Programowanie w C/C++

Informacje organizacyjne

1

Referencje

Informacje organizacyjne:

Krzysztof Trojanowski

Wykład:
środy, 13:15 – 14:45

Strona główna www z komunikatami:
http://troja.uksw.edu.pl/category/pro2020/

E-mail:
k.trojanowski@uksw.edu.pl
maile zbiorowe i anonimowe są ignorowane (konieczne jest imię i nazwisko)

2

Zasady zaliczeń

Zasady zaliczenia Programowania Obiektowego

1. Trzeba zaliczyć **laboratoria PO**.
2. Potem można próbować zaliczyć **wykład PO** (podejść do egzaminu).

Nigdy w odwrotnej kolejności.

3

Zasady zaliczeń

Zasady zaliczenia **laboratoriów PO**

1. trzeba zaliczyć **zajęcia laboratoryjne**
2. trzeba zaliczyć **zadanie semestralne**
(*tematy zadań semestralnych rozdawane są na trzecich lub czwartych zajęciach*)

Również tylko w tej kolejności.

4

Zasady zaliczeń

Zasady zaliczenia **zajęć laboratoryjnych**

- Na każdych zajęciach można uzyskać 0, 6, 7, 8, 9 lub 10 pkt.
- Zajęcia zaczynają się od wejściówki. Jej zaliczenie nie daje punktów, ale uprawnia do przystąpienia do rozwiązywania zadań.
- Na zajęciach zawsze ustalony jest minimalny zestaw zadań do realizacji. Wykonanie tego zestawu w całości gwarantuje 6 pkt.
- Niewykonanie tego zestawu - 0 pkt.
- Nie ma praktyki przyznawania 1, 2 lub 3 pkt. za wykonanie np. połowy minimalnego zestawu zadań.
- Za program, który się nie kompiluje, nie ma żadnych punktów. Nigdy.

5

Zasady zaliczeń

Zasady zaliczenia **zajęć laboratoryjnych**

- Oprócz zadań zestawu minimalnego podawane są na zajęciach jeszcze inne zadania, za które można zdobyć pozostałe 4 punkty.
- Można mieć tylko trzy razy 0 pkt. za udział w zajęciach. Jeżeli przydarzy się to cztery razy, jedne zajęcia można odrobić na koniec semestru. Odrobienie polega na samodzielnym wykonaniu na ostatnich zajęciach dodatkowego zadania, którego zakres dotyczy całego materiału, niezależnie od tego, które zajęcia zostały zaliczone na zero i są odrabiane.
- Pięć razy 0 pkt. oznacza negatywną ocenę z zajęć laboratoryjnych.

6

Zasady zaliczeń

Zasady zaliczenia zadania semestralnego

- Aby zaliczyć zadanie semestralne należy na *przedostatnich* zajęciach w semestrze oddać do oceny prowadzącemu:
  - Kod źródłowy programu opatrzony odpowiednimi komentarzami
  - Dokumentację, obejmującą: manual, diagram klas, oraz specyfikację danych wejściowych i wyjściowych
  - Przykładowe dane wejściowe
- Oddanie powyższego zestawu w późniejszym terminie powoduje zmniejszenie maksymalnej liczby punktów do uzyskania za zadanie semestralne do 80%.
- Brak któregokolwiek z komponentów powyższego zestawu dyskwalifikuje zadanie semestralne.

7

Zasady zaliczeń

Zasady zaliczenia zadania semestralnego

- W programie muszą być zastosowane techniki i technologie obiektowe, których lista zostanie podana przez prowadzącego przy okazji rozdawania tematów zadań semestralnych.
- Brak wykorzystania w/w technologii będzie powodował obniżenie oceny za zadanie semestralne, nawet jeżeli udało się zrealizować kompletną funkcjonalność programu ustaloną w założeniach projektowych.
- Za zadanie semestralne można uzyskać 100 pkt.

8

Zasady zaliczeń

Zasady zaliczenia **ćwiczeń PO**

Warunkiem koniecznym zaliczenia ćwiczeń jest spełnienie łącznie następujących trzech wymagań:

(1) zebranie co najmniej połowy ze wszystkich możliwych do uzyskania punktów (licząc razem za zajęcia i zadanie semestralne),

(2) ewentualnie niezaliczenie nie więcej niż trzech zajęć w semestrze oraz

(3) zaliczenie zadania semestralnego, tj. uzyskanie za niego co najmniej połowy z możliwych do uzyskania punktów.

Wszystko to należy spełnić w terminie do końca semestru.

9

Zasady zaliczeń - UWAGA

Ćwiczenia mogą być **zaliczone w terminie do końca semestru** (10 czerwca) albo **nie zaliczone wcale**.

