Inżynieria oprogramowania: **optymalizacja czasowa programów**

Zdzisław Sroczyński

Politechnika Śląska
Instytut Matematyki
Wydział Matematyki Stosowanej

- Priorytet: poprawność i niezawodność programów

- Priorytet: poprawność i niezawodność programów
- Warunki przeprowadzenia optymalizacji programu:

- Priorytet: poprawność i niezawodność programów
- Warunki przeprowadzenia optymalizacji programu:
  - program wielokrotnego użycia

- Priorytet: poprawność i niezawodność programów
- Warunki przeprowadzenia optymalizacji programu:
  - program wielokrotnego użycia
  - optymalizacja przewidziana w celach projektu

- Priorytet: poprawność i niezawodność programów
- Warunki przeprowadzenia optymalizacji programu:
  - program wielokrotnego użycia
  - optymalizacja przewidziana w celach projektu
  - określone wymogi na sprawność programu

- Priorytet: poprawność i niezawodność programów
- Warunki przeprowadzenia optymalizacji programu:
  - program wielokrotnego użycia
  - optymalizacja przewidziana w celach projektu
  - określone wymogi na sprawność programu
  - istotna poprawa wydajności po optymalizacji

- Priorytet: poprawność i niezawodność programów
- Warunki przeprowadzenia optymalizacji programu:
  - program wielokrotnego użycia
  - optymalizacja przewidziana w celach projektu
  - określone wymogi na sprawność programu
  - istotna poprawa wydajności po optymalizacji
  - ograniczenie ryzyka wprowadzenia nowych błędów

- Priorytet: poprawność i niezawodność programów
- Warunki przeprowadzenia optymalizacji programu:
  - program wielokrotnego użycia
  - optymalizacja przewidziana w celach projektu
  - określone wymogi na sprawność programu
  - istotna poprawa wydajności po optymalizacji
  - ograniczenie ryzyka wprowadzenia nowych błędów
- Sposoby optymalizacji czasowej programów:
  - ponowny projekt z wykorzystaniem ulepszonych algorytmów lub nowych struktur danych

- Priorytet: poprawność i niezawodność programów
- Warunki przeprowadzenia optymalizacji programu:
  - program wielokrotnego użycia
  - optymalizacja przewidziana w celach projektu
  - określone wymogi na sprawność programu
  - istotna poprawa wydajności po optymalizacji
  - ograniczenie ryzyka wprowadzenia nowych błędów
- Sposoby optymalizacji czasowej programów:
  - ponowny projekt z wykorzystaniem ulepszonych algorytmów lub nowych struktur danych
  - usprawnienie fragmentów programu bez modyfikacji algorytmu jako całości

- Priorytet: poprawność i niezawodność programów
- Warunki przeprowadzenia optymalizacji programu:
  - program wielokrotnego użycia
  - optymalizacja przewidziana w celach projektu
  - określone wymogi na sprawność programu
  - istotna poprawa wydajności po optymalizacji
  - ograniczenie ryzyka wprowadzenia nowych błędów
- Sposoby optymalizacji czasowej programów:
  - ponowny projekt z wykorzystaniem ulepszonych algorytmów lub nowych struktur danych
  - **usprawnienie fragmentów programu bez modyfikacji algorytmu jako całości**

- Stosunek kosztów do wartości planowanych usprawnień

- Stosunek kosztów do wartości planowanych usprawnień
- Czynniki istotne podczas optymalizacji czasowej:

- Stosunek kosztów do wartości planowanych usprawnień
- Czynniki istotne podczas optymalizacji czasowej:
  - profil dynamiczny programu – procent czasu zużywany przez poszczególne fragmenty

- Stosunek kosztów do wartości planowanych usprawnień
- Czynniki istotne podczas optymalizacji czasowej:
  - profil dynamiczny programu – procent czasu zużywany przez poszczególne fragmenty
  - stopień (procent) optymalizacji możliwy do osiągnięcia w poszczególnych fragmentach

- Stosunek kosztów do wartości planowanych usprawnień
- Czynniki istotne podczas optymalizacji czasowej:
  - profil dynamiczny programu – procent czasu zużywany przez poszczególne fragmenty
  - stopień (procent) optymalizacji możliwy do osiągnięcia w poszczególnych fragmentach
  - celowość optymalizacji (rzadko używane funkcje, czas reakcji systemu w systemach czasu rzeczywistego)

