Podstawy Programowania

Wykład IV

Specyfikacja funkcji, operacje wejścia i wyjścia na plikach, rekurencja, tablice i wskaźniki

Robert Muszyński
Katedra Cybernetyki i Robotyki, PWr

Zagadnienia: specyfikacja funkcji, operacje wejścia i wyjścia na plikach, formatowane wejście i wyjście, struktury sterujące, rekurencja, niebezpieczeństwa rekurencji, wskaźniki, tablice, funkcje operujące na tablicach, argumenty funkcji a wskaźniki, arytmetyka wskaźników, rzutowanie.

Copyright © 2007–2018 Robert Muszyński

Niniejszy dokument zawiera materiały do wykładu na temat podstaw programowania w językach wysokiego poziomu. Jest on udostępniony pod warunkiem wykorzystania wyłącznie do własnych, prywatnych potrzeb i może być kopiowany wyłącznie w całości, razem ze stroną tytułową.

– Skład FoilTEX –

Specyfikacja funkcji

Specyfikacja powinna zawierać komplet informacji, niezbędnych do napisania poprawnego wywołania funkcji. A wśród nich:

(a) co dana funkcja robi, w tym również określenie wartości, którą wylicza i zwraca,

(b) znaczenie wszystkich jej argumentów,

(c) warunki, które muszą być spełnione, aby funkcja działała poprawnie, np. przedział dopuszczalnych wartości argumentów, oraz

(d) opis trwałych zmian dokonywanych przez funkcję, takich jak: zmiana wartości zmiennych globalnych(!), stanu plików zewnętrznych, informacje wyświetlane na standardowe wyjście, itp.

© R. Muszyński, 29 października 2018

Zapis specyfikacji — warunki PRE i POST

Specyfikacja często przybiera postać odpowiednich komentarzy.

PRE — określają wymagania, które muszą być spełnione, aby funkcja działała poprawnie.

POST — opisują zmiany, które nastąpiły w wyniku zadziałania funkcji.

Przykładowo:

- PRE: program wymaga podania wartości w przedziale [0, 21]
- POST: program zwraca liczbę szczęśliwych dni do końca roku

© R. Muszyński, 29 października 2018

Warunki PRE i POST

- PRE: wartość parametru N musi być zawarta w przedziale [0,100]
- PRE: parametry A, B, C muszą spełniać warunek istnienia rozwiązań rzeczywistych równania kwadratowego: B B - 4 A C $>=$ 0 oraz A ! $=$ 0
- PRE: parametr Oper musi mieć jedną z wartości: ”+”, ”-”, ”*”, ”/”
- PRE: stałe globalne MinWart i MaxWart muszą spełniać zależność MinWart $<=$ MaxWart
- PRE: zmienna plikowa f1 musi być związana z przygotowanym do zapisu plikiem zewnętrznym (wykonane fopen w trybie ”append”)
- POST: parametry X1, X2 przyjmują wartości pierwiastków równania kwadratowego zadanego parametrami A, B, C
- POST: wartością funkcji jest wpisana przez użytkownika z klawiatury liczba z przedziału [0,N] lub liczba -1, jeśli użytkownik wpisał na klawiaturze liczbę spoza tego przedziału
- POST: na pliku zewnętrznym związanym ze zmienną plikową PLIK została zapisana linia zawierająca wartości parametrów X1, X2, X3

© R. Muszyński, 29 października 2018

Operacje wejścia i wyjścia na plikach

- Do plików odwołujemy się poprzez zmienną wskaźnikową typu `FILE`

  ```
  FILE *plik;
  ```

- każdy wykorzystywany plik musi zostać otwarty w odpowiednim trybie funkcją `fopen`

  ```
  FILE *fopen(char *nazwa, char *tryb);
  ```

  która jest zdefiniwana w nagłówku `stdio`,
- w programie przykładowe wywołanie funkcji `fopen` może mieć postać

  ```
  plik = fopen("kubus.txt", "r");
  ```

  możliwe tryby: `"r"` – czytanie, `"w"` – pisanie, `"a"` – dopisywanie
- w przypadku błędu funkcja `fopen` zwraca wartość `NULL`,
- do wczytania znaku z pliku służy funkcja `int getc(FILE *plik)`, która zwraca kolejny znak ze strumienia wskazywanego przez `plik` lub `EOF`,
- by zapisać `znak` do pliku wskazywanego przez `plik` należy użyć funkcji `int putc(int znak, FILE *plik)`, która zwraca wartość tego znaku lub `EOF` jako sygnał wystąpienia błędu,

© R. Muszyński, 29 października 2018

Operacje wejścia i wyjścia na plikach cd.