W tym drugim przypadku jest jeszcze możliwość **poprawy** ćwiczeń we wrześniu ale tylko po uprzednim zaliczeniu zadania semestralnego.

Prowadzący **ma prawo wyznaczyć nowy temat** zadania semestralnego na poprawę we wrześniu.

10

Zasady zaliczeń

Zasady zaliczenia wykładów

- Wykład kończy się egzaminem pisemnym.
- Do egzaminu dopuszczone zostaną tylko te osoby, które wcześniej otrzymają zaliczenie z ćwiczeń. Należy je uzyskać przed rozpoczęciem sesji egzaminacyjnej (do 10 czerwca).
- Przewidywane są dwa terminy egzaminu: jeden w sesji czerwcowej i jeden w sesji wrześniowej.

11

Zasady zaliczeń

Zasady zaliczenia wykładów

- Zestaw egzaminacyjny będzie zawierał kilkanaście pytań z teorii programowania obiektowego oraz dobrych praktyk programistycznych, dla których odpowiedzi są kilkuzdaniowe.
- Na egzaminie nie będzie natomiast wymagane napisanie kodu programu.
- Może być jednak wymagane poprawne zinterpretowanie kodu podanego w materiale egzaminacyjnym (pytania w rodzaju: "dlaczego ten program nie zostanie skompilowany? - znajdź błąd", lub "co pojawi się w oknie konsoli po wykonaniu tego programu?").
- Liczba punktów za część „teoretyczną" i za zinterpretowanie kodu jest zwykle porównywalna.

12

Referencje:

Dowolny podręcznik do programowania obiektowego w C++:

- Język C++, Stroustrup Bjarne, WNT, 2002
- Podstawy języka C++, S.B. Lippman, J. Lajoie, WNT, 2003
- C++ dla każdego, Jesse Liberty, Helion, 2002
- Język C++. Szkoła programowania. Stephen Prata, Helion 2012
- Symfonia C++ Standard t.1/2, J. Grębosz, Edition 2000, 2008
- Thinking in C++. Edycja polska, Bruce Eckel, Helion, 2002
- Thinking in C++ Tom 2. Edycja polska, Bruce Eckel, Chuck Allison, Helion, 2004

...

Dodatkowo:

Algorytmy, Maciej Sysło, WSiP, 2002

13

Bardzo ważna informacja

Obecność na wykładach nie jest obowiązkowa, **ale ..**
jeżeli już ktoś postanowił uczestniczyć:

☹

14

ISO/ANSI C

kontynuacja

15

ISO/ANSI C

Materiał, który jest traktowany jako znany:

- Operacje WE/WY (`printf`, `scanf`)
- Zmienne
- Instrukcje:
  `if-else`, `switch`, `while`, `for`
- Stałe znakowe, kody ASCII znaków
- Wybrane funkcje matematyczne
- Tablice jedno- i dwuwymiarowe
- Łańcuchy tekstowe

16

Operatory

Logiczne, bitowe, arytmetyczne, wskazania, odniesienia

17

ISO/ANSI C

Operacje arytmetyczne:

- a++ – przyrostkowe zwiększenie
- a-- – przyrostkowe zmniejszenie
- --a – przedrostkowe zwiększenie
- ++a – przedrostkowe zmniejszenie
- a*b – mnożenie
- a/b – dzielenie
- a%b – reszta z dzielenia
- a+b – suma
- a-b – różnica

18

ISO/ANSI C

Operacje bitowe:

- a>>b – przesunecie bitowe w prawo: $a/2^b$
- a<<b – przesunecie bitowe w lewo: $a*2^b$
- a|b – alternatywa bitowa (OR)
- a^b – bitowa różnica symetryczna (XOR)
- a&b – koniunkcja bitowa (AND)
- ~a – negacja bitowa

The byte ordering used to represent integers by most Intel® CPUs is little-endian.

Visual Studio

19

ISO/ANSI C

Operacje logiczne:

- !a – negacja logiczna
- a<b – mniejsze od
- a<=b – mniejsze lub równe
- a>b – większe od
- a>=b – większe lub równe
- a==b – równe
- a!=b – różne
- a&&b – koniunkcja logiczna
- a||b – alternatywa logiczna

20

ISO/ANSI C

Złożone operatory przypisania:

- a += b; ⇔ a = a + b;
- a -= b; ⇔ a = a – b;
- a *= b; ⇔ a = a * b;
- a /= b; ⇔ a = a / b;
- a %= b; ⇔ a = a % b;
- a <<= b; ⇔ a = a << b;
- a >>= b; ⇔ a = a >> b;
- a &= b; ⇔ a = a & b;
- a |= b; ⇔ a = a | b;
- a ^= b; ⇔ a = a ^ b;

