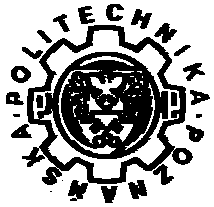


Zestaw 1

1a. Licznik instrukcji dla języka C/C++

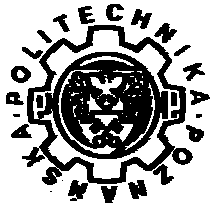
Napisz program, który będzie podawał liczbę instrukcji warunkowych „if” oraz instrukcji pętli „while” w kodzie programu w języku C/C++.

Zasadę działania programu ilustruje poniższy przykład:

```
void foo(int i, short s, char c)
{
  if (s == c)
  {
    if ( (c*3 < 20) && ((c % 3) == 0 ) )
    {
      return;
    }

    i = 2& 3;

    switch (i)
    {
      case 0: s*=i;
              break;
      case 1: s=i;
              break;
    }
  }

  else
  {
    printf("Else...");
  }

  while (i > s)
  {
    printf("i: %d", i);
    i++;
  }
}
```

W programie mamy dwie instrukcje warunkowe „if” oraz jedną instrukcję pętli „while”. Zatem odpowiedź programu powinna mieć postać:

```
Instrukcji "if": 2
Instrukcji "while": 1
Lacznie instrukcji "if" i "while": 3
```



Zestaw 1

1b. Złożoność McCabe'a dla języka C/C++

Jedną z metod oceny złożoności oprogramowania jest miara McCabe'a. Zdefiniowana przez niego 1976 roku zależność, nazwana złożonością McCabe'a (lub złożonością cyklomatyczną) definiowana jest dla modułu oprogramowania. Aby ją obliczyć należy wyznaczyć, w oparciu o graf przepływu sterowania w danym module, liczbę niezależnych ścieżek. Bardziej formalnie, złożoność McCabe'a można wyliczyć na podstawie wzoru:

$$ZC = W + 1$$

gdzie odpowiednio ZC oznacza złożoność cyklomatyczną, zaś W jest liczbą rozgałęzień (warunków) w grafie przepływu.

Zasady obliczania złożoności McCabe'a ilustrują poniższe programy:

```
/* Kod algorytmu Euklidesa obliczania NWD */
/* Zakładamy, że m i n są większe od 0 */
int euclid(int m, int n)
{
  int r;

  if (n > m)                                    // W1
  {
    r=m;
    m=n;
    n=r;
  }

  r = m % n;

  while (r != 0)                                // W2
  {
    m=n;
    n=r;
    r=m % n;
  }

  return n;
}
```

```
/* Bardziej złożony przykład */

void foo(int i, short s, char c)
{
  if (s == c)                                   // W1
  {
    if ( (c*3 < 20) && ((c % 3) == 0 )          // W2, W3
    {
      return;
    }

    i = 2& 3;

    switch (i)
    {
      case 0: s*=i;                             // W4
              break;
      case 1: s=i;                              // W5
              break;
    }
  }

  else
  {
    printf("else");
  }
}
```

W programie pierwszym są dwa warunki: jeden związany z instrukcją „if" (W1), a drugi z instrukcją „while" (W2). Zatem: ZC = 2 + 1 = 3.

Odpowiedź programu powinna mieć postać:

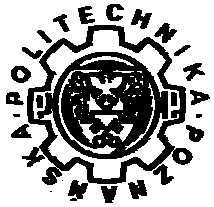
```
Zlozonosc McCabe'a: 3
```

W nieco bardziej złożonym przypadku przedstawionym po prawiej stronie, druga instrukcja „if" składa się z dwóch warunków połączonych operatorem koniuknkcji (&&). Dlatego łączenie mamy pięć warunków, a nie – jak by się mogło wydawać – cztery. Zatem ZC = 5 + 1 = 6;

Odpowiedź programu powinna mieć postać:

```
Zlozonosc McCabe'a: 6
```

Napisz program, obliczający złożoność McCabe'a dla programów w języku C/C++.



Zestaw 2

2a. Licznik instrukcji dla języka Pascal

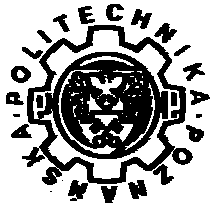
Napisz program, który będzie podawał liczbę instrukcji warunkowych „if” oraz instrukcji pętli „while” w kodzie programu w języku Pascal.

Zasadę działania programu ilustruje poniższy przykład:

```
procedure foo(i : Integer; s : Integer; c : Char);
begin
  if (s = i)
  begin
    if ( (c*3 < 20) and ((c mod 3) == 0 ) )
    begin
      exit;
    end;

    case i of
      0: s:=s*i;
      1: s:=i;
    end;

  end;

  else
  begin
    writeln('Else...');
  end;

  while (i > s) do
  begin
    writeln('i: ');
    write(i);
    i:=i+1;
  end;
end;
```

W programie mamy dwie instrukcje warunkowe „if” oraz jedną instrukcję pętli „while”. Zatem odpowiedź programu powinna mieć postać:

```
Instrukcji "if": 2
Instrukcji "while": 1
Lacznie instrukcji "if" i "while": 3
```



Zestaw 2

2b. Złożoność McCabe'a dla języka Pascal

Jedną z metod oceny złożoności oprogramowania jest miara McCabe'a. Zdefiniowana przez niego 1976 roku zależność, nazwana złożonością McCabe'a (lub złożonością cyklomatyczną) definiowana jest dla modułu oprogramowania. Aby ją obliczyć należy wyznaczyć, w oparciu o graf przepływu sterowania w danym module, liczbę niezależnych ścieżek. Bardziej formalnie, złożoność McCabe'a można wyliczyć na podstawie wzoru:

$$ZC = W + 1$$

gdzie odpowiednio ZC oznacza złożoność cyklomatyczną, zaś W jest liczbą rozgałęzień (warunków) w grafie przepływu.

Zasady obliczania złożoności McCabe'a ilustrują poniższe programy:

```
{ Kod algorytmu Euklidesa obliczania NWD }
{ Zakładamy, że m i n są większe od 0 }
function euclid(m : Integer; n : Integer) : Integer;

var r : Integer;
begin

  if (n > m)                                    {W1}
  begin
    r:=m;
    m:=n;
    n:=r;
  end;

  r := m mod n;

  while (r <> 0) do                             {W2}
  begin
    m:=n;
    n:=r;
    r:=m mod n;
  end;

  euclid := n;
end;
```

```
{ Bardziej złożony przykład }

procedure foo(i : Integer; s : Integer; c : Char) :
Integer;
begin
  if (s = i)                                    {W1}
  begin
    if ( (c*3 < 20) and ((c mod 3) == 0 ) ){W2, W3}
    begin
      exit;
    end;

    case i of
      0: s:=s*i;                                {W4}
      1: s:=i;                                  {W5}
    end;

  end;

  else
  begin
    writeln('Else...');
  end;

  while (i > s) do                               {W6}
  begin
    writeln('i: ');
    write(i);
    i:=i+1;
  end;
end;
```

W programie pierwszym są dwa warunki: jeden związany z instrukcją „if" (W1), a drugi z instrukcją „while" (W2). Zatem: ZC = 2 + 1 = 3.

Odpowiedź programu powinna mieć postać:

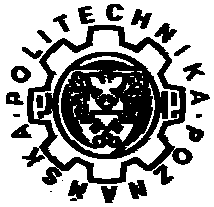
```
Zlozonosc McCabe'a: 3
```

W nieco bardziej złożonym przypadku przedstawionym po prawej stronie, druga instrukcja „if" składa się z dwóch warunków połączonych operatorem koniunkcji (and). Dlatego łączenie mamy pięć warunków, a nie – jak by się mogło wydawać – cztery. Zatem ZC = 6 + 1 = 7;

Odpowiedź programu powinna mieć postać:

```
Zlozonosc McCabe'a: 7
```

Napisz program, obliczający złożoność McCabe'a dla programów w języku Pascal.



Zestaw 3

3a. Licznik instrukcji dla języka Java

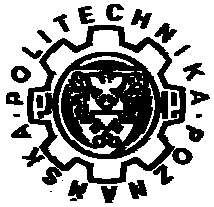
Napisz program, który będzie podawał liczbę instrukcji warunkowych „if” oraz instrukcji pętli „while” w kodzie programu w języku Java.

