Podstawy Programowania

Wykład III

Składnia wyrażeń i instrukcji, złożoność obliczeniowa, operacje wejścia i wyjścia, funkcje

Robert Muszyński
Katedra Cybernetyki i Robotyki, PWr

Zagadnienia: składnia wyrażeń i instrukcji, instrukcje warunkowe i pętli, zagadnienia złożoności obliczeniowej, operacje wejścia i wyjścia, definicja funkcji, jej argumenty, struktura programu, zasięg zmiennych.

– Skład FoilTEX –

Składnia wyrażeń

```
wyrazenie = [ wyrazenie "," ] wyrazenie-przypisania .
```

Składnia wyrażeń

```
wyrazenie = [ wyrazenie "," ] wyrazenie-przypisania .

wyrazenie-przypisania = wyrazenie-warunkowe |
      wyrazenie-jednoargumentowe
                  operator-przypisania wyrazenie-przypisania .
```

Składnia wyrażeń

```
wyrazenie = [ wyrazenie "," ] wyrazenie-przypisania .

wyrazenie-przypisania = wyrazenie-warunkowe |
      wyrazenie-jednoargumentowe
                  operator-przypisania wyrazenie-przypisania .

operator-przypisania = "=" | "*=" | "/=" | "%=" | "+=" | "-="
                       | "<<=" | ">>=" | "&=" | "^=" | "|=" .
```

Składnia wyrażeń

```
wyrazenie = [ wyrazenie "," ] wyrazenie-przypisania .

wyrazenie-przypisania = wyrazenie-warunkowe |
      wyrazenie-jednoargumentowe
                  operator-przypisania wyrazenie-przypisania .

operator-przypisania = "=" | "*=" | "/=" | "%=" | "+=" | "-="
                       | "<<=" | ">>=" | "&=" | "^=" | "|=" .

wyrazenie-jednoargumentowe =
          ( "++" | "--" | "sizeof" ) wyrazenie-jednoargumentowe
```

Składnia wyrażeń

```
wyrazenie = [ wyrazenie "," ] wyrazenie-przypisania .

wyrazenie-przypisania = wyrazenie-warunkowe |
      wyrazenie-jednoargumentowe
                  operator-przypisania wyrazenie-przypisania .

operator-przypisania = "=" | "*=" | "/=" | "%=" | "+=" | "-="
                       | "<<=" | ">>=" | "&=" | "^=" | "|=" .

wyrazenie-jednoargumentowe =
          ( "++" | "--" | "sizeof" ) wyrazenie-jednoargumentowe
        | "sizeof" "(" nazwa-typu ")"
```

Składnia wyrażeń

```
wyrazenie = [ wyrazenie "," ] wyrazenie-przypisania .

wyrazenie-przypisania = wyrazenie-warunkowe |
      wyrazenie-jednoargumentowe
                  operator-przypisania wyrazenie-przypisania .

operator-przypisania = "=" | "*=" | "/=" | "%=" | "+=" | "-="
                       | "<<=" | ">>=" | "&=" | "^=" | "|=" .

wyrazenie-jednoargumentowe =
          ( "++" | "--" | "sizeof" ) wyrazenie-jednoargumentowe
        | "sizeof" "(" nazwa-typu ")" | wyrazenie-przyrostkowe
```

Składnia wyrażeń

```
wyrazenie = [ wyrazenie "," ] wyrazenie-przypisania .

wyrazenie-przypisania = wyrazenie-warunkowe |
      wyrazenie-jednoargumentowe
                  operator-przypisania wyrazenie-przypisania .

operator-przypisania = "=" | "*=" | "/=" | "%=" | "+=" | "-="
                       | "<<=" | ">>=" | "&=" | "^=" | "|=" .

wyrazenie-jednoargumentowe =
          ( "++" | "--" | "sizeof" ) wyrazenie-jednoargumentowe
        | "sizeof" "(" nazwa-typu ")" | wyrazenie-przyrostkowe
        | operator-jednoargumentowy wyrazenie-rzutowania .
```

Składnia wyrażeń

```
wyrazenie = [ wyrazenie "," ] wyrazenie-przypisania .

wyrazenie-przypisania = wyrazenie-warunkowe |
      wyrazenie-jednoargumentowe
                  operator-przypisania wyrazenie-przypisania .

operator-przypisania = "=" | "*=" | "/=" | "%=" | "+=" | "-="
                       | "<<=" | ">>=" | "&=" | "^=" | "|=" .

wyrazenie-jednoargumentowe =
          ( "++" | "--" | "sizeof" ) wyrazenie-jednoargumentowe
        | "sizeof" "(" nazwa-typu ")" | wyrazenie-przyrostkowe
        | operator-jednoargumentowy wyrazenie-rzutowania .

operator-jednoargumentowy = "&" | "*" | "+" | "-" | "~" | "!" .
```

Składnia wyrażeń

```
wyrazenie = [ wyrazenie "," ] wyrazenie-przypisania .

wyrazenie-przypisania = wyrazenie-warunkowe |
      wyrazenie-jednoargumentowe
                  operator-przypisania wyrazenie-przypisania .

operator-przypisania = "=" | "*=" | "/=" | "%=" | "+=" | "-="
                       | "<<=" | ">>=" | "&=" | "^=" | "|=" .

wyrazenie-jednoargumentowe =
          ( "++" | "--" | "sizeof" ) wyrazenie-jednoargumentowe
        | "sizeof" "(" nazwa-typu ")" | wyrazenie-przyrostkowe
        | operator-jednoargumentowy wyrazenie-rzutowania .

operator-jednoargumentowy = "&" | "*" | "+" | "-" | "~" | "!" .

wyrazenie-rzutowania = wyrazenie-jednoargumentowe
                     "(" nazwa-typu ")" wyrazenie-rzutowania .
```

Składnia wyrażeń cd.

```
wyrazenie-przyrostkowe = wyrazenie-proste |
```

Składnia wyrażeń cd.

```
wyrazenie-przyrostkowe = wyrazenie-proste |
      wyrazenie-przyrostkowe (  "++" | "--" | "[" wyrazenie "]"
```

Składnia wyrażeń cd.

```
wyrazenie-przyrostkowe = wyrazenie-proste |
      wyrazenie-przyrostkowe (  "++" | "--" | "[" wyrazenie "]"
            | "(" [ lista-argumentow ] ")"
            | ( "." | "->" ) identyfikator ) .
```

Składnia wyrażeń cd.

```
wyrazenie-przyrostkowe = wyrazenie-proste |
      wyrazenie-przyrostkowe (  "++" | "--" | "[" wyrazenie "]"
            | "(" [ lista-argumentow ] ")"
            | ( "." | "->" ) identyfikator ) .

wyrazenie-proste = identyfikator | stala | napis
                                         | "(" wyrazenie ")" .
```

Składnia wyrażeń cd.

```
wyrazenie-przyrostkowe = wyrazenie-proste |
      wyrazenie-przyrostkowe (  "++" | "--" | "[" wyrazenie "]"
            | "(" [ lista-argumentow ] ")"
            | ( "." | "->" ) identyfikator ) .

wyrazenie-proste = identyfikator | stala | napis
                                         | "(" wyrazenie ")" .

lista-argumentow = [ lista-argumentow ] "," wyrazenie-przypisania .
```

Składnia wyrażeń cd.

```
wyrazenie-przyrostkowe = wyrazenie-proste |
      wyrazenie-przyrostkowe (  "++" | "--" | "[" wyrazenie "]"
            | "(" [ lista-argumentow ] ")"
            | ( "." | "->" ) identyfikator ) .

wyrazenie-proste = identyfikator | stala | napis
                                         | "(" wyrazenie ")" .

lista-argumentow = [ lista-argumentow ] "," wyrazenie-przypisania .

wyrazenie-warunkowe = logiczne-wyrazenie-OR
                    [ "?" wyrazenie ":" wyrazenie-warunkowe ] .
```

Składnia wyrażeń cd.

```
wyrazenie-przyrostkowe = wyrazenie-proste |
      wyrazenie-przyrostkowe (  "++" | "--" | "[" wyrazenie "]"
            | "(" [ lista-argumentow ] ")"
            | ( "." | "->" ) identyfikator ) .

wyrazenie-proste = identyfikator | stala | napis
                                         | "(" wyrazenie ")" .

lista-argumentow = [ lista-argumentow ] "," wyrazenie-przypisania .

wyrazenie-warunkowe = logiczne-wyrazenie-OR
                    [ "?" wyrazenie ":" wyrazenie-warunkowe ] .

logiczne-wyrazenie-OR = logiczne-wyrazenie-AND
                    [ "||" logiczne-wyrazenie-OR ] .
```