21

ISO/ANSI C

Operator selekcji:

```
b ? w1 : w2
```

1. obliczenie wartości b
2. jeżeli b==prawda to obliczenie w1 i ignorowanie w2
3. jeżeli b==fałsz to obliczenie w2 i ignorowanie w1

Przykład:

```
int a = 0, b=1, c;
c = a > b ? a : b;  /*wybiera wartość większą*/
```


22

ISO/ANSI C

Szczególne własności operatorów arytmetycznych:

kod:	*stan zmiennych:*
`int a=1;`	`a: 1`
`int b=a++;`	`a: 2 b: 1`
`int a=1;`	`a: 1`
`int b=++a;`	`a: 2 b: 2`

Argumentem musi być **l-nazwa**: (identyfikator zmiennej trwałej)

```
(x + y)++;          /* Źle! */
(x * 4)++;          /* Źle! */
((a>b) ? a : b )++; /* Źle! */
```


23

ISO/ANSI C

Szczególne własności operatorów logicznych:

operatory logiczne działają również na wartościach całkowitych

Kod programu:	*stan zmiennych:*
`int a=1, b=2;`	`a: 1 b: 2`
`int r = a && b;`	`a: 1 b: 2 r: 1`
`a=0;`	`a: 0 b: 2 r: 1`
`r = a && b;`	`a: 0 b: 2 r: 0`
`int a=0, b=0;`	`a: 0 b: 0`
`int r = a || b++;`	`a: 0 b: 1 r: 0`
`r = ++a && b;`	`a: 1 b: 1 r: 1`

24

ISO/ANSI C

Szczególne własności operatorów logicznych:

kolejność obliczania argumentów w wyrażeniu – od lewej

Kod programu: *stan zmiennych:*

```
int a=0, b=0;                 a: 0  b: 0
int r = ++a || b++;           a: 1  b: 0 (!)   r: 1

int a=0, b=0;                 a: 0  b: 0
int r = a++ || ++b;           a: 1  b: 1  r: 1

int a=-2;
int r = ++a && ++a && ++a && ++a && ++a;
```

Pytanie: jaką wartość ma 'a'?

ISO/ANSI C

Szczególne własności operatorów arytmetycznych:

W wyrażeniach arytmetycznych porządek wartościowania nie jest ustalony

Przykład:

```
int a=1,b=2,c=3,d;
d = (--c<a) + (--c<b); /* ryzykowny zapis */
```

obliczając od lewej: (2<1)+(1<2), razem: 1

obliczając od prawej: (1<1) + (2<2), razem: 0

ISO/ANSI C - wskaźniki

Deklaracja zmiennej wskaźnikowej: `Typ * nazwa;`

Nie ma znaczenia, gdzie jest spacja:

```
int* wsk;
int * wsk;
int *wsk;
```

- Za każdym razem jest to prawidłowo zadeklarowana zmienna, służąca do przechowywania adresu innej zmiennej, przechowującej wartość całkowitą typu `int`
- Zadeklarowana zmienna wskaźnikowa jest typu: `int*`
- Deklaracja kilku wskaźników tego samego typu:

```
int *wsk1, *wsk2, *wsk3;
```


ISO/ANSI C - wskaźniki

Przykłady:

```
int* wsk ;              /* wskaźnik na zmienną całkowitą */

double* wsk_liczby ;    /* wskaźnik na zmienną rzeczywistą */

int* tab_wsk[ 10 ] ;    /* tablica dziesięciu wskaźników na int */

int (*wsk_t)[ 10 ] ;    /* wskaźnik na tablicę 10-ciu liczb int */

int* (*wsk_t)[ 10 ] ;   /* wskaźnik na tablicę 10-ciu wskaźników na int */

int** wsk_w ;           /* wskaźnik na zmienną zawierającą wskaźnik
                           na zmienną typu int (tzn. wskaźnik na wskaźnik) */
```


ISO/ANSI C - wskaźniki

Aby otrzymać adres zmiennej, który potem będzie można zapisać do zmiennej wskaźnikowej, należy np. użyć operatora pobrania adresu: &

Przykład:

```
int x = 9;              /* tworzymy zmienną typu int */
int *wskx = NULL;       /* tworzymy zmienną wskaźnikową typu int*
                           i od razu inicjalizujemy ją wartością NULL,
                           która oznacza adres pusty */
wskx = &x;              /* pobieramy adres zmiennej 'x' i zapisujemy
                           go do zmiennej 'wskx' */
```


ISO/ANSI C - wskaźniki

Po co incjalizować zmienną wskaźnikową adresem pustym NULL?