Zasadę działania programu ilustruje poniższy przykład:

```
void foo(int i, short s, char c)
{
  if (s == c)
  {
    if ( (c*3 < 20) && ((c % 3) == 0 ) )
    {
      return;
    }

    i = 2 & 3;

    switch (i)
    {
      case 0: s*=i;
              break;
      case 1: s=i;
              break;
    }
  }

  else
  {
    System.out.println(“Else...”);
  }

  while (i > s)
  {
    System.out.println(”i: ” + i);
    i++;
  }
}
```

W programie mamy dwie instrukcje warunkowe „if” oraz jedną instrukcję pętli „while”. Zatem odpowiedź programu powinna mieć postać:

```
Instrukcji “if”: 2
Instrukcji “while”: 1
Lacznie instrukcji “if” i “while”: 3
```



Zestaw 3

3b. Złożoność McCabe'a dla języka Java

Jedną z metod oceny złożoności oprogramowania jest miara McCabe'a. Zdefiniowana przez niego 1976 roku zależność, nazwana złożonością McCabe'a (lub złożonością cyklomatyczną) definiowana jest dla modułu oprogramowania. Aby ją obliczyć należy wyznaczyć, w oparciu o graf przepływu sterowania w danym module, liczbę niezależnych ścieżek. Bardziej formalnie, złożoność McCabe'a można wyliczyć na podstawie wzoru:

$$ZC = W + 1$$

gdzie odpowiednio ZC oznacza złożoność cyklomatyczną, zaś W jest liczbą rozgałęzień (warunków) w grafie przepływu.

Zasady obliczania złożoności McCabe'a ilustrują poniższe programy:

```
/* Kod algorytmu Euklidesa obliczania NWD */
/* Zakładamy, że m i n są większe od 0 */
int euclid(int m, int n)
{
  int r;

  if (n > m)                                    // W1
  {
    r=m;
    m=n;
    n=r;
  }

  r = m % n;

  while (r != 0)                                // W2
  {
    m=n;
    n=r;
    r=m % n;
  }

  return n;
}
```

```
/* Bardziej złożony przykład */

void foo(int i, long s, String c)
{
  if (s == c)                                   // W1
  {
    if ( (c*3 < 20) && ((c % 3) == 0 )          // W2, W3
    {
      return;
    }

    i = 2& 3;

    switch (i)
    {
      case 0: s*=i;                             // W4
              break;
      case 1: s=i;                              // W5
              break;
    }
  }

  else
  {
    System.out.println("else");
  }
}
```

W programie pierwszym są dwa warunki: jeden związany z instrukcją „if" (W1), a drugi z instrukcją „while" (W2). Zatem: ZC = 2 + 1 = 3.

Odpowiedź programu powinna mieć postać:

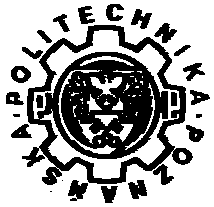
```
Zlozonosc McCabe'a: 3
```

W nieco bardziej złożonym przypadku przedstawionym po prawiej stronie, druga instrukcja „if" składa się z dwóch warunków połączonych operatorem koniuknkcji (&&). Dlatego łączenie mamy pięć warunków, a nie – jak by się mogło wydawać – cztery. Zatem ZC = 5 + 1 = 6;

Odpowiedź programu powinna mieć postać:

```
Zlozonosc McCabe'a: 6
```

Napisz program, obliczający złożoność McCabe'a dla programów w języku Java.



Zestaw 4

4a. Licznik operatorów dla języka C/C++

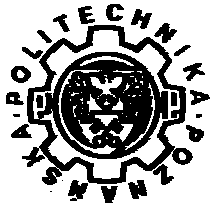
Napisz program, który będzie podawał całkowitą liczbę operatorów w kodzie programu w języku C/C++.

Zasadę działania programu ilustruje poniższy przykład:

```
/* Kod algorytmu Euklidesa obliczania NWD */
/* Zakładamy, że m i n są większe od 0 */
int euclid(int m, int n)
{
  int r;

  if (n>m)                                    // ">"
  {
    r=m;                                      // "="
    m=n;                                      // "="
    n=r;                                      // "="
  }

  r = m % n;                                  // "=", "%"

  while (r != 0)                              // "!="
  {
    m=n;                                      // "="
    n=r;                                      // "="
    r=m % n;                                  // "=", "%"
  }
  return n;
}
```

Ponieważ w powyższym programie łącza liczba operatorów wynosi 11, zatem odpowiedź programu powinna mieć postać:

```
Calkowita liczba operatorow: N1 = 11
```



Zestaw 4

4b. Złożoność Halsteada dla języka C/C++

Jedną z metod oceny złożoności oprogramowania jest miara Halsteada. Zdefiniowana przez niego 1977 złożoność operuje bezpośrednio na kodzie źródłowym programu i dotyczy operatorów i operandów znajdujących się w danym module programowym. Mówiąc bardziej dokładnie miara Halsteada oparta jest na czterech skalarnych wartościach liczbowych:

- n1 = liczba różnych operatorów
- n2 = liczba różnych operandów
- N1 = Całkowita liczba operatorów
- N2 = Całkowita liczba operandów

Na ich podstawie zdefiniowano następujące miary:

- Rozmiar programu: N=N1+N2
- Słownik programu: n = n1 + n2
- Objętość: V = N*(Log_2n)
- Złożoność: D = (n1/2)*(N2/n2)
- Pracochłonność: E = D * V

Zasady obliczania złożoności Halsteada ilustruje poniższy program:

```
/* Kod algorytmu Euklidesa obliczania NWD */
/* Zakładamy, że m i n są większe od 0 */
int euclid(int m, int n)
{
  int r;

  if (n>m)
  {
    r=m;
    m=n;
    n=r;
  }

  r = m % n;

  while (r != 0)
  {
    m=n;
    n=r;
    r=m % n;
  }
  return n;
}
```

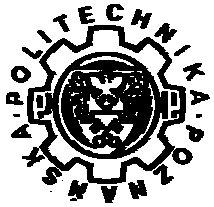
n1 = {>, =, %, !=} = 4
n2 = {n, m, r, 0} = 4
N1 = 11
N2 = 20

N = 11 + 20 = 31
n = 4 + 4 = 8
V = 31 * ($Log_2$8) = 93
D = (4/2)*(20/4) = 2 * 5 = 10
E = 10 * 93 = 930

Odpowiedź programu powinna mieć postać:

```
Liczba roznych operatorow: n1 = 4
Liczba roznych operandow: n2 = 4
Calkowita liczba operatorow: N1 = 11
Calkowita liczba operatorow: N2 = 20

Rozmiar: N = 31
Slownik: n = 8
Objetosc: V = 93
Zlozonosc: D = 9
Pracochlonnosc: E = 837
```

Napisz program obliczający wartości miar Halsteada dla języka C/C++.



Zestaw 5

5a. Licznik operatorów dla języka Pascal

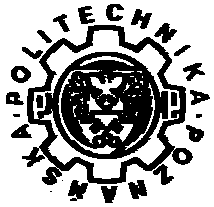
Napisz program, który będzie podawał całkowitą liczbę operatorów w kodzie programu w języku Pascal.