Składnia wyrażeń cd.

```
wyrazenie-przyrostkowe = wyrazenie-proste |
      wyrazenie-przyrostkowe (  "++" | "--" | "[" wyrazenie "]"
            | "(" [ lista-argumentow ] ")"
            | ( "." | "->" ) identyfikator ) .

wyrazenie-proste = identyfikator | stala | napis
                                         | "(" wyrazenie ")" .

lista-argumentow = [ lista-argumentow ] "," wyrazenie-przypisania .

wyrazenie-warunkowe = logiczne-wyrazenie-OR
                    [ "?" wyrazenie ":" wyrazenie-warunkowe ] .

logiczne-wyrazenie-OR = logiczne-wyrazenie-AND
                    [ "||" logiczne-wyrazenie-OR ] .

logiczne-wyrazenie-AND = wyrazenie-OR
                    [ "&&" logiczne-wyrazenie-AND ] .

wyrazenie-OR = wyrazenie-XOR [ "|" wyrazenie-OR ] ...
```

Składnia instrukcji

```
instrukcja-wyrazeniowa = [ wyrazenie ] ";" .
```

Składnia instrukcji

```
instrukcja-wyrazeniowa = [ wyrazenie ] ";" .
```

```
instrukcja = instrukcja-wyrazeniowa | instrukcja-zlozona
```

Składnia instrukcji

```
instrukcja-wyrazeniowa = [ wyrazenie ] ";" .

instrukcja = instrukcja-wyrazeniowa | instrukcja-zlozona
           | instrukcja-wyboru | instrukcja-powtarzania
           | instrukcja-etykietowana | instrukcja-skoku .
```

Składnia instrukcji

```
instrukcja-wyrazeniowa = [ wyrazenie ] ";" .

instrukcja = instrukcja-wyrazeniowa | instrukcja-zlozona
           | instrukcja-wyboru | instrukcja-powtarzania
           | instrukcja-etykietowana | instrukcja-skoku .

instrukcja-wyboru = "if (" wyrazenie ")" instrukcja
                                  [ "else" instrukcja ]
```

Składnia instrukcji

```
instrukcja-wyrazeniowa = [ wyrazenie ] ";" .

instrukcja = instrukcja-wyrazeniowa | instrukcja-zlozona
           | instrukcja-wyboru | instrukcja-powtarzania
           | instrukcja-etykietowana | instrukcja-skoku .

instrukcja-wyboru = "if (" wyrazenie ")" instrukcja
                                    [ "else" instrukcja ]
           | "switch (" wyrazenie ")" instrukcja .
```

Składnia instrukcji

```
instrukcja-wyrazeniowa = [ wyrazenie ] ";" .

instrukcja = instrukcja-wyrazeniowa | instrukcja-zlozona
           | instrukcja-wyboru | instrukcja-powtarzania
           | instrukcja-etykietowana | instrukcja-skoku .

instrukcja-wyboru = "if (" wyrazenie ")" instrukcja
                                    [ "else" instrukcja ]
           | "switch (" wyrazenie ")" instrukcja .

instrukcja-powtarzania = "while (" wyrazenie ")" instrukcja
```

Składnia instrukcji

```
instrukcja-wyrazeniowa = [ wyrazenie ] ";" .

instrukcja = instrukcja-wyrazeniowa | instrukcja-zlozona
           | instrukcja-wyboru | instrukcja-powtarzania
           | instrukcja-etykietowana | instrukcja-skoku .

instrukcja-wyboru = "if (" wyrazenie ")" instrukcja
                                    [ "else" instrukcja ]
           | "switch (" wyrazenie ")" instrukcja .

instrukcja-powtarzania = "while (" wyrazenie ")" instrukcja
                       | "do" instrukcja "while (" wyrazenie ");"
```

Składnia instrukcji

```
instrukcja-wyrazeniowa = [ wyrazenie ] ";" .

instrukcja = instrukcja-wyrazeniowa | instrukcja-zlozona
           | instrukcja-wyboru | instrukcja-powtarzania
           | instrukcja-etykietowana | instrukcja-skoku .

instrukcja-wyboru = "if (" wyrazenie ")" instrukcja
                                    [ "else" instrukcja ]
           | "switch (" wyrazenie ")" instrukcja .

instrukcja-powtarzania = "while (" wyrazenie ")" instrukcja
                       | "do" instrukcja "while (" wyrazenie ");"
  | "for ("[wyrazenie]";"[wyrazenie]";"[wyrazenie]")" instrukcja .
```

Składnia instrukcji

```
instrukcja-wyrazeniowa = [ wyrazenie ] ";" .

instrukcja = instrukcja-wyrazeniowa | instrukcja-zlozona
           | instrukcja-wyboru | instrukcja-powtarzania
           | instrukcja-etykietowana | instrukcja-skoku .

instrukcja-wyboru = "if (" wyrazenie ")" instrukcja
                                    [ "else" instrukcja ]
           | "switch (" wyrazenie ")" instrukcja .

instrukcja-powtarzania = "while (" wyrazenie ")" instrukcja
                       | "do" instrukcja "while (" wyrazenie ");"
  | "for ("[wyrazenie]";"[wyrazenie]";"[wyrazenie]")" instrukcja .

instrukcja-etykietowana = identyfikator ":" instrukcja
           | "case" wyrazenie-stale ";" instrukcja
           | "default :" instrukcja .

instrukcja-skoku = "goto" identyfikator ";" | "continue"
           | "break" | "return" [ wyrazenie ] ";" .
```

Instrukcje warunkowe – przykłady

```
if (n > 0)
  if (a > b)
    z = a;
  else
    z = b;
```

Instrukcje warunkowe – przykłady

```
if (n > 0)
  if (a > b)
    z = a;
  else
    z = b;
```

```
if (n > 0) {
  if (a > b)
    z = a;
}
else
  z = b;
```

Instrukcje warunkowe – przykłady

```
if (n > 0)
  if (a > b)
    z = a;
  else
    z = b;
```

```
if (n > 0) {
  if (a > b)
    z = a;
}
else
  z = b;
```

```
if (wyrazenie)
  instrukcja
else if (wyrazenie)
  instrukcja
else if (wyrazenie)
  instrukcja
else
  instrukcja
```

Instrukcje pętli – przykłady

Równoważność pętli `for` i `while`

```
for (wyr1; wyr2; wyr3)
  instrukcja
```

```
wyr1;
while (wyr2) {
  instrukcja
  wyr3;
}
```

Instrukcje pętli – przykłady

Równoważność pętli `for` i `while`

```
for (wyr1; wyr2; wyr3)
  instrukcja
```

```
wyr1;
while (wyr2) {
  instrukcja
  wyr3;
}
```

```
for (i = 0; i < n; i++) {
  ...
}
```

Instrukcje pętli – przykłady

Równoważność pętli `for` i `while`

```
for (wyr1; wyr2; wyr3)
  instrukcja
```

```
wyr1;
while (wyr2) {
  instrukcja
  wyr3;
}
```

```
for (i = 0; i < n; i++) {
  ...
}
```

```
for (;;) {
  ...
}
```

Instrukcje pętli – przykłady

Równoważność pętli `for` i `while`

```
for (wyr1; wyr2; wyr3)
  instrukcja
```

```
wyr1;
while (wyr2) {
  instrukcja
  wyr3;
}
```

```
for (i = 0; i < n; i++) {
  ...
}
```

goto

```
for (;;) {
  ...
}
```

Instrukcje pętli – przykłady

Równoważność pętli `for` i `while`

```
for (wyr1; wyr2; wyr3)
  instrukcja
```

```
wyr1;
while (wyr2) {
  instrukcja
  wyr3;
}
```

```
for (i = 0; i < n; i++) {
  ...
}
```

~~goto~~ !!!

```
for (;;) {
  ...
}
```

Instrukcje pętli – przykłady

Równoważność pętli `for` i `while`

```
for (wyr1; wyr2; wyr3)
  instrukcja
```

```
wyr1;
while (wyr2) {
  instrukcja
  wyr3;
}
```

```
for (i = 0; i < n; i++) {
  ...
}
```

~~goto~~ !!!

break

```
for (;;) {
  ...
}
```

Instrukcje pętli – przykłady

Równoważność pętli `for` i `while`

```
for (wyr1; wyr2; wyr3)
  instrukcja
```

```
wyr1;
while (wyr2) {
  instrukcja
  wyr3;
}
```

```
for (i = 0; i < n; i++) {
  ...
}
```

~~goto~~ !!!
break???

```
for (;;) {
  ...
}
```

Instrukcje pętli – przykłady

Równoważność pętli `for` i `while`

```
for (wyr1; wyr2; wyr3)
  instrukcja
```

```
wyr1;
while (wyr2) {
  instrukcja
  wyr3;
}
```

```
for (i = 0; i < n; i++) {
  ...
}
```

```
for (;;) {
  ...
}
```

~~goto~~ !!!

break???

continue???

return?