- Stosunek kosztów do wartości planowanych usprawnień
- Czynniki istotne podczas optymalizacji czasowej:
  - profil dynamiczny programu – procent czasu zużywany przez poszczególne fragmenty
  - stopień (procent) optymalizacji możliwy do osiągnięcia w poszczególnych fragmentach
  - celowość optymalizacji (rzadko używane funkcje, czas reakcji systemu w systemach czasu rzeczywistego)
  - liczba osobogodzin koniecznych do wykonania optymalizacji w poszczególnych fragmentach – opłacalność optymalizacji

- Stosunek kosztów do wartości planowanych usprawnień
- Czynniki istotne podczas optymalizacji czasowej:
  - profil dynamiczny programu – procent czasu zużywany przez poszczególne fragmenty
  - stopień (procent) optymalizacji możliwy do osiągnięcia w poszczególnych fragmentach
  - celowość optymalizacji (rzadko używane funkcje, czas reakcji systemu w systemach czasu rzeczywistego)
  - liczba osobogodzin koniecznych do wykonania optymalizacji w poszczególnych fragmentach – opłacalność optymalizacji

- Wybór fragmentu pozwalającego na **istotną** optymalizację

- Wybór fragmentu pozwalającego na **istotną** optymalizację
- **region krytyczny** – stosunkowo niewielki (10%) fragment programu, który zajmuje większą część czasu wykonania (90%)

- Wybór fragmentu pozwalającego na **istotną** optymalizację
- **region krytyczny** – stosunkowo niewielki (10%) fragment programu, który zajmuje większą część czasu wykonania (90%)
- reguła 90–10 (jedna z wielu)

- Wybór fragmentu pozwalającego na **istotną** optymalizację
- **region krytyczny** – stosunkowo niewielki (10%) fragment programu, który zajmuje większą część czasu wykonania (90%)
- reguła 90–10 (jedna z wielu)
- metody znajdowania regionów krytycznych:

- Wybór fragmentu pozwalającego na **istotną** optymalizację
- **region krytyczny** – stosunkowo niewielki (10%) fragment programu, który zajmuje większą część czasu wykonania (90%)
- reguła 90–10 (jedna z wielu)
- metody znajdowania regionów krytycznych:
  - automatyczne wyznaczenie profilu dynamicznego (za pomocą programu/modułu *profilera*)

- Wybór fragmentu pozwalającego na **istotną** optymalizację
- **region krytyczny** – stosunkowo niewielki (10%) fragment programu, który zajmuje większą część czasu wykonania (90%)
- reguła 90–10 (jedna z wielu)
- metody znajdowania regionów krytycznych:
  - automatyczne wyznaczenie profilu dynamicznego (za pomocą programu/modułu *profilera*)
  - pomiar czasu wykonywania fragmentów programu (głównie podprogramów – funkcji, metod)

- Wybór fragmentu pozwalającego na **istotną** optymalizację
- **region krytyczny** – stosunkowo niewielki (10%) fragment programu, który zajmuje większą część czasu wykonania (90%)
- reguła 90–10 (jedna z wielu)
- metody znajdowania regionów krytycznych:
  - automatyczne wyznaczenie profilu dynamicznego (za pomocą programu/modułu *profilera*)
  - pomiar czasu wykonywania fragmentów programu (głównie podprogramów – funkcji, metod)
  - generowanie histogramu z wykorzystaniem przerwań sprzętowych

Raport profilera:

Welcome Page | UOptCzas | AQtimeModule1 | Report

Routine Name	Time ▽	Time with Children
Winapi::Windows::WaitMessage	231,15	231,15
System::Generics::Defaults::Compare_Variant	6,82	6,82
TArray::QuickSort<Uoptczas::TInt *>	3,73	10,65
System::Generics::Defaults::TDelegatedComparer__1<Uoptcz...	3,68	6,91
Winapi::Windows::CallWindowProc	3,60	30,35
TArray::QuickSort<int>	2,10	4,05
System::Generics::Defaults::Compare_I4	1,96	1,96
TForm1::BtnArrayOfTintClick	1,65	1,65
Uoptczas::porownaj	1,63	1,63
TForm1::BtnListTClick	1,58	1,58
TForm1::BtnObjListClick	1,43	3,06
TArray::QuickSort<System::Variant>	1,31	8,14
Winapi::Windows::DispatchMessageW	0,68	31,13
Winapi::Windows::GetWindowThreadProcessId	0,68	0,68
Winapi::Windows::WindowFromPoint	0,43	0,44