Zasadę działania programu ilustruje poniższy przykład:

```
{ Kod algorytmu Euklidesa obliczania NWD }
{ Zakładamy, że m i n są większe od 0 }
function euclid(m : Integer; n : Integer) :
Integer;
var r : Integer;
begin

  if (n > m)                                    {">"}
  begin
    r:=m;                                       {":="}
    m:=n;                                       {":="}
    n:=r;                                       {":="}
  end;

  r := m mod n;                                 {":=", "mod"}

  while (r <> 0) do                             {"<>"}
  begin
    m:=n;                                       {":="}
    n:=r;                                       {":="}
    r:=m mod n;                                 {":=", "mod"}
  end;
                                                {":="}
  euclid := n;
end;
```

Ponieważ w powyższym programie łącza liczba operatorów wynosi 12, zatem odpowiedź programu powinna mieć postać:

```
Calkowita liczba operatorow: N1 = 12
```



Zestaw 5

5b. Złożoność Halsteada dla języka Pascal

Jedną z metod oceny złożoności oprogramowania jest miara Halsteada. Zdefiniowana przez niego 1977 złożoność operuje bezpośrednio na kodzie źródłowym programu i dotyczy operatorów i operandów znajdujących się w danym module programowym. Mówiąc bardziej dokładnie miara Halsteada oparta jest na czterech skalarnych wartościach liczbowych:

- n1 = liczba różnych operatorów
- n2 = liczba różnych operandów
- N1 = Całkowita liczba operatorów
- N2 = Całkowita liczba operandów

Na ich podstawie zdefiniowano następujące miary:

- Rozmiar programu: N=N1+N2
- Słownik programu: n = n1 + n2
- Objętość: V = N*(Log_2n)
- Złożoność: D = (n1/2)*(N2/n2)
- Pracochłonność: E = D * V

Zasady obliczania złożoności Halsteada ilustruje poniższy program:

```
{ Kod algorytmu Euklidesa obliczania NWD }
{ Zakładamy, że m i n są większe od 0 }
function euclid(m : Integer; n : Integer) : Integer;
var r : Integer;
begin

  if (n > m)
  begin
    r:=m;
    m:=n;
    n:=r;
  end;

  r := m mod n;

  while (r <> 0) do
  begin
    m:=n;
    n:=r;
    r:=m mod n;
  end;

  euclid := n;
end;
```

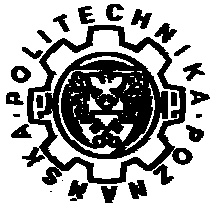
n1 = {>, :=, mod, <>} = 4
n2 = {n, m, r, 0} = 4
N1 = 12
N2 = 22

N = 12 + 22 = 34
n = 4 + 4 = 8
V = 34 * ($Log_2$8) = 102
D = (4/2)*(22/4) = 2 * 5,5 = 11
E = 11 * 93 = 1023

Odpowiedź programu powinna mieć postać:

```
Liczba roznych operatorow: n1 = 4
Liczba roznych operandow: n2 = 4
Calkowita liczba operatorow: N1 = 11
Calkowita liczba operatorow: N2 = 20

Rozmiar: N = 31
Slownik: n = 8
Objetosc: V = 93
Zlozonosc: D = 11
Pracochlonnosc: E = 1023
```

Napisz program obliczający wartości miar Halsteada dla języka Pascal.



Zestaw 6

6a. Licznik operatorów dla języka Java

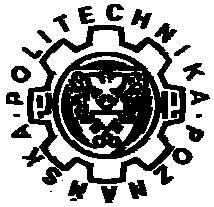
Napisz program, który będzie podawał całkowitą liczbę operatorów w kodzie programu w języku Java.

Zasadę działania programu ilustruje poniższy przykład:

```
/* Kod algorytmu Euklidesa obliczania NWD */
/* Zakładamy, że m i n są większe od 0 */
int euclid(int m, int n)
{
  int r;

  if (n>m)                                    // ">"
  {
    r=m;                                      // "="
    m=n;                                      // "="
    n=r;                                      // "="
  }

  r = m % n;                                  // "=", "%"

  while (r != 0)                              // "!="
  {
    m=n;                                      // "="
    n=r;                                      // "="
    r=m % n;                                  // "=", "%"
  }
  return n;
}
```

Ponieważ w powyższym programie łącza liczba operatorów wynosi 11, zatem odpowiedź programu powinna mieć postać:

```
Calkowita liczba operatorow: N1 = 11
```



Zestaw 6

6b. Złożoność Halsteada dla języka Java

Jedną z metod oceny złożoności oprogramowania jest miara Halsteada. Zdefiniowana przez niego 1977 złożoność operuje bezpośrednio na kodzie źródłowym programu i dotyczy operatorów i operandów znajdujących się w danym module programowym. Mówiąc bardziej dokładnie miara Halsteada oparta jest na czterech skalarnych wartościach liczbowych:

- n1 = liczba różnych operatorów
- n2 = liczba różnych operandów
- N1 = Całkowita liczba operatorów
- N2 = Całkowita liczba operandów

Na ich podstawie zdefiniowano następujące miary:

- Rozmiar programu: N=N1+N2
- Słownik programu: n = n1 + n2
- Objętość: V = N*(Log_2n)
- Złożoność: D = (n1/2)*(N2/n2)
- Pracochłonność: E = D * V

Zasady obliczania złożoności Halsteada ilustruje poniższy program:

```
/* Kod algorytmu Euklidesa obliczania NWD */
/* Zakładamy, że m i n są większe od 0 */
int euclid(int m, int n)
{
  int r;

  if (n>m)
  {
    r=m;
    m=n;
    n=r;
  }

  r = m % n;

  while (r != 0)
  {
    m=n;
    n=r;
    r=m % n;
  }
  return n;
}
```

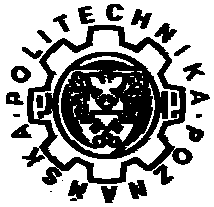
n1 = {>, =, %, !=} = 4
n2 = {n, m, r, 0} = 4
N1 = 11
N2 = 20

N = 11 + 20 = 31
n = 4 + 4 = 8
V = 31 * ($Log_2$8) = 93
D = (4/2)*(20/4) = 2 * 5 = 10
E = 10 * 93 = 930

Odpowiedź programu powinna mieć postać:

```
Liczba roznych operatorow: n1 = 4
Liczba roznych operandow: n2 = 4
Calkowita liczba operatorow: N1 = 11
Calkowita liczba operatorow: N2 = 20

Rozmiar: N = 31
Slownik: n = 8
Objetosc: V = 93
Zlozonosc: D = 9
Pracochlonnosc: E = 837
```

Napisz program obliczający wartości miar Halsteada dla języka Java.



Zestaw 7

7a. Proste formatowanie kodu dla języka C/C++

Dane są następujące założenia: w programie wejściowym każda instrukcja ma znajdować się w osobnej linii, a wszystkie komentarze mają być zastąpione komentarzami blokowymi (`/*... */`).

Napisz program formatujący plik z programem w języku C/C++ zgodnie z podanymi zasadami.

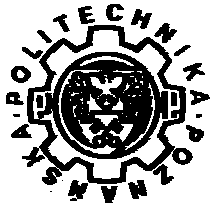
Zasadę działania programu ilustruje poniższy przykład:

Kod przed sformatowaniem:

```
void main(void)
{
  Calendar myClendar;
RandomAccessFile raf = new RandomAccessFile("moj_plik_o_bardzo_dlugiej_nazwie.txt", "RW"); // cos

 if (RandomAccessFile.FileExists("moj_plik_o_bardzo_dlugiej_nazwie.txt") && raf.length > 0) {
       try
    {
Raf.writeBytes(STR_LINE_SEPARATOR + mcrmlc.STR_XML_HEADER);
  }

  catch (Exception e)
{
    printf("Wystapily problemy podczas zapisu pliku: %s", moj_plik_o_bardzo_dlugiej_nazwie); exit(0);
}

else
{
   printf("Wystapily problemy podczas odczytu pliku. "); exit(0);
}
}
```

Kod po sformatowaniu:

```
void main(void)
{
  Calendar myClendar;
RandomAccessFile raf = new RandomAccessFile("moj_plik_o_bardzo_dlugiej_nazwie.txt", "RW"); /* cos */

 if (RandomAccessFile.FileExists("moj_plik_o_bardzo_dlugiej_nazwie.txt") && raf.length > 0) {
       try
    {
Raf.writeBytes(STR_LINE_SEPARATOR + mcrmlc.STR_XML_HEADER);
  }

  catch (Exception e)
{
    printf("Wystapily problemy podczas zapisu pliku: %s", moj_plik_o_bardzo_dlugiej_nazwie);
exit(0);
}

else
{
   printf("Wystapily problemy podzczas odczytu pliku. ");
exit(0);
}
}
```



Zestaw 7

7b. Formatowanie kodu dla języka C/C++

W linii komend dane są następujące parametry: maksymalna długość linii, wielkość pojedynczego wcięcia oraz odstępy pomiędzy blokami programu (czyli fragmentami kodu, pomiędzy ogranicznikami bloków [{}]). Ponadto zakładamy, że w programie każda instrukcja ma znajdować się w osobnej linii, ograniczniki bloków muszą być pisane są w nowej linii zgodnie z wcięciem wyższej instrukcji, wszystkie komentarze mają być zastąpione komentarzami blokowymi (`/*... */`), zaś wszystkie tabulacje powinny być zamienione na ciąg spacji o długości pojedynczego wcięcia.