Instrukcja `while` – komentarze

Typowa konstrukcja

```
int main() {
  int biez, poprz = 0;
  biez = 0;
  while(biez != KONIEC) {
    scanf("%d", &biez);
    (* i wiecej pozytecznych instrukcji *)
    (* jeszcze wiecej instrukcji *)
    (* i jeszcze *)
    if (biez >= MINVAL && biez <= MAXVAL) {
      (* i kolejne pozyteczne instrukcje *)
    }
  }
  return 0;
}
```

Instrukcja `while` – komentarze

Typowa konstrukcja

```
int main() {
  int biez, poprz = 0;
  biez = 0;
  while(biez != KONIEC) {
    scanf("%d", &biez);
    (* i wiecej pozytecznych instrukcji *)
    (* jeszcze wiecej instrukcji *)
    (* i jeszcze *)
    if (biez >= MINVAL && biez <= MAXVAL) {
      (* i kolejne pozyteczne instrukcje *)
    }
  }
  return 0;
}
```

Niepolecana (trudniejsza do analizy)

```
int main() {
  int biez, poprz = 0;

  while(1) {
    scanf("%d", &biez);
    (* i wiecej pozytecznych instrukcji *)
    if (biez == KONIEC) break;
    (* jeszcze wiecej instrukcji *)
    if (biez < MINVAL || biez > MAXVAL)
      continue;
    (* i kolejne pozyteczne instrukcje *)
  }
  return 0;
}
```

Instrukcja `while` – komentarze

Typowa konstrukcja

```
int main() {
  int biez, poprz = 0;
  biez = 0;
  while(biez != KONIEC) {
    scanf("%d", &biez);
    (* i wiecej pozytecznych instrukcji *)
    (* jeszcze wiecej instrukcji *)
    (* i jeszcze *)
    if (biez >= MINVAL && biez <= MAXVAL) {
      (* i kolejne pozyteczne instrukcje *)
    }
  }
  return 0;
}
```

Niepolecana (trudniejsza do analizy)

```
int main() {
  int biez, poprz = 0;

  while(1) {
    scanf("%d", &biez);
    (* i wiecej pozytecznych instrukcji *)
    if (biez == KONIEC) break;
    (* jeszcze wiecej instrukcji *)
    if (biez < MINVAL || biez > MAXVAL)
      continue;
    (* i kolejne pozyteczne instrukcje *)
  }
  return 0;
}
```

- Warunek zakończenia pętli podajemy po słowie kluczowym `while` – umieszczenie go wewnątrz instrukcji złożonej (do tego w kilku miejscach?) ze słowem `break` utrudnia analizę programu

Instrukcja `while` – komentarze

Typowa konstrukcja

```
int main() {
  int biez, poprz = 0;
  biez = 0;
  while(biez != KONIEC) {
    scanf("%d", &biez);
    (* i wiecej pozytecznych instrukcji *)
    (* jeszcze wiecej instrukcji *)
    (* i jeszcze *)
    if (biez >= MINVAL && biez <= MAXVAL) {
      (* i kolejne pozyteczne instrukcje *)
    }
  }
  return 0;
}
```

Niepolecana (trudniejsza do analizy)

```
int main() {
  int biez, poprz = 0;

  while(1) {
    scanf("%d", &biez);
    (* i wiecej pozytecznych instrukcji *)
    if (biez == KONIEC) break;
    (* jeszcze wiecej instrukcji *)
    if (biez < MINVAL || biez > MAXVAL)
      continue;
    (* i kolejne pozyteczne instrukcje *)
  }
  return 0;
}
```

- Warunek zakończenia pętli podajemy po słowie kluczowym `while` – umieszczenie go wewnątrz instrukcji złożonej (do tego w kilku miejscach?) ze słowem `break` utrudnia analizę programu
- Dopuszczalne użycie słowa `continue` w miejsce instrukcji warunkowej `if`

O instrukcji `goto` (by xkcd)

Operatory dzielenia całkowitoliczbowego

Dla dowolnych całkowitych x i y zachodzi

$$x = y * (x \ / \ y) + (x \ \% \ y)$$

Operatory dzielenia całkowitoliczbowego

Dla dowolnych całkowitych x i y zachodzi

$$x = y * (x \ / \ y) + (x \ \% \ y)$$

Przykładowo procentowy wynik wyborów:

```
(IloscZa * 100) / IloscGlosow
```

Operatory dzielenia całkowitoliczbowego

Dla dowolnych całkowitych x i y zachodzi

$$x = y * (x \ / \ y) + (x \ \% \ y)$$

Przykładowo procentowy wynik wyborów:

```
(IloscZa * 100) / IloscGlosow
```

ale nie

```
IloscZa / IloscGlosow * 100
```

Operatory dzielenia całkowitoliczbowego cd.

k-ta cyfra w rozwinięciu liczby n określonej w układzie pozycyjnym o podstawie p

$$n_k = \left(n \ / \ p^{(k-1)} \right) \ \% \ p$$

Operatory dzielenia całkowitoliczbowego cd.

k-ta cyfra w rozwinięciu liczby n określonej w układzie pozycyjnym o podstawie p

$$n_k = \left(n \ / \ p^{(k-1)}\right) \ \% \ p$$

w szczególności, w układzie dziesiętnym

$$n_k = \left(n \ / \ 10^{(k-1)}\right) \ \% \ 10$$

Liczymy więc przykładowo

$$\begin{array}{rcrcrcrcl} (& n & / & 10^{k-1}) & \% & 10 & & & \\ (853 & / & 1) & \% & 10 & = & 3 \end{array}$$

Liczymy więc przykładowo

$$
\begin{array}{rcrcrcl}
(& n & / & 10^{k-1}) & \% & 10 & \\
(853 & / & 1) & \% & 10 & = & 3 \\
(853 & / & 10) & \% & 10 & = & 5
\end{array}
$$

Liczymy więc przykładowo

$$
\begin{array}{rcrcrcl}
(& n & / & 10^{k-1}) & \% & 10 & \\
(853 & / & 1) & \% & 10 & = & \boxed{3} \\
(853 & / & 10) & \% & 10 & = & \boxed{5} \\
(853 & / & 100) & \% & 10 & = & \boxed{8}
\end{array}
$$

Liczymy więc przykładowo

$$
\begin{array}{rcrcrcl}
(& n & / & 10^{k-1}) & \% & 10 & \\
(853 & / & 1) & \% & 10 & = & \boxed{3} \\
(853 & / & 10) & \% & 10 & = & \boxed{5} \\
(853 & / & 100) & \% & 10 & = & \boxed{8}
\end{array}
$$

i piszemy program

```
int Potega(int a, int b) {
  int m = 1;
  for(i=1; i<=b; i++)
    m = m * a;
  return m;
}
                                        (* Algorytm 1 *)
```

Liczymy więc przykładowo

$$
\begin{array}{rcrcrcl}
(& n & / & 10^{k-1}) & \% & 10 & \\
(853 & / & 1) & \% & 10 & = & \boxed{3} \\
(853 & / & 10) & \% & 10 & = & \boxed{5} \\
(853 & / & 100) & \% & 10 & = & \boxed{8}
\end{array}
$$

i piszemy program

```
int Potega(int a, int b) {
  int m = 1;
  for(i=1; i<=b; i++)
    m = m * a;
  return m;
}
int main()                                    (* Algorytm 1 *)
  ...
  NrCyfry = 1;
  while (???)
  {
    Cyfra = Liczba / Potega(10,NrCyfry-1) % 10;
    NrCyfry += 1;
  }
```

cyfr	podstawień	mnożeń	dodawań	porównań
3	30	12	6	?