pomiar.php: czas wykonania programu w PHP

```php
<?php

$_licznik=0;

function pomiar_start()
{
    global $_licznik;
    list($usec, $sec) = explode(' ',microtime());
    $_licznik=((float)$usec + (float)$sec);
}

function pomiar_stop($napis)
{
    global $_licznik;
    list($usec, $sec) = explode(' ',microtime());
    $czas=((float)$usec + (float)$sec)-$_licznik;
    printf('%s - Czas wykonania: %.4f<br />',$napis,$czas);
}

?>
```

- Oszacowanie możliwych rezultatów optymalizacji

- Oszacowanie możliwych rezultatów optymalizacji
- Wykorzystanie heurystyki i doświadczenia programisty, przegląd możliwych do zastosowania optymalizacji

- Oszacowanie możliwych rezultatów optymalizacji
- Wykorzystanie heurystyki i doświadczenia programisty, przegląd możliwych do zastosowania optymalizacji
- Złożoność algorytmów czynnikiem utrudniającym wybór fragmentu do optymalizacji

- Oszacowanie możliwych rezultatów optymalizacji
- Wykorzystanie heurystyki i doświadczenia programisty, przegląd możliwych do zastosowania optymalizacji
- Złożoność algorytmów czynnikiem utrudniającym wybór fragmentu do optymalizacji
  - zależność czasu wykonania od rozmiaru i/lub jakości danych wejściowych

- Oszacowanie możliwych rezultatów optymalizacji
- Wykorzystanie heurystyki i doświadczenia programisty, przegląd możliwych do zastosowania optymalizacji
- Złożoność algorytmów czynnikiem utrudniającym wybór fragmentu do optymalizacji
  - zależność czasu wykonania od rozmiaru i/lub jakości danych wejściowych
  - złożoność optymistyczna, pesymistyczna, średnia

- Oszacowanie możliwych rezultatów optymalizacji
- Wykorzystanie heurystyki i doświadczenia programisty, przegląd możliwych do zastosowania optymalizacji
- Złożoność algorytmów czynnikiem utrudniającym wybór fragmentu do optymalizacji
  - zależność czasu wykonania od rozmiaru i/lub jakości danych wejściowych
  - złożoność optymistyczna, pesymistyczna, średnia
  - testy i pomiary powinny być wykonywane dla różnych zestawów danych, np. z określeniem odchylenia standardowego

- Oszacowanie możliwych rezultatów optymalizacji
- Wykorzystanie heurystyki i doświadczenia programisty, przegląd możliwych do zastosowania optymalizacji
- Złożoność algorytmów czynnikiem utrudniającym wybór fragmentu do optymalizacji
  - zależność czasu wykonania od rozmiaru i/lub jakości danych wejściowych
  - złożoność optymistyczna, pesymistyczna, średnia
  - testy i pomiary powinny być wykonywane dla różnych zestawów danych, np. z określeniem odchylenia standardowego

- Zależność od translatora i konkretnego języka programowania (pseudokod)

- Zależność od translatora i konkretnego języka programowania (pseudokod)
- *Zwijanie* - wykonywanie czynności na etapie kompilacji, np. inicjalizacja zmiennych, wyliczanie wartości wyrażeń zawierających wyłącznie stałe, użycie dyrektyw kompilacji warunkowej.

- Zależność od translatora i konkretnego języka programowania (pseudokod)
- *Zwijanie* - wykonywanie czynności na etapie kompilacji, np. inicjalizacja zmiennych, wyliczanie wartości wyrażeń zawierających wyłącznie stałe, użycie dyrektyw kompilacji warunkowej.
- *Zmniejszanie siły operacji* - zastąpienie operacji czasochłonnych operacjami szybszymi, np. zastąpienie potęgowania mnożeniem:

- Zależność od translatora i konkretnego języka programowania (pseudokod)
- *Zwijanie* - wykonywanie czynności na etapie kompilacji, np. inicjalizacja zmiennych, wyliczanie wartości wyrażeń zawierających wyłącznie stałe, użycie dyrektyw kompilacji warunkowej.
- *Zmniejszanie siły operacji* - zastąpienie operacji czasochłonnych operacjami szybszymi, np. zastąpienie potęgowania mnożeniem:

Zmniejszanie siły operacji

```
w:=a^3;
w:=a*a*a;
w:=2*b;
w:=b+b;
```

- *Usuwanie wyrażeń nadmiarowych* - obliczanie powtarzających się wyrażeń jednorazowo z zapamiętaniem w zmiennej pomocniczej:

- *Usuwanie wyrażeń nadmiarowych* - obliczanie powtarzających się wyrażeń jednorazowo z zapamiętaniem w zmiennej pomocniczej:

Usuwanie wyrażeń nadmiarowych

```
x := sqrt(z) / sin(y) + 16;
w := sqrt(z);
u := fn( sqrt(z) / sin(y) );
```

- *Usuwanie wyrażeń nadmiarowych* - obliczanie powtarzających się wyrażeń jednorazowo z zapamiętaniem w zmiennej pomocniczej:

Usuwanie wyrażeń nadmiarowych

```
x := sqrt(z) / sin(y) + 16;
w := sqrt(z);
u := fn( sqrt(z) / sin(y) );
```

Usuwanie wyrażeń nadmiarowych - poprawka

```
p1 := sqrt(z);
p2 := p1 / sin (y);
x  := p2 + 16;
w  := p1;
u  := fn( p2 );
```

- *Usuwanie wyrażeń nadmiarowych* - obliczanie powtarzających się wyrażeń jednorazowo z zapamiętaniem w zmiennej pomocniczej:

Usuwanie wyrażeń nadmiarowych

```
x := sqrt(z) / sin(y) + 16;
w := sqrt(z);
u := fn( sqrt(z) / sin(y) );
```

Usuwanie wyrażeń nadmiarowych - poprawka

```
p1 := sqrt(z);
p2 := p1 / sin (y);
x  := p2 + 16;
w  := p1;
u  := fn( p2 );
```

Usuwanie wyrażeń nadmiarowych - poprawka 2

```
w   := sqrt(z);
pom := w / sin (y);
x   := pom + 16;
u   := fn( pom );
```

- *Usuwanie wyrażeń niezmienniczych* - jednokrotne wyliczenie przed pętlą wartości wyrażeń, które nie zależą od wartości zmiennej sterującej, ani innych zmiennych modyfikowanych w ciele pętli:

- *Usuwanie wyrażeń niezmienniczych* - jednokrotne wyliczenie przed pętlą wartości wyrażeń, które nie zależą od wartości zmiennej sterującej, ani innych zmiennych modyfikowanych w ciele pętli:

Usuwanie wyrażeń niezmienniczych

```
for i:= 1 to 100 do
 t[i] := sin(y)*t[i];
```

- *Usuwanie wyrażeń niezmienniczych* - jednokrotne wyliczenie przed pętlą wartości wyrażeń, które nie zależą od wartości zmiennej sterującej, ani innych zmiennych modyfikowanych w ciele pętli:

Usuwanie wyrażeń niezmienniczych

```
for i:= 1 to 100 do
 t[i] := sin(y)*t[i];
```

Usuwanie wyrażeń niezmienniczych - poprawka

```
pom := sin(y);
for i:= 1 to 100 do
 t[i] := pom*t[i];
```

PHP - usuwanie wyrażeń niezmienniczych

```
<?php
 include('pomiar.php');

 pomiar_start();
 ob_start();
 for ($j=1;$j<=100;$j++)
 for ($i=1;$i<=3600;$i++)
   echo 'Kąt w stopniach: '.$i.' = Kąt w radianach: '.($i*3.14/180).'<br />';
 ob_clean();
 pomiar_stop('Wariant 1 - Powtarzanie obliczeń');


 pomiar_start();
 ob_start();
 $strd=3.14/180;
 for ($j=1;$j<=100;$j++)
 for ($i=1;$i<=3600;$i++)
    echo 'Kąt w stopniach: '.$i.' = Kąt w radianach: '.($i*$strd).'<br />';
 ob_clean();
 pomiar_stop('Wariant 2 - Bez powtarzania obliczeń');

?>
```

- *Rozszczepianie* - podział pętli w przypadku, gdy wewnątrz pętli występuje warunek niezależny od zmiennej sterującej:

- *Rozszczepianie* - podział pętli w przypadku, gdy wewnątrz pętli występuje warunek niezależny od zmiennej sterującej:

Rozszczepianie

```
for i:= 1 to nmax do
 if dodawanie=true then
  a := a + n
 else
  a := a - n;
```

- *Rozszczepianie* - podział pętli w przypadku, gdy wewnątrz pętli występuje warunek niezależny od zmiennej sterującej:

Rozszczepianie

```
for i:= 1 to nmax do
 if dodawanie=true then
  a := a + n
 else
  a := a - n;
```

Rozszczepianie - poprawka

```
if dodawanie=true then
  for i:= 1 to nmax do
      a := a + n
else
  for i:= 1 to nmax do
      a := a - n;
```

- *Rozwijanie pętli* - zwiększenie kroku pętli zmniejszające liczbę sprawdzeń warunku:

- *Rozwijanie pętli* - zwiększenie kroku pętli zmniejszające liczbę sprawdzeń warunku:

Rozwijanie pętli

```
i:=0;
while (i < 300) do
begin
 a[i] := 0;
 i := i + 1;
end;
```

- *Rozwijanie pętli* - zwiększenie kroku pętli zmniejszające liczbę sprawdzeń warunku:

Rozwijanie pętli

```
i:=0;
while (i < 300) do
begin
 a[i] := 0;
 i := i + 1;
end;
```

Rozwijanie pętli - poprawka

```
i:=0;
while (i < 300) do
begin
 a[i] := 0;
 a[i+1] := 0;
 a[i+2] := 0;
 i := i + 3;
end;
```

- *Łączenie pętli* - polega na połączeniu ciał pętli o identycznych ograniczeniach zmiennej sterującej:

- *Łączenie pętli* - polega na połączeniu ciał pętli o identycznych ograniczeniach zmiennej sterującej:

Łączenie pętli

```
for i := 1 to 100 do
 t[i]:=0;
for i := 1 to 100 do
 w[i]:=0;
```

- *Łączenie pętli* - polega na połączeniu ciał pętli o identycznych ograniczeniach zmiennej sterującej:

Łączenie pętli

```
for i := 1 to 100 do
 t[i]:=0;
for i := 1 to 100 do
 w[i]:=0;
```

Łączenie pętli - poprawka

```
for i := 1 to 100 do
begin
 t[i]:=0;
 w[i]:=0;
end;
```

- *Reorganizacja pętli* - pętle zewnętrzne powinny być powtarzane możliwie najmniej razy:

- *Reorganizacja pętli* - pętle zewnętrzne powinny być powtarzane możliwie najmniej razy:

Reorganizacja pętli (startów: 2101, zamknięć: 12100)

```
for i := 1 to 100 do {start 1 raz}
  for j := 1 to 20 do {start 100 razy}
    for z := 1 to 5 do {start 2000 razy}
     t[i,j,z]:=0;
   {zamknięcie 10000 razy}
  {zamkniecie dla j 2000 razy}
{zamkniecie dla i 100 razy}
```

- *Reorganizacja pętli* - pętle zewnętrzne powinny być powtarzane możliwie najmniej razy:

Reorganizacja pętli (startów: 2101, zamknięć: 12100)

```
for i := 1 to 100 do {start 1 raz}
  for j := 1 to 20 do {start 100 razy}
    for z := 1 to 5 do {start 2000 razy}
     t[i,j,z]:=0;
   {zamknięcie 10000 razy}
  {zamkniecie dla j 2000 razy}
{zamkniecie dla i 100 razy}
```

Reorganizacja pętli - popr. (startów: 106, zamkn.: 10105)

```
for z := 1 to 5 do {start 1 raz}
  for j := 1 to 20 do {start 5 razy}
    for i := 1 to 100 do {start 100 razy}
     t[i,j,z]:=0;
    {zamknięcie 10000 razy}
  {zamkniecie dla j 100 razy}
{zamkniecie dla z 5 razy}
```

PHP - reorganizacja pętli

```
<?php

include('pomiar.php');

pomiar_start();
for($i= 1;$i<=1000;$i++)
 for($j=1;$j<=200;$j++)
  for($z=1;$z<=5;$z++)
    $t[i][j][z]=0;
pomiar_stop('Wariant 1 - duze przebiegi w zewn. petli');

pomiar_start();
for($z=1;$z<=5;$z++)
 for($j=1;$j<=200;$j++)
  for($i= 1;$i<=1000;$i++)
    $t[i][j][z]=0;
pomiar_stop('Wariant 2 - male przebiegi w zewn. petli');
?>
```