- jako `plik` można użyć standardowych `stdin`, `stdout` i `stderr`,
- by zamknąć otwarty strumień danych należy się posłużyć funkcją `fclose`

  ```
  int fclose(FILE *plik);
  ```

  która zwraca 0 w przypadku pomyślnego zamknięcia strumienia lub `EOF` w przeciwnym razie,
- większość systemów operacyjnych nakłada ograniczenie na liczbę jednocześnie otwartych plików w jednym programie,
- wszystkie operacje wejścia i wyjścia są buforowane,
- użycie funkcji `fclose` opróżnia bufor,
- funkcja `fclose` jest wywoływana automatycznie dla wszystkich jeszcze otwartych plików, gdy program kończy się normalnie.

© R. Muszyński, 29 października 2018

Operacje wejścia i wyjścia na plikach – przykład

```
#include<stdio.h>
int main(){                    /* kopiowanie pliku na standardowe wyjscie */
  FILE *plik;
  void kopiuj(FILE *, FILE *);

  if ((plik=fopen("kubus.txt", "r")) == NULL){
    printf("kopiuj: nie moge otworzyc kubusia");
    return 1;
  } else {
    kopiuj(plik,stdout);
    fclose(plik);
  }
  return 0;
}

/* kopiuj: kopiuje zawartosc pliku wej do pliku wyj */
void kopiuj(FILE *wej, FILE *wyj){
  int znak

  while ((znak = getc(wej)) != EOF)
    putc(znak, wyj);
}
```


© R. Muszyński, 29 października 2018

Formatowane wejścia i wyjścia na plikach

Przy formatowanym czytaniu z i pisaniu do plików możemy korzystać z funkcji `fscanf` i `fprintf`

```
int fscanf(FILE *plik, char *format, ...)
int fprintf(FILE *plik, char *format, ...)
```

- Wywołanie `printf(...)` jest równoznaczne z `fprintf(stdout,...)`
- Wywołanie `scanf(...)` jest równoznaczne z `fscanf(stdin,...)`
- By wysłać komunikat na `stderr` należy posłużyć się funkcją `fprintf`

```
fprintf(stderr, "%s: blad pisania do stdout\n", program);
fprintf(stderr, "%s: nie moge otworzyc %s\n", program, nazwa);
```


© R. Muszyński, 29 października 2018

Iteracje a rekurencja

Struktury sterujące:

- bezpośrednie następstwo,
- wybór warunkowy,
- iteracja ograniczona,
- iteracja nieograniczona (warunkowa).

Inna metoda tworzenia powtórzeń — **rekurencja**.

Rodzaje rekurencji (rekursji):

- Rekurencja bezpośrednia — zdolność funkcji do wywołania samej siebie.
- Rekurencja pośrednia — wzajemne wywoływanie się dwóch lub większej liczby funkcji.

© R. Muszyński, 29 października 2018

Rekurencja a iteracje

```
int silnia(int n;)
/* Funkcja wylicza rekurencyjnie */
/* wartosc silni.                */
/* UWAGA: dla n <= 0 zwraca 1    */
{
  if (n <= 1)
    return 1;
  else
    return n * silnia(n-1);
}
```

```
int silnia(int n;)
/* Funkcja wylicza iteracyjnie */
/* wartosc silni.              */
/* UWAGA: dla n <= 0 zwraca 1  */
{
  int wynik = 1;

  while (n > 1)
  {
    wynik *= n;
    n--;
  }
  return wynik;
}
```

