ZESTAW 1

1.

- wskaźnik do elementu kl. wektor typu float

```
float wektor::*a;
```

- siedmioelementowo, tablice wskaźników do funkcji int* fun(int)

```
int* (*ptr[7])(int);   albo
typedef int* (*ptr)(int);
ptr tab[7];
```

- zainicjalizuj ten wskaźnik

```
a = &wektor::q;
for(int i=0; i<7; i++)
    tab[i] = fun;
```

2.

```
class wektor {
private:
    float *x, *y, *z;
public:
    float q;
    wektor...;
};
```

Zadeklarować i zdefiniować

- Konstruktor inicjujący wartości x, y, z

```
wektor(float xp, float yp, float zp):
    x(exp), y(eyp), z(ezp) {};
```

wskaźnika na funkcję int* (*ptr)(int)

```
wektor(float xp, float yp, float zp){:
    x(new float(xn)),
    y(new float(yn)),
    z(new float(zn)){};
```

konstruktor kopiujący
wektor(const wektor& w): x(new float(*w.x))
{

funkcję składową dodającą dwa wektory $(x_1+x_2, y_1+y_2, z_1+z_2)$

wektor wektor::dodaj(const wektor& a)
return wektor(*(this->x)+*a.x, *y+*a.y, z*+*a.z);
}

class X{
int i;
public:
static int j;
static ~~void~~ int f(int j);

która z definicja f jest poprawna
dlaczego
a) NIE –
int X::f(int j) {return 2*j*i;}

b) nie ma zakresu

c) NIE
nie ma operatora zakresu

d) OK
int X::f(int j) { return z*j; }

ZESTAW 2

1.

- tablica pięciu wskaźników typu int
int *tab[5];
for (int i=0; i<5; i++)
tab[i] = NULL;
}

- wskaźnik do funkcji char* fun(int, char
char* (*ptr)(int, char)
ptr = fun;

Konstruktor kopiujący

```
wektor(const wektor& w): x(new float(*w.x))
{
```

funkcje składowa dodająca dwa wektory $(x_1+x_2, y_1+y_2, z_1+z_2)$

```
wektor wektor::dodaj(const wektor& a) {
return wektor(*(this->x)+*a.x, *y+*a.y,
              z*+*a.z);
}
```

```
class X{
int i;
public:
static int j;
static int f(int j);
```

Która z definicja f jest poprawna dlaczego

a) NIE –

```
int X::f(int j) {return 2*j*i;}
```

b) nie ma zakresu
nie jest static

c) NIE

nie ma operatora zakresu

d) Ok

```
int X::f(int j) { return x*j; }
```

ZESTAW 2

1.

- tablica pięciu wskaźników typu int

```
int *tab[5];
for (int i=0; i<5; i++)
    tab[i] = NULL;
}
```

- wskaźnik do funkcji char* fun(int, cha

```
char* (*ptr)(int, char)
ptr = fun;
```

```
class string{
  char* wsk;
  int roz; // dł. napisu

public:
  osoba (char n[])
```

konstruktor + konstruktor kopiujący dla inicjalizacyjny

```
#define osoba string
string::osoba(char* n): roz(strlen(n)),
wsk(new char[roz])
{ for(int i=0; i<roz; i++)
    wsk[i] = n[i];
}
```

konstruktor kopiujący

```
string::osoba(const osoba& obj):
wsk(new char[obj.roz]), roz(obj.roz){
  for(int i=0; i<roz; i++){
    wsk[i] = obj.wsk[i];
  }
```

lub

```
#include <cstring>
strncpy(wsk, obj.wsk, roz);
```

3.

```
class A{
public:
    int x;
protected:
    int y;
private:
    int z;
};
class B: public A{
};
```

private nie jest widoczne dla nas protected dostępne, ale można nie odnieść tylko wewnątrz klasy

```
class B: protected A{
};
class B: private A{
};
```

w B dostępne

Dziedziczenie wyższe

```
class A: private B{
public:
  using B::x;
};
```

using (?)

```
class C: public B{
};
```

nieidoczne

ZESTAW 3

1.

• wskaźnik do wskaźnika do znaku

```
char ** znak;
(znak =&wsk;
char* wsk = new char('a');
```

• stały wskaźnik do stałej całkowitej

```
int * const a = new int(a);
int * const a= new int(5);
```

2.

```
class string{
    char * wsk;
    int roz;
public:
    //konstruktor i operator
};
```

• operator przypisania

```
string& operator=(const string& s){
    roz= s.roz;
    if(wsk) delete[] wsk;
    wsk = new char[roz];
    for(int i=0; i<roz; i++){
        wsk[i]= s.wsk[i];
    }
    return *this;
}
```

• destruktor

3. Czego się nie dziedziczy z dla niego:

- operator =
- konstruktor
- destruktor
- konstruktor kopiujący
- przyjaźń

ZESTAW 4

wsk do tablicy 7-el wsk. do funkcji void fun(void)

```
typedef void(*wsk)();
wsk* tab[7];
```

```
wektor {
int *x, *y, *z;
public:
  wektor...;
```

funkcja operatorowa << wypisująca

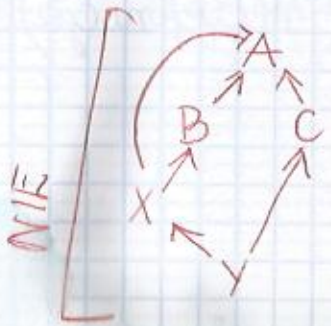
```
std::ostream& operator<<(const std::ostream &
output, const wektor &w) {

return output << '[' << *w.x << ',' << *w.y <<
',' << *w.z << ']';
```

- funkcja operatorowa += dodająca wsp. poszczególnych wektorów

3. Kolejność wywoływania konstruktorów i destruktorów.

- class A{};
  B{};
  C{};
  class X: public A, public B{};
  class Y: public X, public C{};


- K: A, B, X, C, Y
  D: Y, C, X, B, A




K: C,


K: A, C, B, X
D: X, B, C, A

bez virtual


K: A, C, A, B, X

ZESTAW 5

Które z przeładowań funkcji są niepoprawne i dla czego

) OK

) OK - domyślna wartość

)

int tab[] = int *wsk

nie skompiluje się (redefinicja)

2.

```
class l_zes{
    int re, im;
public:
    wektor.
};
```

l_zes(int _re, int _im)

operator int() const { return re; }

explicit (ale nie do tego przykładu)

l_zes(int _re) : re(_re), im(0) { }

ZESTAW 6

1.

- Stały wskaźnik do stałej znakowej

const char* const wsk;

- wskaźnik do funkcji składowej klasy wektor

wektor(wektor::(*ptr)(wektor&);

2.

ZESTAW 10

3.

wf = wi2;

[• na void można wszystko!]

cout << *wv; // void *d;
*d = costam;
nie można tak!

wi1 = *a; // wi1 = &a; OK
typowa inicjalizacja tym

wf = wk; // wf = (float*)wk OK

f* → void*

void* ✗ f*