- *Optymalizacja instrukcji warunkowych:*

- *Optymalizacja instrukcji warunkowych:*
  - dla rozłącznych warunków użycie członu *else*

- *Optymalizacja instrukcji warunkowych:*
  - dla rozłącznych warunków użycie członu *else*
  - dla translatora sprawdzającego warunki złożone tylko do momentu, w którym znana jest wartość wyrażenia:

- *Optymalizacja instrukcji warunkowych:*
  - dla rozłącznych warunków użycie członu *else*
  - dla translatora sprawdzającego warunki złożone tylko do momentu, w którym znana jest wartość wyrażenia:
    - dla operatora *oraz/and* najpierw człony zwykle fałszywe

- *Optymalizacja instrukcji warunkowych:*
  - dla rozłącznych warunków użycie członu *else*
  - dla translatora sprawdzającego warunki złożone tylko do momentu, w którym znana jest wartość wyrażenia:
    - dla operatora *oraz/and* najpierw człony zwykle fałszywe
    - dla operatora *lub/or* najpierw człony zwykle prawdziwe

- *Optymalizacja instrukcji warunkowych:*
  - dla rozłącznych warunków użycie członu *else*
  - dla translatora sprawdzającego warunki złożone tylko do momentu, w którym znana jest wartość wyrażenia:
    - dla operatora *oraz/and* najpierw człony zwykle fałszywe
    - dla operatora *lub/or* najpierw człony zwykle prawdziwe

Optymalizacja instrukcji warunkowych

```
if x = 1 then p1;
if x = 2 then p2;
if x = 2 then p3;
```

- *Optymalizacja instrukcji warunkowych:*
  - dla rozłącznych warunków użycie członu *else*
  - dla translatora sprawdzającego warunki złożone tylko do momentu, w którym znana jest wartość wyrażenia:
    - dla operatora *oraz/and* najpierw człony zwykle fałszywe
    - dla operatora *lub/or* najpierw człony zwykle prawdziwe

Optymalizacja instrukcji warunkowych

```
if x = 1 then p1;
if x = 2 then p2;
if x = 2 then p3;
```

Optymalizacja instrukcji warunkowych - poprawka

```
if x = 1 then p1
 else
  if x = 2 then p2
   else
    if x = 2 then p3;
```

- *Odwołania do elementów tablic* – podstawienie pod zmienną prostą wielokrotnie wykorzystywanego elementu tablicy:

- *Odwołania do elementów tablic* – podstawienie pod zmienną prostą wielokrotnie wykorzystywanego elementu tablicy:

Odwołania do elementów tablic

```
x := t[i] + t[i] + t[i];
for i:= 1 to 100 do
 t[i] := w[ k + 2*j ];
```

- *Odwołania do elementów tablic* – podstawienie pod zmienną prostą wielokrotnie wykorzystywanego elementu tablicy:

Odwołania do elementów tablic

```
x := t[i] + t[i] + t[i];
for i:= 1 to 100 do
 t[i] := w[ k + 2*j ];
```

Odwołania do elementów tablic - poprawka

```
pi := t[i]; x := pi + pi + pi;
wp := w[ k + 2*j ];
for i:= 1 to 100 do
 t[i] := wp;
```

- *Odwołania do pól rekordów/struktur/obiektów* – duża zależność od translatora.

- *Odwołania do pól rekordów/struktur/obiektów* – duża zależność od translatora.
- *Rozwijanie funkcji* – umieszczenie ciała funkcji w miejsce jej wywołania (jeśli wywoływanie następuje w niewielu miejscach).

- *Odwołania do pól rekordów/struktur/obiektów* – duża zależność od translatora.
- *Rozwijanie funkcji* – umieszczenie ciała funkcji w miejsce jej wywołania (jeśli wywoływanie następuje w niewielu miejscach).
- *Optymalizacje translatora* – optymalizacja kodu w opcjach kompilatora, konieczność ponownego testowania po włączeniu optymalizacji.