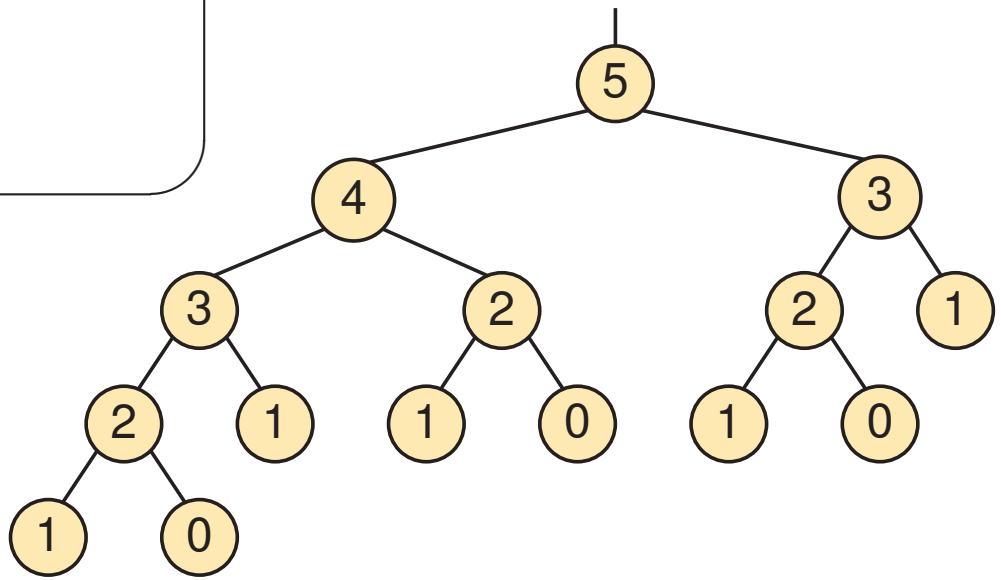

© R. Muszyński, 29 października 2018

Niebezpieczeństwa rekurencji

- złożoność obliczeniowa

```
int fib(int n;)
/* Funkcja wylicza rekurencyjnie liczby  */
/* Fibonacciego.  UWAGA: n musi byc >= 0 */
{
  if (n == 0) return 0; else
  if (n == 1) return 1; else
  return fib(n-1) + fib(n-2);
}
```

Diagram wywołań dla `n=5`:



- problem stopu

© R. Muszyński, 29 października 2018

Wskaźniki

- Pamięć zorganizowana jest zazwyczaj w ciąg kolejno numerowanych komórek
- Adres obiektu (zmiennej) podaje jednoargumentowy operator adresu (referencji) `&`

  ```
  p = &c
  ```

  Teraz zmienna `p` „wskazuje na” zmienną `c`. Zmienna `p` jest wskaźnikiem.
- By odwołać się do wartości zmiennej wskazywanej przez wskaźnik `p` należy użyć operatora odwołania pośredniego (wyłuskania, dereferencji), oznaczanego symbolem `*`

  ```
  zm = *p             /* rownowazne zm = c */
  *p = cos
  if (*p == c)...     /* zawsze prawdziwe */
  ```
- By zadeklarować zmienną wskaźnikową również należy się posłużyć operatorem `*`

  ```
  int x = 1, y = 2;
  int *ip;            /* ip jest wskaznikiem do obiektow typu int */
  ip = &x;            /* teraz ip wskazuje na x */
  y = *ip;            /* y ma teraz wartosc 1 */
  *ip = 0;            /* x ma teraz wartosc 0 */
  ip = &y;            /* teraz ip wskazuje na y */
  *ip = 0;            /* y ma teraz wartosc 0 */
  ```


© R. Muszyński, 29 października 2018

Wskaźniki cd

- Gdy wskaźnik `ip` wskazuje na zmienną całkowitą `x`, to `*ip` może wystąpić wszędzie tam, gdzie może wystąpić `x`

```
*ip = *ip +1;
```