cyfr	podstawień	mnożeń	dodawań	porównań
3	30	12	6	?
6	69	33	12	?

cyfr	podstawień	mnożeń	dodawań	porównań
3	30	12	6	?
6	69	33	12	?
n	n^2+5n+3	$\frac{n^2}{2}+\frac{5n}{2}$	$2n$	**?**

Napisaliśmy:

```
int Potega(int a, int b) {
  int m = 1;

  for(i=1; i<=b; i++)
    m = m * a;
  return m;
}
  ...                        (* Algorytm 1 *)
  NrCyfry = 1;
  while (???)
  {
    Cyfra = Liczba /
              Potega(10,NrCyfry-1) % 10;
    NrCyfry += 1;
```

Napisaliśmy:

```
int Potega(int a, int b) {
  int m = 1;

  for(i=1; i<=b; i++)
    m = m * a;
  return m;
}
  ...                          (* Algorytm 1 *)
  NrCyfry = 1;
  while (???)
  {
    Cyfra = Liczba /
              Potega(10,NrCyfry-1) % 10;
    NrCyfry += 1;
  }
```

Można prościej:

```
...                            (* Algorytm 2 *)
Dzielnik = 1;
while (Dzielnik < Liczba)
{
Cyfra = Liczba /
            Dzielnik % 10;
Dzielnik *= 10;
}
```

Napisaliśmy:

```
int Potega(int a, int b) {
  int m = 1;

  for(i=1; i<=b; i++)
    m = m * a;
  return m;
}
  ...                          (* Algorytm 1 *)
  NrCyfry = 1;
  while (???)
  {
    Cyfra = Liczba /
              Potega(10,NrCyfry-1) % 10;
    NrCyfry += 1;
  }
```

Można prościej:

```
...                            (* Algorytm 2 *)
Dzielnik = 1;
while (Dzielnik < Liczba)
{
Cyfra = Liczba /
            Dzielnik % 10;
Dzielnik *= 10;
}
```

Napisaliśmy:

```
int Potega(int a, int b) {
  int m = 1;

  for(i=1; i<=b; i++)
    m = m * a;
  return m;
}
  ...                          (* Algorytm 1 *)
  NrCyfry = 1;
  while (???)
  {
    Cyfra = Liczba /
                Potega(10,NrCyfry-1) % 10;
    NrCyfry += 1;
  }
```

Można prościej:

```
...                            (* Algorytm 2 *)
Dzielnik = 1;
while (Dzielnik < Liczba)
{
Cyfra = Liczba /
              Dzielnik % 10;
Dzielnik *= 10;
}
```

Napisaliśmy:

```
int Potega(int a, int b) {
  int m = 1;

  for(i=1; i<=b; i++)
    m = m * a;
  return m;
}
  ...                          (* Algorytm 1 *)
  NrCyfry = 1;
  while (???)
  {
    Cyfra = Liczba /
              Potega(10,NrCyfry-1) % 10;
    NrCyfry += 1;
  }
```

Można prościej:

```
...                            (* Algorytm 2 *)
Dzielnik = 1;
while (Dzielnik < Liczba)
{
Cyfra = Liczba /
            Dzielnik % 10;
Dzielnik *= 10;
}
```

Obecnie:

cyfr	podstawień	mnożeń	dodawań	porównań
3	7	9	0	4

Obecnie:

cyfr	podstawień	mnożeń	dodawań	porównań
3	7	9	0	4
6	13	18	0	7

Obecnie:

cyfr	podstawień	mnożeń	dodawań	porównań
3	7	9	0	4
6	13	18	0	7
n	$2n+1$	$3n$	0	$n+1$

Obecnie:

cyfr	podstawień	mnożeń	dodawań	porównań
3	7	9	0	4
6	13	18	0	7
n	$2n+1$	$3n$	0	$n+1$

A mieliśmy:

cyfr	podstawień	mnożeń	dodawań	porównań
3	30	12	6	?
6	69	33	12	?
n	n^2+5n+3	$\frac{n^2}{2}+\frac{5n}{2}$	$2n$	?

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1		

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1	3	

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1	3	85

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1	3	85
2		

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1	3	85
2	5	

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1	3	85
2	5	8

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1	3	85
2	5	8
3		

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1	3	85
2	5	8
3	8	

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1	3	85
2	5	8
3	8	0

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1	3	85
2	5	8
3	8	0
4		

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1	3	85
2	5	8
3	8	0
4	STOP	

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1	3	85
2	5	8
3	8	0
4	STOP	

Teraz:

cyfr	podstawień	mnożeń	dodawań	porównań
3	6	6	0	4

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                    (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1	3	85
2	5	8
3	8	0
4	STOP	

Teraz:

cyfr	podstawień	mnożeń	dodawań	porównań
3	6	6	0	4
6	12	12	0	7

Lub jeszcze prościej:

```
Liczba = 853
while (Liczba > 0)
{                       (* Algorytm 3 *)
  Cyfra = Liczba % 10;
  Liczba = Liczba / 10;
}
```

co daje

krok	Cyfra	Liczba
0	?	853
1	3	85
2	5	8
3	8	0
4	STOP	

Teraz:

cyfr	podstawień	mnożeń	dodawań	porównań
3	6	6	0	4
6	12	12	0	7
n	$2n$	$2n$	0	$n+1$

Złożoność obliczeniowa — zarys

Podsumowując:

algorytm	podstawień	mnożeń	dodawań	porównań	złożoność
1	$n^2 + 5n + 3$	$\frac{n^2}{2} + \frac{5n}{2}$	$2n$	?	$O(n^2)$

Złożoność obliczeniowa — zarys

Podsumowując:

algorytm	podstawień	mnożeń	dodawań	porównań	złożoność
1	n^2+5n+3	$\frac{n^2}{2}+\frac{5n}{2}$	$2n$	?	$O(n^2)$
2	$2n+1$	$3n$	0	$n+1$	$O(n)$

Złożoność obliczeniowa — zarys

Podsumowując:

algorytm	podstawień	mnożeń	dodawań	porównań	złożoność
1	n^2+5n+3	$\frac{n^2}{2}+\frac{5n}{2}$	$2n$	?	$O(n^2)$
2	$2n+1$	$3n$	0	$n+1$	$O(n)$
3	$2n$	$2n$	0	$n+1$	$O(n)$

Złożoność obliczeniowa — zarys

Podsumowując:

algorytm	podstawień	mnożeń	dodawań	porównań	złożoność
1	n^2+5n+3	$\frac{n^2}{2}+\frac{5n}{2}$	$2n$	?	O(n^2)
2	$2n+1$	$3n$	0	$n+1$	O(n)
3	$2n$	$2n$	0	$n+1$	O(n)

dla $n=100$

algorytm	podstawień	mnożeń	dodawań	porównań
1	10503	5250	200	?

Złożoność obliczeniowa — zarys

Podsumowując:

algorytm	podstawień	mnożeń	dodawań	porównań	złożoność
1	n^2+5n+3	$\frac{n^2}{2}+\frac{5n}{2}$	$2n$	?	O(n^2)
2	$2n+1$	$3n$	0	$n+1$	O(n)
3	$2n$	$2n$	0	$n+1$	O(n)

dla $n=100$

algorytm	podstawień	mnożeń	dodawań	porównań
1	10503	5250	200	?
2	201	300	0	101

Złożoność obliczeniowa — zarys

Podsumowując:

algorytm	podstawień	mnożeń	dodawań	porównań	złożoność
1	n^2+5n+3	$\frac{n^2}{2}+\frac{5n}{2}$	$2n$	?	O(n^2)
2	$2n+1$	$3n$	0	$n+1$	O(n)
3	$2n$	$2n$	0	$n+1$	O(n)

dla $n=100$

algorytm	podstawień	mnożeń	dodawań	porównań
1	10503	5250	200	?
2	201	300	0	101
3	200	200	0	101

Standardowe strumienie danych

W systemie, dla procesów automatycznie udostępniany jest

- standardowy strumień wejściowy `stdin`

Standardowe strumienie danych

W systemie, dla procesów automatycznie udostępniany jest

- standardowy strumień wejściowy `stdin`
- standardowy strumień wyjściowy `stdout`