Napisz program formatujący plik z programem w języku C/C++ zgodnie z podanymi zasadami.

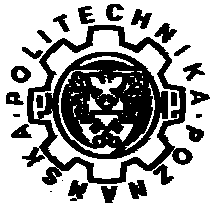
Zasadę działania programu ilustruje poniższy przykład. Dane wejściowe: długość linii: 74 znaki, wcięcie: 4 znaki, odstęp między blokami: 1 linia.

Kod przed sformatowaniem:

```
void main(void)
{
  Calendar myClendar;
RandomAccessFile raf = new RandomAccessFile(“moj_plik_o_bardzo_dlugiej_nazwie.txt”, “RW”); // cos

 if (RandomAccessFile.FileExists(“moj_plik_o_bardzo_dlugiej_nazwie.txt”) && raf.length > 0) {
       try
    {
Raf.writeBytes(STR_LINE_SEPARATOR + mcrmlc.STR_XML_HEADER);
  }

  catch (Exception e)
{
    printf(“Wystapily problemy podczas zapisu pliku: %s”, moj_plik_o_bardzo_dlugiej_nazwie); exit(0);
}

else
{
   printf(“Wystapily problemy podczas odczytu pliku. ”); exit(0);
}}
```

Kod po sformatowaniu:

```
void main(void)
{
    Calendar myClendar;
    RandomAccessFile raf = new RandomAccessFile(“moj_plik_o_bardzo_dlu” +
        “giej_nazwie”, “RW”);                                          // Przykład „złamania” ciągu
                                                                       // znaków
    if (RandomAccessFile.FileExists(“moj_plik_o_bardzo_dlugiej_nazwie”)      // Przykład „złamania”
        &&  raf.length >0)                                             // koniunkcji warunków
    {
        try
        {
            raf.writeBytes(mcrmlc.STR_LINE_SEPARATOR +
                mcrmlc.STR_LINE_SEPARATOR);                            // Przykład „złamania” zbyt
        }                                                              // długiej instrukcji

        catch (Exception e)
        {
            System.out.println(“Wystapily problemy podzczas proby zapi” +
                “su do pliku: ” + moj_plik_o_bardzo_dlugiej_nazwie);
            exit(0);                                                   // Przeniesienie instrukcji do
        }                                                              // nowej linii
    }

    else
    {
        System.out.println(“Wystapily problemy podzczas odczytu pliku.”);
        exit(0);
     }
}
```



Zestaw 8

8a. Proste formatowanie kodu dla języka Pascal

Dane są następujące założenia: w programie wejściowym każda instrukcja ma znajdować się w osobnej linii, a wszystkie komentarze mają być zastąpione komentarzami blokowymi (`{... }`).

Napisz program formatujący plik z programem w języku Pascal zgodnie z podanymi zasadami.

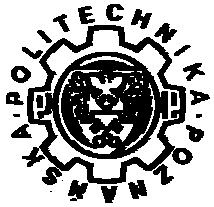
Zasadę działania programu ilustruje poniższy przykład:

Kod przed sformatowaniem:

```
procedure main;
var
  myInt : Integer;
  S : String;
begin
// cos

 if (myInt > 0 and 1 > 0)
begin
write(bardzo_dluga_nazwa_zmiennej_nr_1 + bardzo_dluga_nazwa_zmiennej_nr_2); exit(0);
 end;

else
begin
write("Wystapily jakies problemy"); exit(0);
end;
end;
```

Kod po sformatowaniu:

```
procedure main;
var
  myInt : Integer;
  S : String;
begin
{ cos }                                                                    // Przykład zamiany
                                                                           // komentarza

 if (myInt > 0 and 1 > 0)
begin
write(bardzo_dluga_nazwa_zmiennej_nr_1 + bardzo_dluga_nazwa_zmiennej_nr_2);
exit(0);                                                                   // Przykład przeniesienia
 end;                                                                      // instrukcji do nowej
                                                                           // linii
else
begin
write("Wystapily jakies problemy");
exit(0);
end;
end;
```



Zestaw 8

8b. Formatowanie kodu dla języka Pascal

W linii komend dane są następujące parametry: maksymalna długość linii, wielkość pojedynczego wcięcia oraz odstępy pomiędzy blokami programu (czyli fragmentami kodu, pomiędzy ogranicznikami bloków [`begin...end`]). Ponadto zakładamy, że w programie każda instrukcja ma znajdować się w osobnej linii, ograniczniki bloków (`{}`) muszą być pisane są w nowej linii zgodnie z wcięciem wyższej instrukcji, wszystkie komentarze mają być zastąpione komentarzami blokowymi (`{... }`), zaś wszystkie tabulacje powinny być zamienione na ciąg spacji o długości pojedynczego wcięcia.

Napisz program formatujący plik z programem w języku Pascal zgodnie z podanymi zasadami.

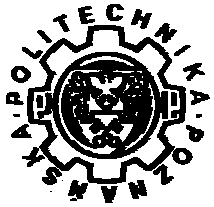
Zasadę działania programu ilustruje poniższy przykład. Dane wejściowe: długość linii: 74 znaki, wcięcie: 4 znaki, odstęp między blokami: 1 linia.

Kod przed sformatowaniem:

```
procedure main;
var
  myInt : Integer;
  S : String;
begin
// cos

 if (myInt > 0 and 1 > 0)
begin
write(bardzo_dluga_nazwa_zmiennej_nr_1 + bardzo_dluga_nazwa_zmiennej_nr_2); exit(0);
 end;

else
begin
write("Wystapily jakies problemy"); exit(0);
end;
end;
```

Kod po sformatowaniu:

```
procedure main;
var
    myInt : Integer;                                          // Wcięcie
    S : String;
begin
    { cos }                                                   // Przykład zamiany
                                                              // komentarza
    if (myInt > 0 and 1 > 0)
    begin
        write(bardzo_dluga_nazwa_zmiennej_nr_1 +              // Przykład „złamania" zbyt
            bardzo_dluga_nazwa_zmiennej_nr_2);                // długich instrukcji
        exit(0);
    end;

    else
    begin
        write("Wystapily jakies problemy");
        exit(0);                                              // Przykład przeniesienia
     end;                                                     // instrukcji do nowej
end;                                                          // linii
```



Zestaw 9

9a. Proste formatowanie kodu dla języka Java

Dane są następujące założenia: w programie wejściowym każda instrukcja ma znajdować się w osobnej linii, a wszystkie komentarze mają być zastąpione komentarzami blokowymi (`{... }`).

Napisz program formatujący plik z programem w języku Java zgodnie z podanymi zasadami.

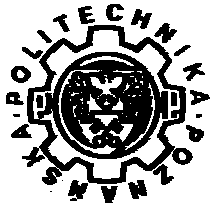
Zasadę działania programu ilustruje poniższy przykład:

Kod przed sformatowaniem:

```
void main(void)
{
  Calendar myClendar;
RandomAccessFile raf = new RandomAccessFile("moj_plik_o_bardzo_dlugiej_nazwie.txt", "RW");
// cos
 If (RandomAccessFile.FileExists("moj_plik_o_bardzo_dlugiej_nazwie.txt") && raf.length >0) {
       Try
    {
Raf.writeBytes(mcrmlc.STR_LINE_SEPARATOR + mcrmlc.STR_LINE_SEPARATOR);
  }

  catch (Exception e)
{
    System.out.println("Wystapily problemy podzczas proby zapisu do pliku: " +
moj_plik_o_bardzo_dlugiej_nazwie); exit(0);
}

else
{
   System.out.println("Wystapily problemy podzczas odczytu pliku. "); exit(0);
}
}
```

Kod po sformatowaniu:

```
void main(void)
{
  Calendar myClendar;
RandomAccessFile raf = new RandomAccessFile("moj_plik_o_bardzo_dlugiej_nazwie.txt", "RW");
/* cos */                                                                          // Zmiana
 If (RandomAccessFile.FileExists("moj_plik_o_bardzo_dlugiej_nazwie.txt") && raf.length >0) {   // komentarza
       Try
    {
Raf.writeBytes(mcrmlc.STR_LINE_SEPARATOR + mcrmlc.STR_LINE_SEPARATOR);
  }

  catch (Exception e)
{
    System.out.println("Wystapily problemy podzczas proby zapisu do pliku: " +
moj_plik_o_bardzo_dlugiej_nazwie);
exit(0);                                                                         //Przeniesienie
}                                                                                // instrukcji
                                                                                 // do nowej
else                                                                             // linii
{
   System.out.println("Wystapily problemy podzczas odczytu pliku. ");
exit(0);
}
}
```



Zestaw 9

9b. Formatowanie kodu dla języka Java

W linii komend dane są następujące parametry: maksymalna długość linii, wielkość pojedynczego wcięcia oraz odstępy pomiędzy blokami programu. Ponadto zakładamy, że w programie każda instrukcja ma znajdować się w osobnej linii, ograniczniki bloków (`{}`) muszą być pisane są w nowej linii zgodnie z wcięciem wyższej instrukcji, wszystkie komentarze mają być zastąpione komentarzami blokowymi (`/*... */`), zaś wszystkie tabulacje powinny być zamienione na ciąg spacji o długości pojedynczego wcięcia.