- Jednoargumentowe operatory `&` i `*` wiążą silniej niż operatory arytmetyczne, tak więc, by zwiększyć zmienną wskazywaną przez `ip` można także napisać

```
*ip += 1;
++*ip;
(*ip)++              /* *ip++ zwiekszy wartosc wskaznika */
```

- Wskaźniki są zwykłymi zmiennymi

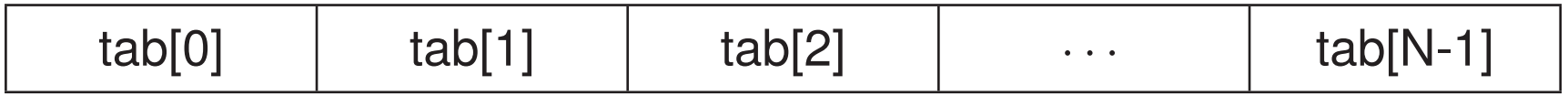
```
int x, *ip, *iq;
ip = &x;             /* teraz ip wskazuje na x */
iq = ip;             /* teraz takze iq wskazuje na x */
```

- Wiemy więc już, dlaczego w funkcji `scanf` używaliśmy konstrukcji z operatorem adresu `&`, postaci `&zmienna`

© R. Muszyński, 29 października 2018

Tablice

- Tablica jednowymiarowa N elementowa



tab[i] $\Leftrightarrow$ prosta zmienna

- Deklaracja tablicy ma postać

```
int oceny[10];  /* dziesiecioelementowa tablica liczb int */
```

- By odwołać się do elementu tablicy można napisać

```
oceny[1] = 5;   /* element 1 tablicy oceny ma wartosc 5 */
i = 5;
oceny[i] = 3;   /* element 5 tablicy oceny ma wartosc 3 */
oceny[i++] = 1; /* element 5 tablicy oceny ma wartosc 1 */
oceny[i] = 4;   /* element 6 tablicy oceny ma wartosc 4 */
oceny[++i] = 0; /* element 7 tablicy oceny ma wartosc 0 */
```


© R. Muszyński, 29 października 2018

```
#define NSTUDENTOW 200      /* calkowita liczba studentow */
int main() {
   int Grupa[NSTUDENTOW]:
   int stud, min, max, srednia, suma;
/*****************************************************/
  suma = 0;
  min = 10;
  max = 0;
  for (stud = 0; stud < NSTUDENTOW; stud++)
  {
    printf("Prosze podac zaliczenie dla studenta %d",stud);
    scanf("%d",&(Grupa[stud]));  /* mozna to zrobic ladniej */
    suma += Grupa[stud];
    if (Grupa[stud] > max) max = Grupa[stud];
    if (Grupa[stud] < min) min = Grupa[stud];
  }
  srednia = suma / NStudentow;
```

Tablice jednowymiarowe a wskaźniki

- Nazwa tablicy reprezentuje położenie jej początkowego elementu – jest wskaźnikiem na ten element: `nazwa_tablicy` $\Leftrightarrow$ `&nazwa_tablicy[0]`

```
*oceny = 2;      /* element 0 tablicy oceny ma wartosc 2 */
*(oceny+2) = 3; /* element 2 tablicy oceny ma wartosc 3 */
```

- elementy tablicy mogą być wskazywane przez zmienne wskaźnikowe odpowiedniego typu

```
int *ptr;        /* ptr jest wskaznikiem do obiektow typu int */

ptr = &oceny[0]; /* ptr wskazuje na 0. element tablicy oceny */
ptr = oceny;      /* jak powyzej tylko prosciej */
x = *ptr;      /* zawartosc oceny[0] zostaje skopiowana do x */
x = *(ptr+1); /* zawartosc oceny[1] zostaje skopiowana do x */
x = *(++ptr); /* jw, tyle ze teraz ptr pokazuje na oceny[1] */
x = *ptr++;   /* x ma wart. oceny[1], ptr pokazuje na oceny[2]*/
```


© R. Muszyński, 29 października 2018

Proste operacje na tablicach

Przy wykonywaniu operacji na tablicach należy pamiętać, że:

- pierwszy element tablicy w języku C ma indeks 0,
- nazwa tablicy reprezentuje położenie jej początkowego elementu,
- w języku C nie jest sprawdzana poprawność (zakres) indeksów!