Standardowe strumienie danych

W systemie, dla procesów automatycznie udostępniany jest

- standardowy strumień wejściowy `stdin`
- standardowy strumień wyjściowy `stdout`
- standardowy strumień diagnostyczny `stderr`

Standardowe strumienie danych

W systemie, dla procesów automatycznie udostępniany jest

- standardowy strumień wejściowy `stdin` (domyślnie zazwyczaj klawiatura terminala)
- standardowy strumień wyjściowy `stdout` (domyślnie zazwyczaj ekran terminala)
- standardowy strumień diagnostyczny `stderr` (domyślnie zazwyczaj ekran terminala)

Standardowe strumienie danych

W systemie, dla procesów automatycznie udostępniany jest

- standardowy strumień wejściowy `stdin` (domyślnie zazwyczaj klawiatura terminala)
- standardowy strumień wyjściowy `stdout` (domyślnie zazwyczaj ekran terminala)
- standardowy strumień diagnostyczny `stderr` (domyślnie zazwyczaj ekran terminala)

W dobrych środowiskach istnieje możliwość przekierowywania strumieni

```
./a.out <plikwe >plikwy
```

Standardowe strumienie danych

W systemie, dla procesów automatycznie udostępniany jest

- standardowy strumień wejściowy `stdin` (domyślnie zazwyczaj klawiatura terminala)
- standardowy strumień wyjściowy `stdout` (domyślnie zazwyczaj ekran terminala)
- standardowy strumień diagnostyczny `stderr` (domyślnie zazwyczaj ekran terminala)

W dobrych środowiskach istnieje możliwość przekierowywania strumieni

```
./a.out <plikwe >plikwy
```

i przetwarzania potokowego

```
prog1 | prog2 | prog3
```

Standardowa biblioteka we/wy `stdio`

Dołączenie pliku nagłówkowego biblioteki

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


Standardowa biblioteka we/wy `stdio`

Dołączenie pliku nagłówkowego biblioteki

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

Najprostsze mechanizmy wejścia/wyjścia

```
int getchar(void); /* zwraca przeczytany znak lub EOF */
int putchar(int);  /* zwraca wypisany znak lub EOF */
```

Standardowa biblioteka we/wy `stdio`

Dołączenie pliku nagłówkowego biblioteki

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

Najprostsze mechanizmy wejścia/wyjścia

```
int getchar(void); /* zwraca przeczytany znak lub EOF */
int putchar(int);  /* zwraca wypisany znak lub EOF */
```

```
#include <stdio.h>
#include <ctype.h>
main() { /* zamien wielkie litery na male */
  int znak;         /* co bedzie, gdy zadeklarujemy 'char znak'? */

  while ((znak = getchar()) != EOF)
    putchar(tolower(znak));
  return 0;
}
```

Formatowane wyjście – `printf`

Wypisywanie formatowanego wyjścia umożliwia funkcja

```
int printf(char *format, arg1, arg2 ...)
```

Formatowane wyjście – `printf`

Wypisywanie formatowanego wyjścia umożliwia funkcja

```
int printf(char *format, arg1, arg2 ...)
```

- funkcja pod nadzorem argumentu `format` przekształca i wypisuje swoje argumenty
- wartością funkcji jest liczba pomyślnie wypisanych znaków

Formatowane wyjście – `printf`

Wypisywanie formatowanego wyjścia umożliwia funkcja

```
int printf(char *format, arg1, arg2 ...)
```

- funkcja pod nadzorem argumentu `format` przekształca i wypisuje swoje argumenty
- wartością funkcji jest liczba pomyślnie wypisanych znaków

Format zawiera:

- zwykłe znaki, które są kopiowane do strumienia wyjściowego
- specyfikacje przekształceń (rozpoczynające się znakiem %), z których każda wskazuje sposób przekształcenia i wypisania kolejnego argumentu

Formatowane wyjście – przykłady

```
printf("Oto jestem swiecie\n");
```


Formatowane wyjście – przykłady

```
printf("Oto jestem swiecie\n");
printf("Znak uzyskany to %c\n",znak);  /* przy char znak; */
```

Formatowane wyjście – przykłady

```
printf("Oto jestem swiecie\n");
printf("Znak uzyskany to %c\n",znak);  /* przy char znak; */
printf("Znak uzyskany to %u\n",znak); /* a takze %d */
```

Formatowane wyjście – przykłady

```
printf("Oto jestem swiecie\n");
printf("Znak uzyskany to %c\n",znak);  /* przy char znak; */
printf("Znak uzyskany to %u\n",znak); /* a takze %d */
printf("Wartosci to %d i %d\n",i1,i2); /*przy int i1, i2;*/
```

Formatowane wyjście – przykłady

```
printf("Oto jestem swiecie\n");
printf("Znak uzyskany to %c\n",znak);  /* przy char znak; */
printf("Znak uzyskany to %u\n",znak); /* a takze %d */
printf("Wartosci to %d i %d\n",i1,i2); /*przy int i1, i2;*/
printf("Wartosci to %3d i %6d\n",i1,i2);
```

Formatowane wyjście – przykłady

```
printf("Oto jestem swiecie\n");
printf("Znak uzyskany to %c\n",znak);  /* przy char znak; */
printf("Znak uzyskany to %u\n",znak); /* a takze %d */
printf("Wartosci to %d i %d\n",i1,i2); /*przy int i1, i2;*/
printf("Wartosci to %3d i %6d\n",i1,i2);
printf("Wartosci to %3.0f i %6.1f\n",f1,f2);
```

Formatowane wyjście – przykłady cd

Dostępne specyfikacje przekształceń:

`%d` `%i` liczba dziesiętna

Formatowane wyjście – przykłady cd

Dostępne specyfikacje przekształceń:

`%d %i`	liczba dziesiętna
`%6d`	liczba dziesiętna, zajmująca co najmniej 6 znaków

Formatowane wyjście – przykłady cd

Dostępne specyfikacje przekształceń:

`%d %i`	liczba dziesiętna
`%6d`	liczba dziesiętna, zajmująca co najmniej 6 znaków
`%f`	liczba zmiennopozycyjna
`%6f`	liczba zmiennopozycyjna, zajmująca co najmniej 6 znaków
`%.2f`	liczba zmiennopozycyjna z 2 miejscami po kropce dziesiętnej
`%6.2f`	liczba zmiennopozycyjna z 2 miejscami po kropce dziesiętnej, zajmująca co najmniej 6 znaków

Formatowane wyjście – przykłady cd

Dostępne specyfikacje przekształceń:

`%d %i`	liczba dziesiętna
`%6d`	liczba dziesiętna, zajmująca co najmniej 6 znaków
`%f`	liczba zmiennopozycyjna
`%6f`	liczba zmiennopozycyjna, zajmująca co najmniej 6 znaków
`%.2f`	liczba zmiennopozycyjna z 2 miejscami po kropce dziesiętnej
`%6.2f`	liczba zmiennopozycyjna z 2 miejscami po kropce dziesiętnej, zajmująca co najmniej 6 znaków
`%u`	liczba dziesiętna bez znaku
`%o`	liczba ósemkowa bez znaku
`%x`	liczba szesnastkowa bez znaku
`%ld`	liczba całkowita typu long
`%c`	jeden znak
`%s`	ciąg znaków wypisany do napotkania `\0` lub wyczerpania znaków
`%%`	wypisanie znaku %

Formatowane wejście – `scanf`

Wczytywanie formatowanego wejścia umożliwia funkcja

```
int scanf(char *format, ...);
```


Formatowane wejście – `scanf`

Wczytywanie formatowanego wejścia umożliwia funkcja

```
int scanf(char *format, ...);
```

- funkcja pod nadzorem argumentu `format` wczytuje swoje pozostałe argumenty, które muszą być wskaźnikami wskazującymi, gdzie należy przekazać przekształcone dane wejściowe
- wartością funkcji jest liczba pomyślnie wczytanych argumentów lub `EOF`

Formatowane wejście – `scanf`

Wczytywanie formatowanego wejścia umożliwia funkcja

```
int scanf(char *format, ...);
```

