji jednej liczby naturalnej wykorzystywał tylko jeden bit.

Zamiana baz Napisz funkcję, która dla zadanej liczby naturalnej n zwraca jej reprezentacje binarną oraz w układzie szesnastkowym.

Generowanie podzbiorów Napisz procedurę, która generuje wszystkie podzbiory k - elementowe zbioru $\{1, 2, \ldots, n\}$.

Wybrane zadania II

Generowanie permutacji Napisz procedurę, która generuje wszystkie permutacje zbioru $\{1, 2, \ldots, n\}$.

Tautologie Napisz program, który wczytuje formułę rachunku zdań zapisaną w notacji polskiej, zbudowaną ze zmiennych zdaniowych p, q, r, s, t, drukuje jej tablicę zero-jedynkową i sprawdza, czy jest ona tautologią.

Funkcja Ackremana Pokaż, że funkcja Ackremana jest poprawnie zdefiniowana. Wyznacz wartości funkcji Ackremana dla małych wartości parametrów.

Liczba zapalonych bitów Napisz funkcję, która dla podanej liczby typu **unsigned** oblicza liczbę jej zapalonych bitów.

Anagramy Słowo A nazywamy anagramem słowa B jeśli słowo A można otrzymać ze słowa B za pomocą przestawienia kolejność liter. Napisz procedurę, który dla podanych dwóch ciągów znaków rozstrzyga, czy są one wzajemnymi anagramami.

Problem skoczka szachowego Napisz procedurę, który wyznaczy taką sekwencję ruchów skoczka szachowego, aby obszedł on całą szachownicę, przechodząc przez każde pole dokładnie jeden raz.