Napisz program formatujący plik z programem w języku Java zgodnie z podanymi zasadami.

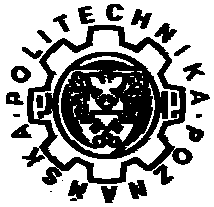
Zasady działania programu ilustruje poniższy przykład. Dane wejściowe: długość linii: 74 znaki, wcięcie: 4 znaki, odstęp między blokami: 1 linia,

Kod przed sformatowaniem:

```
void main(void)
{
  Calendar myClendar;
RandomAccessFile raf = new RandomAccessFile("moj_plik_o_bardzo_dlugiej_nazwie.txt", "RW");
// cos
 If (RandomAccessFile.FileExists("moj_plik_o_bardzo_dlugiej_nazwie.txt") && raf.length > 0) {
       Try
    {
Raf.writeBytes(mcrmlc.STR_LINE_SEPARATOR + mcrmlc.STR_LINE_SEPARATOR);
  }

  catch (Exception e)
{
    System.out.println("Wystapily problemy podzczas proby zapisu do pliku: " + plik_o_ nazwie); exit(0); }

else
{
   System.out.println("Wystapily problemy podzczas odczytu pliku. "); exit(0);
}}
```

Kod po sformatowaniu:

```
void main(void)
{
    Calendar myClendar;
    RandomAccessFile raf = new RandomAccessFile("moj_plik_o_bardzo_dlu" +
        "giej_nazwie", "RW");                                          // Przykład „złamania"
                                                                       // ciągu znaków
/* cos */
    if (RandomAccessFile.FileExists("moj_plik_o_bardzo_dlugiej_nazwie")      // Przykład „złamania"
        &&  raf.length >0)                                             // warunków
    {
        Try
        {
            raf.writeBytes(mcrmlc.STR_LINE_SEPARATOR +
                mcrmlc.STR_LINE_SEPARATOR);                            // Przykład „złamania"
        }                                                              // zbyt długiej instrukcji

        catch (Exception e)
        {
            System.out.println("Wystapily problemy podzczas proby zapi" +
                "su do pliku: " + plik_o_ nazwie);
            exit(0);
        }

    }

    else
    {
        System.out.println("Wystapily problemy podzczas odczytu pliku.");
        exit(0);
     }
}
```



Zestaw 10

10a. Generator opisów plików źródłowych języka C/C++

Napisz program, który, jeśli to konieczne, doda do kodu programu w języku C/C++ szablony komentarzy dotyczących programu. Komentarze powinny być dodawane według następującego schematu:

a) Na początku pliku powinien znaleźć się komentarz obejmujący: nazwę pliku (@file), nazwę projektu(@project), opis zawartości pliku(@content), wersję(@version), autora(@author) oraz informacje o prawach autorskich(@copyright).

Dodawane komentarze powinny być ujęte w znaki `/** */`, przy czym każda linia powinna zaczynać się od znaku `*`.

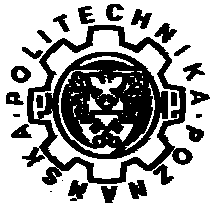
Zasadę działania programu ilustruje poniższy przykład:

Wejściowy kod źródłowy

```
void main int argc, char* argv[])
{
  int a = addOne(5);
  PrintInt("a:", a);
}

int addOne(int counter)
{
  return counter + 1;
}

void printInt(char* s, int i)
{
  printf("%s%d", i);
}
```

Wyjściowy kod źródłowy

```
/**
 * @file: Program.cpp
 * @project: nazwa projektu
 * @content: opis zawartosci pliku
 * @version: x.x, 2002-05-14
 *
 * @author: autor
 * @copyright: Copyright (c) 2002 by autor. Wszystkie
 * prawa zastrzezone.
 */

 void main int argc, char* argv[])
 {
   int a = addOne(5);
   PrintInt("a:", a);
 }

 int addOne(int counter)
 {
   return counter + 1;
 }

 void printInt(char* s, int i)
 {
   printf("%s%d", i);
 }
```



Zestaw 10

10b. Generator szablonów komentarzy dla języka C/C++

Napisz program, który, jeśli to konieczne, doda do kodu programu w języku C/C++ szablony komentarzy dotyczących programu oraz specyfikacji metod (typ wartości wynikowej, parametry wejściowe, powiązania z innymi metodami tego samego modułu). Komentarze powinny być dodawane według następującego schematu:

a) Na początku pliku powinien znaleźć się komentarz obejmujący: nazwę pliku (@file), nazwę projektu(@project), opis zawartości pliku(@content), wersję(@version), autora(@author), oraz informacje o prawach autorskich(@copyright).
b) Przed każdą metodą powinien znaleźć się komentarz obejmujący: parametry (@param), typ wartości wynikowej (@return), powiązania z innymi metodami zdefiniowanymi w tym samym pliku (@see).

Dodawane komentarze powinny być ujęte w znaki `/** */`, przy czym każda linia powinna zaczynać się od znaku `*`.

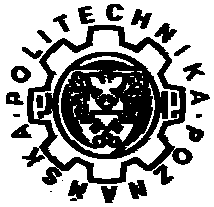
Zasadę działania programu ilustruje poniższy przykład:

Wejściowy kod źródłowy

```
void main (int argc, char* argv[])
{
  int a = addOne(5);
  PrintInt("a:", a);
}

int addOne(int counter)
{
  return counter + 1;
}

void printInt(char* s, int i)
{
  printf("&s%d", i);
}
```

Wyjściowy kod źródłowy

```
/**
 * @file: Program.cpp
 * @project: nazwa projektu
 * @content: opis zawartosci pliku
 * @version: x.x, 2002-05-14
 *
 * @author: autor
 * @copyright: Copyright (c) 2002 by autor. Wszystkie
 * prawa zastrzezone.
 */

/**
 * @param argc opis
 * @param argv opis
 * @see addOne
 * @see printInt
 */
void main (int argc, char* argv[])
 {
    int a = addOne(5);
    PrintInt("a:", a);
  }

/**
 * @param counter opis
 * @return opis
 * /
int addOne(int counter)
{
  return counter + 1;
}

/**
 * @param s opis
 * @param i opis
 */
void printInt(char* s, int i)
{
  printf("&s%d", i);
}
```



Zestaw 11

11a. Generator opisów plików źródłowych języka Pascal

Napisz program, który, jeśli to konieczne, doda do kodu programu w języku Pascal szablony komentarzy dotyczących programu. Komentarze powinny być dodawane według następującego schematu:

a) Na początku pliku powinien znaleźć się komentarz obejmujący: nazwę pliku (@file), nazwę projektu(@project), opis zawartości pliku(@content), wersję(@version), autora(@author) oraz informacje o prawach autorskich(@copyright).

Dodawane komentarze powinny być ujęte w znaki `{/** */}`, przy czym każda linia powinna zaczynać się od znaku `*`.