- *Odwołania do pól rekordów/struktur/obiektów* – duża zależność od translatora.
- *Rozwijanie funkcji* – umieszczenie ciała funkcji w miejsce jej wywołania (jeśli wywoływanie następuje w niewielu miejscach).
- *Optymalizacje translatora* – optymalizacja kodu w opcjach kompilatora, konieczność ponownego testowania po włączeniu optymalizacji.
- Negatywny wpływ na przejrzystość kodu – niektóre z optymalizacji.

- operacje wejścia-wyjścia (dla języków skryptowych, webowych - np. PHP)

- operacje wejścia-wyjścia (dla języków skryptowych, webowych - np. PHP)

PHP - różne metody przetwarzania tekstu

```
<?php
 include('pomiar.php');

 pomiar_start();
 $tab=array();
 for ($i=0;$i<100000;$i++) $tab[]="Zawartość zmiennej $i <br />";
 pomiar_stop('Wariant 1 - Cudzysłowy');

 pomiar_start();
 $tab=array();
 for ($i=0;$i<100000;$i++) $tab[]='Zawartość zmiennej '.$i.' <br />';
 pomiar_stop('Wariant 2 - Apostrofy');

?>
```

PHP - funkcje wyjściowe dla tekstów

```php
<?php
 include('pomiar.php');

 pomiar_start();
 ob_start();
 for ($i=0;$i<100000;$i++) printf('Zawartość zmiennej '.$i.'<br />');
 ob_clean();
 pomiar_stop('Wariant 1 - printf');

 pomiar_start();
 ob_start();
 for ($i=0;$i<100000;$i++) print('Zawartość zmiennej '.$i.'<br />');
 ob_clean();
 pomiar_stop('Wariant 2 - print');


 pomiar_start();
 ob_start();
 for ($i=0;$i<100000;$i++) echo('Zawartość zmiennej '.$i.'<br />');
 ob_clean();
 pomiar_stop('Wariant 3 - echo');

?>
```

- buforowanie operacji we/wy, mapowanie plików w pamięci operacyjnej

- buforowanie operacji we/wy, mapowanie plików w pamięci operacyjnej
- narzut wprowadzany przez system operacyjny/inne oprogramowanie (np. antywirusowe)

- buforowanie operacji we/wy, mapowanie plików w pamięci operacyjnej
- narzut wprowadzany przez system operacyjny/inne oprogramowanie (np. antywirusowe)
- obciążenie wynikające z wielozadaniowości i wielodostępu – wpływ niedeterministyczny

- buforowanie operacji we/wy, mapowanie plików w pamięci operacyjnej
- narzut wprowadzany przez system operacyjny/inne oprogramowanie (np. antywirusowe)
- obciążenie wynikające z wielozadaniowości i wielodostępu – wpływ niedeterministyczny
- działanie pamięci wirtualnej i dynamicznej gospodarki pamięcią (odśmiecacz - *garbage collector* w C#, Java)

- buforowanie operacji we/wy, mapowanie plików w pamięci operacyjnej
- narzut wprowadzany przez system operacyjny/inne oprogramowanie (np. antywirusowe)
- obciążenie wynikające z wielozadaniowości i wielodostępu – wpływ niedeterministyczny
- działanie pamięci wirtualnej i dynamicznej gospodarki pamięcią (odśmiecacz - *garbage collector* w C#, Java)

- listy wbudowane w bibliotekę standardową ArrayList

- listy wbudowane w bibliotekę standardową ArrayList
- listy oparte na Generykach (Templates) `List<T>`

- listy wbudowane w bibliotekę standardową ArrayList
- listy oparte na Generykach (Templates) `List<T>`
- tablice zmiennych prostych

- listy wbudowane w bibliotekę standardową ArrayList
- listy oparte na Generykach (Templates) `List<T>`
- tablice zmiennych prostych
- tablice dynamiczne (o ile istnieją w danym języku)

- listy wbudowane w bibliotekę standardową ArrayList
- listy oparte na Generykach (Templates) `List<T>`
- tablice zmiennych prostych
- tablice dynamiczne (o ile istnieją w danym języku)
- inne specyficzne typy danych (np. Variant, typy anonimowe var)

- listy wbudowane w bibliotekę standardową ArrayList
- listy oparte na Generykach (Templates) `List<T>`
- tablice zmiennych prostych
- tablice dynamiczne (o ile istnieją w danym języku)
- inne specyficzne typy danych (np. Variant, typy anonimowe var)

Dziękuję za uwagę

Następny temat:
optymalizacja pamięciowa programów