- funkcja pod nadzorem argumentu `format` wczytuje swoje pozostałe argumenty, które muszą być wskaźnikami wskazującymi, gdzie należy przekazać przekształcone dane wejściowe
- wartością funkcji jest liczba pomyślnie wczytanych argumentów lub `EOF`

```
scanf("%f%f%f", &a, &b, &c);
scanf("Dane:%f%f%f", &a, &b, &c);
```

Składnia funkcji

Ogólna definicja funkcji ma postać

```
[ typ-zwracanej-wartosci ] nazwa-funkcji
                          "(" [ deklaracja-parametrow ] ")"
"{"
   { deklaracja }
   { instrukcja }
"}"
```

Składnia funkcji

Ogólna definicja funkcji ma postać

```
[ typ-zwracanej-wartosci ] nazwa-funkcji
                          "(" [ deklaracja-parametrow ] ")"
"{"
   { deklaracja }
   { instrukcja }
"}"
```

```
int Potega(int podstawa, int wykladnik)  /* przyklad funkcji */
{                       /* podnies podstawa do potegi wykladnik */
  int i, tmp = 1;
                                                  /* wersja 1 */
  for(i=1; i<=wykladnik; i++)
    tmp *= podstawa;
  return tmp;
}
```

Funkcje – przykłady

```
int Potega(int podstawa, int wykladnik){
  int i, tmp = 1;
                                          /* wersja 1 */
  for(i=1; i<=wykladnik; i++)
    tmp *= podstawa;
  return tmp;
}
```

- Jeśli pominięto `typ-zwracanej-wartosci`, to przyjmuje się, że funkcja zwraca wartość typu `int`,

Funkcje – przykłady

```
int Potega(int podstawa, int wykladnik){
  int i, tmp = 1;
                                        /* wersja 1 */
  for(i=1; i<=wykladnik; i++)
    tmp *= podstawa;
  return tmp;
}
```

- Jeśli pominięto `typ-zwracanej-wartosci`, to przyjmuje się, że funkcja zwraca wartość typu `int`,
- instrukcja `return` jest narzędziem, dzięki któremu wywołana funkcja przekazuje do miejsca wywołania wartość wyrażenia (`return wyrazenie;`),

Funkcje – przykłady

```
int Potega(int podstawa, int wykladnik){
  int i, tmp = 1;
                                        /* wersja 1 */
  for(i=1; i<=wykladnik; i++)
    tmp *= podstawa;
  return tmp;
}
```

- Jeśli pominięto `typ-zwracanej-wartosci`, to przyjmuje się, że funkcja zwraca wartość typu `int`,
- instrukcja `return` jest narzędziem, dzięki któremu wywołana funkcja przekazuje do miejsca wywołania wartość wyrażenia (`return wyrazenie;`),
- deklaracja

  `int Potega(int podstawa, int wykladnik);`

  zwana jest prototypem funkcji i musi być zgodna z definicją funkcji,

Funkcje – przykłady

```
int Potega(int podstawa, int wykladnik){
  int i, tmp = 1;
                                        /* wersja 1 */
  for(i=1; i<=wykladnik; i++)
    tmp *= podstawa;
  return tmp;
}
```

- Jeśli pominięto `typ-zwracanej-wartosci`, to przyjmuje się, że funkcja zwraca wartość typu `int`,
- instrukcja `return` jest narzędziem, dzięki któremu wywołana funkcja przekazuje do miejsca wywołania wartość wyrażenia (`return wyrazenie;`),
- deklaracja
  `int Potega(int podstawa, int wykladnik);`
  zwana jest prototypem funkcji i musi być zgodna z definicją funkcji,
- nazwy argumentów funkcji jak i zadeklarowanych w niej zmiennych są dla niej całkowicie lokalne i są niedostępne dla wszystkich innych funkcji.

Funkcje – przekazywanie argumentów

W języku C wszystkie argumenty funkcji są przekazywane przez wartość (Wywoływana funkcja zamiast oryginałów otrzymuje wartości swoich argumentów w zmiennych tymczasowych)

Funkcje – przekazywanie argumentów

W języku C wszystkie argumenty funkcji są przekazywane przez wartość (Wywoływana funkcja zamiast oryginałów otrzymuje wartości swoich argumentów w zmiennych tymczasowych)

```
int Potega(int podstawa, int wykladnik){
  int i;
                                          /* wersja 2 */
  for(i=1; wykladnik>0; --wykladnik)
    i *= podstawa;
  return i;
}
```

Funkcje – przekazywanie argumentów

W języku C wszystkie argumenty funkcji są przekazywane przez wartość (Wywoływana funkcja zamiast oryginałów otrzymuje wartości swoich argumentów w zmiennych tymczasowych)

```
int Potega(int podstawa, int wykladnik){
  int i;
                                          /* wersja 2 */
  for(i=1; wykladnik>0; --wykladnik)
    i *= podstawa;
  return i;
}
```

- Argumentem `wykladnik` posłużono się jako zmienną tymczasową, zmniejszaną stopniowo do zera,

Funkcje – przekazywanie argumentów

W języku C wszystkie argumenty funkcji są przekazywane przez wartość (Wywoływana funkcja zamiast oryginałów otrzymuje wartości swoich argumentów w zmiennych tymczasowych)

```
int Potega(int podstawa, int wykladnik){
  int i;
                                          /* wersja 2 */
  for(i=1; wykladnik>0; --wykladnik)
    i *= podstawa;
  return i;
}
```

- Argumentem `wykladnik` posłużono się jako zmienną tymczasową, zmniejszaną stopniowo do zera,
- cokolwiek zrobiono ze zmienną `wykladnik` wewnątrz funkcji `Potega`, nie ma to żadnego wpływu na wartość argumentu, z którym funkcja ta została wywołana.

Struktura programu w C

Program w języku C ma następującą strukturę

```
[ definicja-zmiennych-globalnych ]
{ definicja-funkcji }
```

Struktura programu w C

Program w języku C ma następującą strukturę

```
[ definicja-zmiennych-globalnych ]
{ definicja-funkcji }
```

- Nie można deklarować funkcji wewnątrz innych funkcji,

Struktura programu w C

Program w języku C ma następującą strukturę

```
[ definicja-zmiennych-globalnych ]
{ definicja-funkcji }
```

- Nie można deklarować funkcji wewnątrz innych funkcji,
- jedna z zadeklarowanych funkcji musi nazywać się `main`,

Struktura programu w C

Program w języku C ma następującą strukturę

```
[ definicja-zmiennych-globalnych ]
{ definicja-funkcji }
```

- Nie można deklarować funkcji wewnątrz innych funkcji,
- jedna z zadeklarowanych funkcji musi nazywać się `main`,
- odwołania mogą być dokonywane tylko do wcześniej określonych (przez definicję lub prototyp) obiektów,

Struktura programu w C

Program w języku C ma następującą strukturę

```
[ definicja-zmiennych-globalnych ]
{ definicja-funkcji }
```

- Nie można deklarować funkcji wewnątrz innych funkcji,
- jedna z zadeklarowanych funkcji musi nazywać się `main`,
- odwołania mogą być dokonywane tylko do wcześniej określonych (przez definicję lub prototyp) obiektów,
- zmienne globalne są dostępne w dowolnej funkcji programu,

Struktura programu w C

Program w języku C ma następującą strukturę

```
[ definicja-zmiennych-globalnych ]
{ definicja-funkcji }
```

- Nie można deklarować funkcji wewnątrz innych funkcji,
- jedna z zadeklarowanych funkcji musi nazywać się `main`,
- odwołania mogą być dokonywane tylko do wcześniej określonych (przez definicję lub prototyp) obiektów,
- zmienne globalne są dostępne w dowolnej funkcji programu,
- zmienne lokalne definiuje się wewnątrz dowolnej instrukcji złożonej, nie tylko tej definiującej funkcję,

Struktura programu w C

Program w języku C ma następującą strukturę

```
[ definicja-zmiennych-globalnych ]
{ definicja-funkcji }
```

- Nie można deklarować funkcji wewnątrz innych funkcji,
- jedna z zadeklarowanych funkcji musi nazywać się `main`,
- odwołania mogą być dokonywane tylko do wcześniej określonych (przez definicję lub prototyp) obiektów,
- zmienne globalne są dostępne w dowolnej funkcji programu,
- zmienne lokalne definiuje się wewnątrz dowolnej instrukcji złożonej, nie tylko tej definiującej funkcję,
- zmienna lokalna zaczyna istnieć w chwili wywołania instrukcji złożonej i znika po jej zakończeniu,

Struktura programu w C

Program w języku C ma następującą strukturę

```
[ definicja-zmiennych-globalnych ]
{ definicja-funkcji }
```

- Nie można deklarować funkcji wewnątrz innych funkcji,
- jedna z zadeklarowanych funkcji musi nazywać się `main`,
- odwołania mogą być dokonywane tylko do wcześniej określonych (przez definicję lub prototyp) obiektów,
- zmienne globalne są dostępne w dowolnej funkcji programu,
- zmienne lokalne definiuje się wewnątrz dowolnej instrukcji złożonej, nie tylko tej definiującej funkcję,
- zmienna lokalna zaczyna istnieć w chwili wywołania instrukcji złożonej i znika po jej zakończeniu,
- tak zadeklarowane zmienne zasłaniają identycznie nazwane zmienne z bloków zewnętrznych.