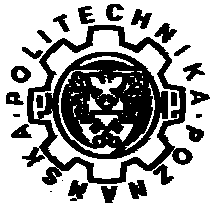
Zasadę działania programu ilustruje poniższy przykład:

Wejściowy kod źródłowy

```
program Some_Program

  procedure main;
  var a : Integer;
  begin
    a := addOne(5)
    PrintInt('a:', a);
  end;

  function addOne(counter : Integer) : Integer
  begin
    addOne := counter + 1;
  end;

  procedure printInt(s : String, i : Integer)
  {
    writeln(s);
    write(i);
  }

begin
  main;
end.
```

Wyjściowy kod źródłowy

```
{/**
 * @file: Some_Program.pas
 * @project: nazwa projektu
 * @content: opis zawartosci pliku
 * @version: x.x, 2002-05-14
 *
 * @author: autor
 * @copyright: Copyright (c) 2002 by autor.
 * Wszystkie prawa zastrzezone.
 */}

program Some_Program

  procedure main;
  var a : Integer;
  begin
    a := addOne(5)
    PrintInt('a:', a);
  end;

  function addOne(counter : Integer) : Integer
  begin
    addOne := counter + 1;
  end;

  procedure printInt(s : String, i : Integer)
  {
    writeln(s);
    write(i);
  }

begin
  main;
end.
```



Zestaw 11

11b. Generator szablonów komentarzy dla języka Pascal

Napisz program, który, jeśli to konieczne, doda do kodu programu w języku Pascal szablony komentarzy dotyczących programu oraz specyfikacji metod (typ wartości wynikowej, parametry wejściowe, powiązania z innymi metodami tego samego modułu). Komentarze powinny być dodawane według następującego schematu:

a) Na początku pliku powinien znaleźć się komentarz obejmujący: nazwę pliku (@file), nazwę projektu(@project), opis zawartości pliku(@content), wersję(@version), autora(@author), oraz informacje o prawach autorskich(@copyright).
b) Przed każdą metodą powinien znaleźć się komentarz obejmujący: parametry (@param), typ wartości wynikowej (@return), powiązania z innymi metodami zdefiniowanymi w tym samym pliku (@see).

Dodawane komentarze powinny być ujęte w znaki `{{/** */}}`, przy czym każda linia powinna zaczynać się od znaku `*`.

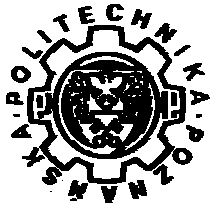
Zasadę działania programu ilustruje poniższy przykład:

Wejściowy kod źródłowy

```
program Some_Program

  procedure main;
  var a : Integer;
  begin
    a := addOne(5)
    PrintInt('a:', a);
  end;

  function addOne(counter : Integer) : Integer
  begin
    addOne := counter + 1;
  end;

  procedure printInt(s : String, i : Integer)
  {
    writeln(s);
    write(i);
  }


begin
  main;
end.
```

Wyjściowy kod źródłowy

```
{/**
 * @file: Some_Program.pas
 * @project: nazwa projektu
 * @content: opis zawartosci pliku
 * @version: x.x, 2002-05-14
 *
 * @author: autor
 * @copyright: Copyright (c) 2002 by autor.
 * Wszystkie prawa zastrzezone.
 */}

program Some_Program

  {/**
   * @see addOne
   * @see printInt
   */}
  procedure main;
  var a : Integer;
  begin
    a := addOne(5)
    PrintInt('a:', a);
  end;

  {/**
   * @param counter opis
   * @return opis
   */}
  function addOne(counter : Integer) : Integer
  begin
    addOne := counter + 1;
  end;

  {/**
   * @param s opis
   * @param i opis
   */}
  procedure printInt(s : String, i : Integer)
  {
    writeln(s);
    write(i);
  }

begin
  main;
end.
```



Zestaw 12

12a. Generator opisów plików źródłowych języka Java

Napisz program, który, jeśli to konieczne, doda do kodu programu w języku Java szablony komentarzy dotyczących programu. Komentarze powinny być dodawane według następującego schematu:

a) Na początku pliku powinien znaleźć się komentarz obejmujący: nazwę pliku (@file), nazwę projektu(@project), opis zawartości pliku(@content), wersję(@version), autora(@author) oraz informacje o prawach autorskich(@copyright).

Dodawane komentarze powinny być ujęte w znaki `{/** */}`, przy czym każda linia powinna zaczynać się od znaku `*`.

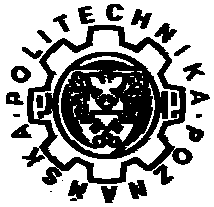
Zasadę działania programu ilustruje poniższy przykład:

Wejściowy kod źródłowy

```
import java.io.*;

public class Program
{

  void main void(String[] args)
  {
    int a = addOne(5);
    PrintInt("a:", a);
  }

  int addOne(int counter)
  {
    return counter + 1;
  }

  Sting printInt(String s, int i)
  {
    System.out.println(s + i);
    return s + String.toString(i);
  }

}
```

Wyjściowy kod źródłowy

```
/**
 * @file: Program.java
 * @project: nazwa projektu
 * @content: opis zawartosci pliku
 * @version: x.x, 2002-05-14
 *
 * @author: autor
 * @copyright: Copyright (c) 2002 by autor. Wszystkie
 * prawa zastrzezone.
 */

import java.io.*;

public class Program
{

  void main void(String[] args)
  {
    int a = addOne(5);
    PrintInt("a:", a);
  }

  int addOne(int counter)
  {
    return counter + 1;
  }

  Sting printInt(String s, int i)
  {
    System.out.println(s + i);
    return s + String.toString(i);
  }

}
```



Zestaw 12

12b. Generator szablonów komentarzy dla języka Java

Napisz program, który, jeśli to konieczne, doda do kodu programu w języku Java szablony komentarzy dotyczących programu oraz specyfikacji metod (typ wartości wynikowej, parametry wejściowe, powiązania z innymi metodami tego samego modułu). Komentarze powinny być dodawane według następującego schematu:

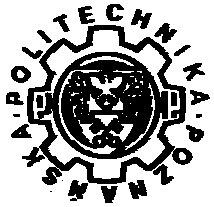
a) Na początku pliku powinien znaleźć się komentarz obejmujący: nazwę pliku (@file), nazwę projektu(@project), opis zawartości pliku(@content), wersję(@version), autora(@author) oraz informacje o prawach autorskich(@copyright)
b) Przed każdą metodą powinien znaleźć się komentarz obejmujący: parametry (@param), opis wartości wynikowej (@return), powiązania z innymi metodami (@see)

Dodawane komentarze powinny być ujęte w znaki `/** */`, przy czym każda linia powinna zaczynać się od znaku `*`.

Zasady działania programu ilustruje poniższy przykład:

```
import java.io.*;

public class Program
{

  void main void(String[] args)
  {
    int a = addOne(5);
    PrintInt("a:", a);
  }


  int addOne(int counter)
  {
    return counter + 1;
  }

  Sting printInt(String s, int i)
  {
    System.out.println(s + i);
    return s + String.toString(i);
  }

}
```

```
/**
 * @file: Program.java
 * @project: nazwa projektu
 * @content: opis zawartosci pliku
 * @version: x.x, 2002-05-14
 *
 * @author: autor
 * @copyright: Copyright (c) 2002 by autor. Wszystkie
 * prawa zastrzezone.
 */

import java.io.*;

public class Program
{

  /**
   * @param args opis
   * @see addOne
   * @see printInt
   */
  void main void(Strings[] args)
  {
    int a = addOne(5);
    PrintInt("a:", a);
  }


  /**
   * @param counter opis
   * @return opis
   */
  int addOne(int counter)
  {
    Return counter + 1;
  }

  /**
   * @param s opis
   * @param i opis
   * @return opis
   */
  String printInt(String s, int i)
  {
    System.out.println(s + i);
    return s + String.toString(i);
  }

}
```



Zestaw 13

13a. S(imple)HTML2C

Zakładając, że na stronie nie są wykorzystywane arkusze stylów CSS, **napisz program**, który ze strony WWW w formacie HTML wyekstrahuje kod programu w języku C/C++ w taki sposób, aby był on gotowy do kompilacji. Dla ułatwienia można założyć, że kod programu zawsze znajduje się wewnątrz znaczników `<PRE> </PRE>`. Znaczniki HTML mogą być pisane **zarówno kapitalikami jak i małymi literami**.