Zainicjowanie elementów tablicy

```
#define ROZMIAR 100   /* Rozmiar tablicy danych */
int tablica[ROZMIAR], i;
    ⋮
  for(i = 0; i < ROZMIAR; i++)
    tablica[i] = 0;
```

Wypisanie wszystkich elementów tablicy

```
  for(i = 0; i < ROZMIAR; i++)
    printf("Tablica[%2d] = %5d\n", i, tablica[i]);
```


© R. Muszyński, 29 października 2018

Funkcje operujące na tablicach

```
#define ROZMIAR 10
void WczytajTablice(double tablica[]){
  int i;
  printf("Podaj wartosci elementow tablicy \n");
  for(i = 0; i < ROZMIAR; i++){
    printf("Tab[%2d] = ", i+1);
    scanf("%f", &tablica[i]);
  }
}
void WyswietlTablice(double tablica[]){ ... }
void DodajTablice(double wej_1[], double wej_2[], double wynik[]){
  int i;
  for(i = 0; i < ROZMIAR; i++)
    wynik[i] = wej_1[i] + wej_2[i];
}
int main(void){
  double A[ROZMIAR], B[ROZMIAR], C[ROZMIAR];

  WczytajTablice(A);
  WyswietlTablice(A);
  WczytajTablice(B);
  DodajTablice(A, B, C);
  WyswietlTablice(C);
}
```


© R. Muszyński, 29 października 2018

Funkcje operujące na tablicach cd.

```
void KopiujNapis(char wej[], char wyj[])
{                                   /* wersja z indeksowaniem tablic */
  int i=0;

  while ((wyj[i] = wej[i]) != '\0')
    i++;
}
```

```
void KopiujNapis(char *wej, char *wyj)    /* naglowki rownowazne! */
{                                         /* wersja wskaznikowa 1 */
  while ((*wyj = *wej) != '\0'){
    wej++;
    wyj++;
  }
}
```

```
void KopiujNapis(char *wej, char *wyj)
{                                         /* wersja wskaznikowa 2 */
  while ((*wyj++ = *wej++) != '\0');
}
```


© R. Muszyński, 29 października 2018

Tablice a wskaźniki – przydział pamięci

Widzieliśmy, że w deklaracji nagłówka funkcji napisy

```
char wej[]
char *wej
```

są równoważne. Jednakże, gdy zadeklarujemy

```
char tab[] = "To jest string.";
char *ptr  = "Jak rowniez to.";
```

- `tab` jest tablicą, której zawartość jest zainicjalizowana określonymi znakami, której nie można zmienić jako zmiennej, ale której wszystkie pozycje znakowe mogą być dowolnie zmieniane.

- `ptr` jest zmienną wskaźnikową zainicjalizowaną wskaźnikiem na napis znakowy. Wartość tej zmiennej wskaźnikowej można zmieniać dowolnie, lecz zawartości pozycji znakowych nie (napis jest tablicą stałą, przydzieloną w pamięci stałych).

© R. Muszyński, 29 października 2018

Tablice a wskaźniki – przydział pamięci cd.

Tak więc, przy rzeczonych deklaracjach

```
char tab[] = "To jest string.";
char *ptr  = "Jak rowniez to.";
```

mamy co następuje:

```
tab[1] = ptr[1];     /* poprawne kopiowanie znakow */
*(tab+1) = *(ptr+1); /* rowniez poprawne */
ptr[1] = tab[1];     /* kopiowanie znakow NIEDOZWOLONE */
*(ptr+1) = *(tab+1); /* rowniez NIEDOZWOLONE */
tab = ptr;           /* to przypisanie jest NIEDOZWOLONE */
ptr = tab;           /* poprawne, choc gubi pamiec */
```