Struktura programu w C – przykłady

```
#include <stdio.h>
int Potega(int, int);
/* testowanie funkcji Potega */
main() {

  int i;

  for (i=0, i<=10, i++)
    printf("%d %d %d\n", i, Potega(2,i), Potega(-3,i));
  return 0;
}
/* Podnies podstawa do potegi wykladnik */
int Potega(int podstawa, int wykladnik){

  int i;                                          /* wersja 3 */
                                                /* niezalecana */
  for(i=1; wykladnik>0; --wykladnik, i*=podstawa);
  return i;
}
```

Struktura programu w C – przykłady

```
#include <stdio.h>
int Potega(int, int);
/* testowanie funkcji Potega */
main() {

  int i;

  for (i=0, i<=10, i++)
    printf("%d %d %d\n", i, Potega(2,i), Potega(-3,i));
  return 0;
}
/* Podnies podstawa do potegi wykladnik */
int Potega(int podstawa, int wykladnik){

  int i;                                          /* wersja 3 */
                                                /* niezalecana */
  for(i=1; wykladnik>0; --wykladnik, i*=podstawa);
  return i;
}
```

Struktura programu w C – przykłady cd

```
#include <stdio.h>
int Max(int, int);                    /* prototyp funkcji */
int wiek_taty, wiek_mamy;             /* zmienne globalne */
int main(){                          /* definicja funkcji */
  int tmp;                            /* zmienna lokalna */
  printf("Podaj wiek taty i mamy\n");
  scanf("%d %d", &wiek_taty, &wiek_mamy);
  printf("Starsza osoba ma %d lat\n", Max(wiek_taty,wiek_mamy));
  return 0;
}
/* Zwroc wieksza z dwoch podanych wartosci */
int Max(int a, int b){               /* definicja funkcji */
  return a>b?a:b;
}
```

Struktura programu w C – przykłady cd

```
#include <stdio.h>
int Max(int, int);                    /* prototyp funkcji */
int wiek_taty, wiek_mamy;             /* zmienne globalne */
int main(){                          /* definicja funkcji */
  int tmp;                            /* zmienna lokalna */
  printf("Podaj wiek taty i mamy\n");
  scanf("%d %d", &wiek_taty, &wiek_mamy);
  printf("Starsza osoba ma %d lat\n", Max(wiek_taty,wiek_mamy));
  return 0;
}
/* Zwroc wieksza z dwoch podanych wartosci */
int Max(int a, int b){               /* definicja funkcji */
  return a>b?a:b;
}
```

Struktura programu w C – przykłady cd

```
#include <stdio.h>
int Max(int, int);                    /* prototyp funkcji */
int wiek_taty, wiek_mamy;             /* zmienne globalne */
int main(){                          /* definicja funkcji */
  int tmp;                            /* zmienna lokalna */
  printf("Podaj wiek taty i mamy\n");
  scanf("%d %d", &wiek_taty, &wiek_mamy);
  printf("Starsza osoba ma %d lat\n", Max(wiek_taty,wiek_mamy));
  return 0;
}
/* Zwroc wieksza z dwoch podanych wartosci */
int Max(int a, int b){               /* definicja funkcji */
  return a>b?a:b;
}
```

- Pojawiające się w definicji funkcji Max argumenty `a`, `b` to parametry formalne funkcji,

Struktura programu w C – przykłady cd

```
#include <stdio.h>
int Max(int, int);                    /* prototyp funkcji */
int wiek_taty, wiek_mamy;             /* zmienne globalne */
int main(){                          /* definicja funkcji */
  int tmp;                            /* zmienna lokalna */
  printf("Podaj wiek taty i mamy\n");
  scanf("%d %d", &wiek_taty, &wiek_mamy);
  printf("Starsza osoba ma %d lat\n", Max(wiek_taty,wiek_mamy));
  return 0;
}
/* Zwroc wieksza z dwoch podanych wartosci */
int Max(int a, int b){               /* definicja funkcji */
  return a>b?a:b;
}
```

- Pojawiające się w definicji funkcji Max argumenty `a`, `b` to parametry formalne funkcji,
- pojawiające się w jej wywołaniu argumenty `wiek_taty`, `wiek_mamy` to parametry aktualne funkcji.

Zmienne globalne a lokalne, przesłanianie zmiennych

- Zasadniczo, funkcje powinny przekazywać sobie wartości przez argumenty

Zmienne globalne a lokalne, przesłanianie zmiennych

- Zasadniczo, funkcje powinny przekazywać sobie wartości przez argumenty
- Zmienne globalne są ogólnie dostępne – są alternatywą dla argumentów

Zmienne globalne a lokalne, przesłanianie zmiennych

- Zasadniczo, funkcje powinny przekazywać sobie wartości przez argumenty
- Zmienne globalne są ogólnie dostępne – są alternatywą dla argumentów
- Dla funkcji wymagających dostępu do dużej liczby danych zmienne globalne mogą okazać się wygodniejsze i bardziej skuteczne

Zmienne globalne a lokalne, przesłanianie zmiennych

- Zasadniczo, funkcje powinny przekazywać sobie wartości przez argumenty
- Zmienne globalne są ogólnie dostępne – są alternatywą dla argumentów
- Dla funkcji wymagających dostępu do dużej liczby danych zmienne globalne mogą okazać się wygodniejsze i bardziej skuteczne – choć może wtedy trzeba najpierw pomyśleć nad uporządkowaniem struktur danych

Zmienne globalne a lokalne, przesłanianie zmiennych

- Zasadniczo, funkcje powinny przekazywać sobie wartości przez argumenty
- Zmienne globalne są ogólnie dostępne – są alternatywą dla argumentów
- Dla funkcji wymagających dostępu do dużej liczby danych zmienne globalne mogą okazać się wygodniejsze i bardziej skuteczne – choć może wtedy trzeba najpierw pomyśleć nad uporządkowaniem struktur danych
- Zmienne globalne mogą niekorzystnie wpływać na strukturę programu – nie nadużywać

Zmienne globalne a lokalne, przesłanianie zmiennych

- Zasadniczo, funkcje powinny przekazywać sobie wartości przez argumenty
- Zmienne globalne są ogólnie dostępne – są alternatywą dla argumentów
- Dla funkcji wymagających dostępu do dużej liczby danych zmienne globalne mogą okazać się wygodniejsze i bardziej skuteczne – choć może wtedy trzeba najpierw pomyśleć nad uporządkowaniem struktur danych
- Zmienne globalne mogą niekorzystnie wpływać na strukturę programu – nie nadużywać
- Zmienne lokalne są dla funkcji wewnętrzne – zaczynają istnieć w chwili wywołania funkcji i nikną zaraz po jej zakończeniu nie zachowują więc swoich wartości – dopóki ich wartości początkowe nie zostaną określone, należy przyjąć, że zawierają nieznane wartości

Zmienne globalne a lokalne, przesłanianie zmiennych

- Zasadniczo, funkcje powinny przekazywać sobie wartości przez argumenty
- Zmienne globalne są ogólnie dostępne – są alternatywą dla argumentów
- Dla funkcji wymagających dostępu do dużej liczby danych zmienne globalne mogą okazać się wygodniejsze i bardziej skuteczne – choć może wtedy trzeba najpierw pomyśleć nad uporządkowaniem struktur danych
- Zmienne globalne mogą niekorzystnie wpływać na strukturę programu – nie nadużywać
- Zmienne lokalne są dla funkcji wewnętrzne – zaczynają istnieć w chwili wywołania funkcji i nikną zaraz po jej zakończeniu nie zachowują więc swoich wartości – dopóki ich wartości początkowe nie zostaną określone, należy przyjąć, że zawierają nieznane wartości
- Jeśli wewnątrz bloku zdefiniowano zmienną o takiej samej nazwie jak nazwa zmiennej ”zewnętrznej”, to zmienna ”wewnętrzna” przesłania zmienną ”zewnętrzną” – do zmiennej ”zewnętrznej” nie ma dostępu