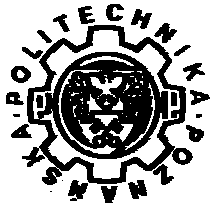
Zasady działania programu ilustruje poniższy przykład:

Kod strony HTML przed ekstrakcją:

```
<PRE>/*
 * Prosty program
 */
#include "stdafx.h"

int main(int argc, char* argv[])
{
	printf("Hello World!\n"); // Wyświetlenie ciągu znaków
	return 0;
}
</pre>
```

Kod programu po ekstrakcji:

```
/*
 * Prosty program
 */
#include "stdafx.h"

int main(int argc, char* argv[])
{
	printf("Hello World!\n"); // Wyświetlenie ciągu znaków
	return 0;
}
```



Zestaw 13

13b. HTML2C

Zakładając, że na stronie nie są wykorzystywane arkusze stylów CSS, **napisz program**, który ze strony WWW w formacie HTML wyekstrahuje kod programu w języku C/C++ w taki sposób, aby był on gotowy do kompilacji. Znaczniki HTML mogą być pisane **zarówno kapitalikami jak i małymi literami**.

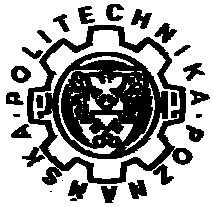
Zasady działania programu ilustruje poniższy przykład:

Kod strony HTML przed ekstrakcją:

```
<I>/*
 * Prosty program
 */ </I>
<B>#include "stdafx.h"</B>

<FONT size=3> int main(int argc, char* argv[]) <br>
{ <br>
       printf("Hello World!\n"); <PRE>// Wyświetlenie ciągu znaków </PRE><br>
       return 0; <br>
}</FONT>
```

Kod programu po ekstrakcji:

```
/*
 * Prosty program
 */
#include "stdafx.h"

int main(int argc, char* argv[])
{
       printf("Hello World!\n"); // Wyświetlenie ciągu znaków
       return 0;
}
```



Zestaw 14

14a. S(imple)HTML2Pascal

Zakładając, że na stronie nie są wykorzystywane arkusze stylów CSS, **napisz program**, który ze strony WWW w formacie HTML wyekstrahuje kod programu w języku Pascal w taki sposób, aby był on gotowy do kompilacji. Dla ułatwienia można założyć, że kod programu zawsze znajduje się wewnątrz znaczników `<PRE> </PRE>`. Znaczniki HTML mogą być pisane **zarówno kapitalikami jak i małymi literami**.

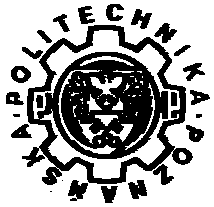
Zasady działania programu ilustruje poniższy przykład:

Kod strony HTML przed ekstrakcją:

```
<PRE>{
  Prosty program
}
uses Graph, Crt;

procedure main;
begin
       write('Hello World!\n'); // Wyświetlenie ciągu znaków
end;

begin
       main;
end.
</pre>
```

Kod programu po ekstrakcji:

```
{
  Prosty program
}
uses Graph, Crt;

procedure main;
begin
       write('Hello World!\n'); // Wyświetlenie ciągu znaków
end;

begin
       main;
end.
```



Zestaw 14

14b. HTML2Pascal

Zakładając, że na stronie nie są wykorzystywane arkusze stylów CSS, **napisz program**, który ze strony WWW w formacie HTML wyekstrahuje kod programu w języku Pascal w taki sposób, aby był on gotowy do kompilacji. Znaczniki HTML mogą być pisane **zarówno kapitalikami jak i małymi literami**.

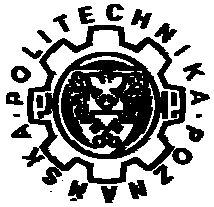
Zasady działania programu ilustruje poniższy przykład:

Kod strony HTML przed ekstrakcją:

```
<I>{
  Prosty program
}</I>
<B>uses Graph, Crt;</B> <br>

<B>procedure main;</B><br>
<FONT size=3>begin
       write('Hello World!\n'); <PRE>// Wyświetlenie ciągu znaków </PRE> <br>
end;<br><br> </font>

<FONT size=4> begin
       main;
end.
</FONT size=3>
```

Kod programu po ekstrakcji:

```
{
  Prosty program
}
uses Graph, Crt;

procedure main;
begin
       write('Hello World!\n'); // Wyświetlenie ciągu znaków
end;

begin
       main;
end.
```



Zestaw 15

15a. S(imple)HTML2Java

Zakładając, że na stronie nie są wykorzystywane arkusze stylów CSS, **napisz program**, który ze strony WWW w formacie HTML wyekstrahuje kod programu w języku Java w taki sposób, aby był on gotowy do kompilacji. Dla ułatwienia można założyć, że kod programu zawsze znajduje się wewnątrz znaczników `<PRE> </PRE>`. Znaczniki HTML mogą być pisane **zarówno kapitalikami jak i małymi literami**.

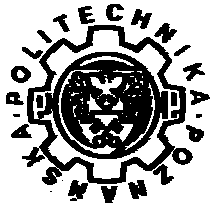
Zasady działania programu ilustruje poniższy przykład:

Kod strony HTML przed ekstrakcją:

```
<PRE>/**
 * Prosty program.
 */
class HelloWorldApp {
       public static void main(String[] args)
       {
              System.out.println("Hello World!"); // Wyświetlenie ciągu znaków
       }
}</pre>
```

Kod programu po ekstrakcji:

```
/**
 * Prosty program.
 */
class HelloWorldApp {
       public static void main(String[] args)
       {
              System.out.println("Hello World!"); // Wyświetlenie ciągu znaków
       }
}
```



Zestaw 15

15b. HTML2Java

Zakładając, że na stronie nie są wykorzystywane arkusze stylów CSS, **napisz program**, który ze strony WWW w formacie HTML wyekstrahuje kod programu w języku Java w taki sposób, aby był on gotowy do kompilacji. Znaczniki HTML mogą być pisane **zarówno kapitalikami jak i małymi literami**.

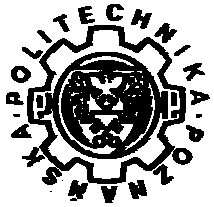
Zasady działania programu ilustruje poniższy przykład:

Kod strony HTML przed ekstrakcją:

```
<I>/**
 * Prosty program.
 */ </I>
<B>class HelloWorldApp</B> { <br>
    <FONT size=3>public static void main(String[] args) { <br>
        System.out.println("Hello World!"); <PRE>// Wyświetlenie ciągu znaków </PRE>
    }</FONT> <br>
}
```

Kod programu po ekstrakcji:

```
/**
 * Prosty program.
 */
class HelloWorldApp {
       public static void main(String[] args)
       {
              System.out.println("Hello World!"); // Wyświetlenie ciągu znaków
       }
}
```



Zestaw 16

16a. CounterProd dla YACC

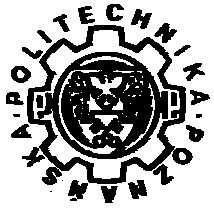
Napisz program zliczający liczbę produkcji dla programu w języku YACC.

Zasadę działania programu ilustruje poniższy przykład:

```
%{
  #include <stdio.h>
    int tab[10000];
    int cnt1=0, cnt2=0;
%}

%token number

%start S

%%

  S : Z number      { tab[cnt2++]=$2;
                      for (cnt1=0; cnt1<cnt2; cnt1++)
                      {
                          if (cnt1==0)
                            printf("(");
                            printf("%d", tab[cnt1], $2);

                          if (cnt1!=(cnt2-1))
                            printf(",");

                          if (cnt1==(cnt2-1))
                            printf(")");
                      }
                     }
    ;

  Z : Z number      { tab[cnt2++]=$2;                }
    | number        { tab[cnt2++]=$1;                }
    ;
%%
#include <stdio.h>
extern int yylineno;
yyerror (char *s)
  {
     printf("%d: %s", yylineno, s);
  }
```

W programie mamy trzy produkcje gramatyczne. Zatem odpowiedź programu powinna mieć postać:

```
Liczba produkcji gramatycznych: 3
```



Zestaw 16

16b. EffortCounter dla YACC

Napisz program zliczający dla programu w języku YACC:

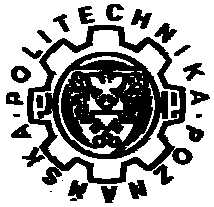
- liczbę produkcji,
- liczbę symboli nieterminalnych,
- liczbę symboli terminalnych,
- liczbę akcji semantycznych,
- liczbę pojedynczych instrukcji języka C.