© R. Muszyński, 29 października 2018

Wskaźniki a argumenty funkcji

```
void zamien(int x, int y)        /* !!!!!!!!! Z L E !!!!!!!!!! */
{                                /* zamiana wartosci argumentow */
  int tmp;

  tmp = x;
  x = y;
  y = tmp;
}
```

```
void zamien(int *px, int *py)    /* !!!!!! D O B R Z E !!!!!!! */
{                                /* zamiana wartosci argumentow */
  int tmp;

  tmp = *px;
  *px = *py;
  *py = tmp;
}
```

Wywołanie: `zamien(&a, &b);` przy `int a, b;`

© R. Muszyński, 29 października 2018

Arytmetyka wskaźników

- Wskaźniki stanowią dane typu wskaźnikowego, który jest podobny do typu liczb całkowitych. Pod wskaźnik można podstawić wartość, wartość wskaźnika można wyświetlić funkcją printf.

```
char c, *cp;
int  i, *ip;
ip = 0;   /* inicjalizacja wartoscia 0 */
ip = &i;  /* inicjalizacja poprawna wartoscia wskaznikowa */
printf("ip = %d\n", ip);
```

- Wartość wskaźników można powiększać lub zmniejszać, powodując, że wskaźnik wskazuje na następne/poprzednie elementy względem wskazywanego przed zmianą wartości

```
cp = &c;
cp += 1; /* cp wskazuje do nastepnego elementu po c */
ip += 1; /* ip wskazuje do nastepnego elementu po i */
```

  Wartość liczbowa wskaźnika `cp` zwiększyła się o 1, natomiast wskaźnik `ip` zwiększył się być może o 4 (dokładniej: o liczbę bajtów przypadającą na zmienną typu `int`, która może być różna na różnych systemach (`sizeof`)).

Arytmetyka wskaźników cd.

Do poprawnych operacji wskaźnikowych należą:

- przypisanie wskaźników do obiektów tego samego typu
- przypisanie wskaźnikowi wartości zero (`NULL`)
- przyrównanie wskaźnika do zera (`NULL`)
- dodawanie lub odejmowanie wskaźnika i liczby całkowitej
- odejmowanie bądź porównywanie dwóch wskaźników do elementów tej samej tablicy

Wszystkie inne operacje na wskaźnikach są nielegalne

- dodawanie do siebie wskaźników
- ich mnożenie, dzielenie, przesuwanie, składanie z maskami
- dodawanie do nich liczby typu `float` lub `double`
- nie wolno nawet (z wyjątkiem typu `void *`) wskaźnikowi do obiektów jednego typu przypisywać bez rzutowania wskaźnika do obiektów innego typu

© R. Muszyński, 29 października 2018

Rzutowanie

- Zadaniem rzutowania jest konwersja danej jednego typu na daną innego typu. Konwersja może być niejawna (domyślna konwersja przyjęta przez kompilator) lub jawna (podana explicite przez programistę).
- Przykłady konwersji niejawnej:

```
int i = 42.7;              /* konwersja z double do int */
float f = i;               /* konwersja z int do float */
double d = f;              /* konwersja z float do double */
unsigned u = i;            /* konwersja z int do unsigned int */
```

- Do jawnego wymuszenia konwersji służy jednoargumentowy operator rzutowania `(typ)`, np.:

```
double d = 3.14;
int pi = (int) d;          /* konwersja z double do int */
d = (double) pi;           /* konwersja z int do double */
```

  Nigdy nie należy stosować rzutowania by „uciszyć kompilator”.

© R. Muszyński, 29 października 2018

Rzutowanie cd.