Z powodów praktycznych.

W przypadku błędnie zaimplementowanego algorytmu może się zdarzyć, że program będzie próbował używać zmiennej wskaźnikowej, która nie ma przypisanej żadnej prawidłowej wartości.

No to co?..

Wtedy procesor nie rozpozna, że jest to wartość przypadkowa i będzie ją interpretował jako prawidłowy adres. W przypadku próby pisania do obszaru pamięci wskazywanego przez taki przypadkowy adres może nastąpić przerwanie pracy programu. 😖

ISO/ANSI C - wskaźniki

Korzystanie ze zmiennych wskaźnikowych:

Przykład:

```
int m, k = 7;
int *pk = NULL;
pk = &k;
*pk += 1;
m = *pk;
```


31

ISO/ANSI C - wskaźniki

Wskaźniki a tablice:

Weźmy tablicę: `int tab[20];`

`tab[3]` --- czwarta komórka tablicy, stąd:
`x = tab[3];` --- instrukcja kopiowania czwartego elementu do zmiennej `x`
`tab` --- ...?
`y = tab;` --- ...?

Wartością '`tab`' jest adres pierwszego elementu tablicy (!)
Stąd: `*tab` reprezentuje pierwszy element tablicy
[pobranie danych za pośrednictwem wskaźnika = *wyłuskanie* (ang. *dereferencing*)]

Jakiego typu musi być zmienna `y`?

32

ISO/ANSI C - wskaźniki

Wskaźniki a tablice:

Do wskaźników można dodawać (i odejmować) liczby całkowite. Dodawanie jest zdefiniowane w szczególny sposób.

Niech: `int *p = tab;`

Wtedy '`p`' przechowuje adres pierwszego elementu tablicy '`tab`', natomiast wartością wyrażenia '`p+x`' będzie adres '`tab`' powiększony o '`x`' wielokrotności wymiaru zmiennej typu '`int`'
(bo takiego typu wartości akurat przechowuje ta tablica).

Czyli: '`p+x`' przechowuje adres elementu tablicy o indeksie '`x`' (!)

33

ISO/ANSI C - wskaźniki

Arytmetyka wskaźników:

```
int tab[10] = {1,2,3,4,5};
int i, *p, *q;
q = tab;
p = q + 2;          /* p wskazuje na: 3 */
i = *(tab + 2);     /* i ma wartość: 3 */
i = q[5];           /* i ma wartość: 0 */
p = &tab[0] + 3;    /* p wskazuje na: 4 */
p -= 2;             /* p wskazuje na: 2 */
```


34

ISO/ANSI C - wskaźniki

Arytmetyka wskaźników:

```
&tab[0]     ↔   tab
p[i]        ↔   *( p + i )
i[p]        ↔   *( p + i )      (!)
```

`p - q`

różnica między dwoma wartościami adresowymi – ma sens, jeżeli obydwa wskazują na elementy tej samej tablicy. Wynikiem jest „odległość" między komórkami tablicy mierzona w jednostkach równych szerokości pojedynczej komórki.

35

ISO/ANSI C - wskaźniki

Arytmetyka dla tablic dwuwymiarowych:

`int tab[3][5];`

`tab[0][0]`	`tab[0][1]`	`tab[0][2]`	`tab[0][3]`	`tab[0][4]`
`tab[1][0]`	`tab[1][1]`	`tab[1][2]`	`tab[1][3]`	`tab[1][4]`
`tab[2][0]`	`tab[2][1]`	`tab[2][2]`	`tab[2][3]`	`tab[2][4]`

Alokacja w pamięci:

tab[0][0]	tab[0][1]	tab[0][2]	tab[0][3]	tab[0][4]	tab[1][0]	tab[1][1]	tab[1][2]	tab[1][3]	tab[1][4]	tab[2][0]	tab[2][1]	tab[2][2]	tab[2][3]	tab[2][4]

36

ISO/ANSI C - wskaźniki

Arytmetyka dla tablic dwuwymiarowych:

```
int tab[3][5];
tab[i][j]       ↔    *( tab[i] + j )
tab[i]          ↔    *( tab + i )
```

stąd:

```
tab[i][j]       ↔    *( *( tab + i ) + j )
```


37

ISO/ANSI C - wskaźniki

```
int i, tab[20];
int* x = tab;

  for ( i = 0; i < 20; ++i )
     tab[i] = 2;     // indeksowanie

  while ( x != tab + 20 )
     *x++ = 1;       // wyłuskanie
```


38