Zasadę działania programu ilustruje poniższy przykład:

```
%{
  #include <stdio.h>
    int tab[10000];
    int cnt1=0, cnt2=0;
%}

%token number

%start S

%%

  S : Z number      { tab[cnt2++]=$2;
                      for (cnt1=0; cnt1<cnt2; cnt1++)
                      {
                          if (cnt1==0)
                            printf("(");
                            printf("%d", tab[cnt1], $2);

                          if (cnt1!=(cnt2-1))
                            printf(",");

                          if (cnt1==(cnt2-1))
                            printf(")");
                      }
                    }
    ;

  Z : Z number      { tab[cnt2++]=$2;                }
    | number        { tab[cnt2++]=$1;                }
    ;
%%
#include <stdio.h>
extern int yylineno;
yyerror (char *s)
  {
     printf("%d: %s", yylineno, s);
  }
```

W programie mamy trzy produkcje gramatyczne. Symbolami nieterminalymi są S oraz Z, zaś symbolem terminalnym jest w tym przypadku „number”. Ponadto możemy wyróżnić trzy akcje semantyczne (każda jest związana z jedną produkcją gramatyczną). Ponieważ pomijamy nagłówek oraz deklaracje funkcji pomocniczych, liczba instrukcji wynosi 11 (jedna pętla „for”, trzy instrukcje warunkowe „if” oraz siedem instrukcji przypisania). Zatem odpowiedź programu powinna mieć postać:

```
Liczba produkcji gramatycznych: 3
Liczba symboli nieterminalnych: 2
Liczba symboli terminalnych: 1
Liczba akcji semantycznych: 3
Liczba pojedynczych instrukcji jezyka C: 11
```



Zestaw 17

17a. CounterProd dla LEX

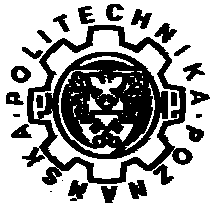
Napisz program zliczający liczbę reguł przetwarzania dla programu w języku LEX.

Zasadę działania programu ilustruje poniższy przykład:

```
%{
  int personcnt=0;
  float dig, sum=0;
  int cnt=1, IfFirst=1;
%}
id     [A-Z][a-z]+
grp    [A-Z][0-9]+
snum   [1-9][0-9]+
notint [0-9]+"."[0-9]+
%%
{snum}    { if (IfFirst==1)
                {
                  printf("LP\tNR INDEKSU\tNAZWISKO\tIMIE\t\tSUMA PUNKTOW\n\n");
                  IfFirst=0;
                  printf("%d\t%s\t\t", cnt, yytext);
                }
            else
               printf("%d\t%s\t\t", cnt, yytext);
            cnt++;
          }
{id}      {if (yyleng<=6)
               printf("%s\t\t", yytext);
           else
               printf("%s\t", yytext);}
{grp}     ;
[\ \t]    ;
{notint}  { sscanf(yytext, "%f", &dig);
            sum=sum+dig;
          }
\n        { if (sum>=1.8)
               personcnt++;
            printf("%g\n", sum);
            sum=0;
          }

%%
int yywrap (void)
   {
     printf("\nLiczba osob: %d\n", personcnt);
     return 1;
   }
```

W programie mamy sześć reguł przetwarzania związanych ze wzorcami. Zatem odpowiedź programu powinna mieć postać:

```
Liczba regul przetwarzania: 6
```



Zestaw 17

17b. EffortCounter dla LEX

Napisz program zliczający dla programu w języku LEX:

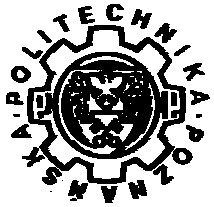
- liczbę reguł przetwarzania,
- wyrażeń regularnych,
- sumaryczną liczbę instrukcji języka C,
- liczbę niedomyślnych reguł przetwarzania.

Zasadę działania programu ilustruje poniższy przykład:

```
%{
  int personcnt=0;
  float dig, sum=0;
  int cnt=1, IfFirst=1;
%}
id     [A-Z][a-z]+
grp    [A-Z][0-9]+
snum   [1-9][0-9]+
notint [0-9]+"."[0-9]+
%%
{snum}    { if (IfFirst==1)
                {
                  printf("LP\tNR INDEKSU\tNAZWISKO\tIMIE\t\tSUMA PUNKTOW\n\n");
                  IfFirst=0;
                  printf("%d\t%s\t\t", cnt, yytext);
                }
              else
                 printf("%d\t%s\t\t", cnt, yytext);
              cnt++;
            }
{id}      {if (yyleng<=6)
                 printf("%s\t\t", yytext);
             else
                 printf("%s\t", yytext);}
{grp}     ;
[\ \t]    ;
{notint}  { sscanf(yytext, "%f", &dig);
              sum=sum+dig;
            }
\n        { if (sum>=1.8)
                 personcnt++;
              printf("%g\n", sum);
              sum=0;
            }

%%
int yywrap (void)
   {
     printf("\nLiczba osob: %d\n", personcnt);
     return 1;
   }
```

W programie mamy sześć reguł przetwarzania związanych ze wzorcami. Zdefiniowano także pięć wyrażeń regularnych (cztery jako definicje regularne, jedną [jako wzorzec] w kodzie programu. Ponadto w programie użyto piętnastu instrukcji języka C (trzy instrukcje warunkowe „if” oraz dwanaście pozostałych instrukcji, w tym przypisania, wypisania, inkrementacji, etc.). Liczba niedomyślnych reguł przetwarzania równa jest liczbie reguł i wynosi sześć. Zatem odpowiedź programu powinna mieć postać:

```
Liczba regul przetwarzania: 6
Liczba wyrazen regularnych: 5
Liczba pojedynczych instrukcji jezyka C: 15
Liczba niedomyslnych reguł przetwarzania: 6
```



Zestaw 18

18a. CounterProd dla AWK

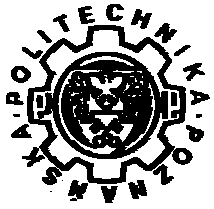
Napisz program zliczający liczbę reguł przetwarzania dla programu w języku AWK.

Zasadę działania programu ilustruje poniższy przykład:

```
BEGIN               { getline<"data"; split($0, tab, " ");
                      True=1;
                    }
$1~/^[x][x]*$/      { if (length($1)==tab[2])
                            ;
                      else
                            True=0;
                    }
$1!~/^[x][x]*$/     {True=0;}
END                 { if ((True==1) && NR==tab[1])
                            printf("O.K.!\n");
                      else
                            printf("Error!\n");}
```

W programie mamy cztery reguły przetwarzania związane ze wzorcami (przy czym dwa z nich to wzorce predefiniowane AWK: BEGIN i END). Zatem odpowiedź programu powinna mieć postać:

```
Liczba regul przetwarzania: 4
```



Zestaw 18

18b. EffortCounter dla AWK

Napisz program zliczający dla programu w języku AWK:

- liczbę reguł przetwarzania,
- wyrażeń regularnych,
- sumaryczną liczbę instrukcji języka C,
- liczbę niedomyślnych reguł przetwarzania.

Zasadę działania programu ilustruje poniższy przykład:

```
BEGIN               { getline<"data"; split($0, tab, " ");
                      True=1;
                    }
$1~/^[x][x]*$/      { if (length($1)==tab[2])
                            ;
                      else
                            True=0;
                    }
$1!~/^[x][x]*$/     {True=0;}
END                 { if ((True==1) && NR==tab[1])
                            printf("O.K.!\n");
                      else
                            printf("Error!\n");}
```

W programie mamy cztery reguły przetwarzania związane ze wzorcami, przy czym dwa z nich to wzorce predefiniowane AWK: BEGIN i END, stąd mamy tylko dwa wyrażenia regularne. Ponadto w programie użyto dziewięciu instrukcji języka C (w tym dwie instrukcje warunkowe „if", trzy instrukcje przypisania, dwie instrukcje wypisania, jedną instrukcję pustą oraz instrukcję wywołania funkcji getline). Liczba niedomyślnych reguł przetwarzania równa jest liczbie reguł i wynosi cztery. Zatem odpowiedź programu powinna mieć postać:

```
Liczba regul przetwarzania: 4
Liczba wyrazen regularnych: 2
Liczba pojedynczych instrukcji jezyka C: 9
Liczba niedomyslnych reguł przetwarzania: 4
```