- Rzutowanie może prowadzić do utraty informacji:

```
int i;
char c;

c = i;
i = c;
```

aczkolwiek nie zawsze:

```
int i;
char c;

i = c;
c = i;
```

- By „poeksperymentować” z rzutowaniem można posłużyć się konstrukcjami w rodzaju:

```
int i = 17;
float f = 22.5;

printf ("i bez rzutowania jako float %f\n", i);
printf ("i z rzutowaniem jako float %f\n", (float) i);
printf ("f bez rzutowania jako int %d\n", f);
printf ("f z rzutowaniem jako int %d\n", (int) f);
```


© R. Muszyński, 29 października 2018

Podsumowanie

- Zagadnienia podstawowe

1. Wskaż elementy, które powinna zawierać specyfikacja funkcji.
2. Czy wewnątrz warunków PRE i POST umieszcza się opis działania funkcji?
3. Czy w części PRE specyfikacji funkcji powinna być podana informacja o tym, jakiego typu funkcja zwraca wartość?
4. Czy warunek `POST: wszystkie parametry funkcji musza byc dodatnie` jest poprawny?
5. Jaka wartość zostanie zwrócona przez funkcję `fopen`, jeśli użytkownik nie ma prawa do odczytu wskazanego w jej argumencie pliku?
6. Jakie są konsekwencje nie zamknięcia funkcją `fclose` otwartego w programie strumienia danych?
7. W jaki sposób można sprawdzić wartość kolejnego znaku występującego w pliku tak, aby w dalszej części programu było możliwe jego ponowne odczytanie?
8. Wymień zalety iteracji i rekurencji.
9. Czy istnieje możliwość, aby dwie zmienne wskazywały na to samo miejsce w pamięci?
10. Czy zapis `int &x;` jest prawidłowy?
11. Czy dla `#define LICZBA 10` możliwe jest przypisanie `x = &LICZBA`?
12. Czy odwołania do i-tego elementu tablicy jednowymiarowej `t` postaci `t[i]` oraz `*(t+i)` są zawsze równoważne?

© R. Muszyński, 29 października 2018

13. Wskaż różnice pomiędzy przekazywaniem zmiennej jako argumentu do funkcji poprzez jej wartość a poprzez wskaźnik na nią. Podaj wady i zalety każdego z rozwiązań.

• Zagadnienia rozszerzające

1. W jaki sposób można podać ścieżkę do pliku w funkcji `fopen` (względną, bezwzględną)?
2. W jaki sposób sprawdzić, jaki błąd spowodowało niepowodzenie otwarcia pliku lub operacji wejściawyjścia?
3. Czy poprawna jest instrukcja `fclose(stdout);`?
4. Czy można dowolny algorytm iteracyjny zapisać w formie rekurencyjnej? A na odwrót?
5. Jakie trudności, poza określeniem warunku stopu, mogą się pojawić podczas wykorzystywania rekurencji?
6. Rozróżnia się rekurencje zwykłą i ogonową – czym one się od siebie różnią?
7. Czy istnieje możliwość zdefiniowania wskaźnika na `void`? Jeśli tak, to w jakim celu można to wykorzystać?
8. Jakie mogą być skutki rzutowania zmiennej o mniejszym rozmiarze na zmienną o rozmiarze większym? A większej na mniejszą?

• Zadania

1. Napisz program, który we wskazanym pliku policzy liczbę znaków, słow (jednostki oddzielone od siebie białymi znakami) oraz linii.

© R. Muszyński, 29 października 2018

2. Napisz program, który we wskazanym pliku znajdzie linie, które zawierają podany wzorzec.
3. Napisz funkcję `void UsunZnak(char *napis, char znak)` usuwającą wszystkie znaki `znak` z napisu `napis`.
4. Stwórz iteracyjną i rekurencyjną wersję programu wyliczającego n-ty element ciągu arytmetycznego.
5. Wczytaj do tablicy n liczb. Korzystając ze wskaźników znajdź największą z nich.
6. Napisz program umożliwiający wczytanie stopnia wielomianu i współczynników stojących przy zmiennej, a następnie wyliczający wartości wielomianu dla wprowadzanych wartości tej zmiennej. Uwzględnij możliwość zapisywania wyników do pliku.
7. Dwuwymiarową tablicę można wykorzystać m.in. do zapisania planszy do gry w ”wilka i owce” (takiej jak w warcabach). Napisz program, w którym użytkownicy (gracze) na przemian mogliby wykonywać ruchy pionków na takiej planszy, wpisując początkowe i końcowe numery wiersza i kolumny.

© R. Muszyński, 29 października 2018