Zmienne globalne a lokalne, przesłanianie zmiennych

- Zasadniczo, funkcje powinny przekazywać sobie wartości przez argumenty
- Zmienne globalne są ogólnie dostępne – są alternatywą dla argumentów
- Dla funkcji wymagających dostępu do dużej liczby danych zmienne globalne mogą okazać się wygodniejsze i bardziej skuteczne – choć może wtedy trzeba najpierw pomyśleć nad uporządkowaniem struktur danych
- Zmienne globalne mogą niekorzystnie wpływać na strukturę programu – nie nadużywać
- Zmienne lokalne są dla funkcji wewnętrzne – zaczynają istnieć w chwili wywołania funkcji i nikną zaraz po jej zakończeniu nie zachowują więc swoich wartości – dopóki ich wartości początkowe nie zostaną określone, należy przyjąć, że zawierają nieznane wartości
- Jeśli wewnątrz bloku zdefiniowano zmienną o takiej samej nazwie jak nazwa zmiennej ”zewnętrznej”, to zmienna ”wewnętrzna” przesłania zmienną ”zewnętrzną” – do zmiennej ”zewnętrznej” nie ma dostępu
- Chodzi o to, by się nie pomylić!

Funkcje — podsumowanie

Dekompozycja problemu: podział problemu na wiele mniejszych i wydzielenie rozwiązań tych mniejszych problemów od rozwiązania problemu głównego jest drogą do zredukowania wielkości i stopnia komplikacji problemu.

Funkcje — podsumowanie

Dekompozycja problemu: podział problemu na wiele mniejszych i wydzielenie rozwiązań tych mniejszych problemów od rozwiązania problemu głównego jest drogą do zredukowania wielkości i stopnia komplikacji problemu.

Czytelność i przejrzystość programu: niezależnie od stopnia złożoności problemu, podział na funkcje jest drogą do zwiększenia czytelności programu, co zawsze jest celowe. Jednakże dodawanie procedur nie jest celem samym w sobie.

Funkcje — podsumowanie

Dekompozycja problemu: podział problemu na wiele mniejszych i wydzielenie rozwiązań tych mniejszych problemów od rozwiązania problemu głównego jest drogą do zredukowania wielkości i stopnia komplikacji problemu.

Czytelność i przejrzystość programu: niezależnie od stopnia złożoności problemu, podział na funkcje jest drogą do zwiększenia czytelności programu, co zawsze jest celowe. Jednakże dodawanie procedur nie jest celem samym w sobie.

Unikanie powtórzeń: często warto wyodrębnić w funkcję powtarzający się zestaw instrukcji, wyrażeń, bądź schemat obliczeniowy.

Funkcje — podsumowanie

Dekompozycja problemu: podział problemu na wiele mniejszych i wydzielenie rozwiązań tych mniejszych problemów od rozwiązania problemu głównego jest drogą do zredukowania wielkości i stopnia komplikacji problemu.

Czytelność i przejrzystość programu: niezależnie od stopnia złożoności problemu, podział na funkcje jest drogą do zwiększenia czytelności programu, co zawsze jest celowe. Jednakże dodawanie procedur nie jest celem samym w sobie.

Unikanie powtórzeń: często warto wyodrębnić w funkcję powtarzający się zestaw instrukcji, wyrażeń, bądź schemat obliczeniowy.

Zasada lokalności: określa, że wszystkie elementy mające ze sobą związek powinny znaleźć się jak najbliżej siebie w programie.

Funkcje — podsumowanie

Dekompozycja problemu: podział problemu na wiele mniejszych i wydzielenie rozwiązań tych mniejszych problemów od rozwiązania problemu głównego jest drogą do zredukowania wielkości i stopnia komplikacji problemu.

Czytelność i przejrzystość programu: niezależnie od stopnia złożoności problemu, podział na funkcje jest drogą do zwiększenia czytelności programu, co zawsze jest celowe. Jednakże dodawanie procedur nie jest celem samym w sobie.

Unikanie powtórzeń: często warto wyodrębnić w funkcję powtarzający się zestaw instrukcji, wyrażeń, bądź schemat obliczeniowy.

Zasada lokalności: określa, że wszystkie elementy mające ze sobą związek powinny znaleźć się jak najbliżej siebie w programie.

Wielkość funkcji: niezbyt duże, np. < 50 linii, ważna jest też spójność.

Podsumowanie

- Zagadnienia podstawowe

1. Jakie znasz instrukcje warunkowe w C i jaka jest ich składnia?
2. Jakie znasz instrukcje pętli w C? Czym się one od siebie różnią?
3. Czy pętla `for(i = 0; i <= 10; i++)` wykona się 10 razy?
4. Porównaj efekt wykonania instrukcji pętli `for(i=0;i<10;i++){...}` z instrukcją `for(i=0;i<10;){i++;...}`.
5. Jaka biblioteka języka C zawiera deklaracje funkcji obsługi standardowego wejście?
6. W jaki sposób ustala się sposób formatowania wyjścia dla poznanych typów zmiennych?
7. Czy podane wyrażenie ma prawidłową składnię:
   `3>5||4<3&&2+2?printf("poz\n");:2<3?printf("poz2\n");:printf("neg\n");`?
8. Czy funkcja `void inicjuj(int a){a=5;}` ma sens? Dlaczego?
9. Czym różnią się parametry formalne funkcji od jej parametrów aktualnych?
10. Czym różnią się zmienne lokalne od globalnych?
11. Na czym polega zjawisko przesłaniania zmiennych?
12. Czy wskazane jest używanie zmiennych globalnych w miejsce zmiennych lokalnych?
13. Czy prawdą jest, że zmienne globalne należy stosować w celu zmniejszenia ilości danych, które trzeba przekazać przy wywołaniu funkcji?

- Zagadnienia rozszerzające

1. Czy znając złożoność obliczeniową dwóch programów możemy stwierdzić który z nich wykona się szybciej? Uzasadnij i podaj przykłady.
2. Czy w warunku instrukcji `if` może wystąpić wywołanie funkcji `scanf`? Jeżeli tak, to w jakich sytuacjach może to być wskazane?
3. Poszukaj, jakie jeszcze są możliwości formatowania tekstu za pomocą funkcji `printf` i `scanf`.
4. Jakimi liczbami oznaczone są standardowe wyjścia w systemie UNIX/linux?
5. Czy istnieje inny sposób przekazywania zmiennej do funkcji niż przez wartość? Jeśli tak, to w jakim celu?
6. Niektóre języki programowania oprócz przekazywania argumentów do funkcji przez wartość umożliwiają przekazywanie ich przez zmienną/referancję. Jakie są różnice między tymi sposobami?

• Zadania

1. Zapisz funkcję `int Min(int a, int b)` wyznaczającą wartość minimalną.
2. Napisz program wyszukujący wszystkie dzielniki zadanej liczby.
3. Napisz algorytm który wczyta liczbę w systemie dziesiętnym i wypisze ją w systemie siódemkowym (lub ogólniej dowolnym zadanym). Napisz na jego podstawie program.
4. Napisz algorytm pozwalający na wyliczenie numeru grupy do haszówki na podstawie numeru indeksu i współczynników przy trzech ostatnich cyfrach. Napisz na jego podstawie program.

5. Napisz program wypisujący na ekranie kolejne potęgi liczby podanej przez użytkownika aż do przekroczenia wartości miliona wykorzystując formatowanie tekstu w celu uzyskania czytelnych rezultatów.
6. Znajdź algorytm, pozwalający na znajdowanie liczb pierwszych. Na jego podstawie napisz program, który będzie znajdował wszystkie liczby pierwsze w podanym przedziale.
7. Za pomocą zagnieżdżonych pętli `for` wypisz wszystkie dni występujące w roku 2010 w formacie `dd:mm`.

