Klasy, czyli definiowanie własnych typów danych.

Oprócz typu int, double, char tworzymy własny typ danej, np. Wektor lub Punkt. Wewnątrz ciała klasy definiuje się **składniki klasy** oraz **funkcje składowe (metody)**.

Gdy tworzymy **klasę** oznacza to, że tworzymy *szablon*, *nowy typ*, którym będziemy się następnie posługiwać wewnątrz kodu programu.

```
class nowy_typ{ public: int a, b; };
```

Gdy tworzymy **obiekt**, **instancję danej klasy**, oznacza to, że tworzymy konkretny obiekt, zajmujący określone miejsce w pamięci, mający własne, charakterystyczne dla niego wartości składników.

`nowy_typ A;` - deklaracja obiektu A klasy nowy_typ.

`nowy_typ *A;` - deklaracja wskaźnika A na obiekty nowy_typ;

Jeśli stworzyliśmy wskaźnik na obiekt, nie oznacza to, że stworzyliśmy również obiekt, na który nasz wskaźnik wskazuje. Taki wskaźnik możemy sobie wyobrazić jako zawieszoną w przestrzeni strzałkę, która jeszcze nie wskazuje na żaden obszar pamięci, ale *mogłaby* wskazywać na taki, w którym byśmy przechowywali obiekt klasy nowy_typ.

By zadeklarować wskaźnik wraz z obiektem, korzystamy z operatora **new**:

```
nowy_typ *B = new nowy_typ();
```

lub, korzystając z uprzednio stworzonego wskaźnika:

```
A = new nowy_typ();
```

Aby odnieść się do składników obiektu możemy posługiwać się jedną z poniższych notacji:

```
obiekt.składnik
wskaźnik->składnik
referencja.składnik
```

Aby wywołać metodę na danym obiekcie:

`obiekt.metoda();` - dla metody bezargumentowej, **pamiętamy o nawiasach**!

`obiekt.metoda(arg1, arg2, …);` – dla metody przyjmującej argumenty, podajemy je po przecinku.

Przy wywoływaniu metody nie deklarujemy typów podawanych argumentów (nie piszemy np. `int i`, tylko używamy zadeklarowanej *wcześniej* zmiennej `i`). Podobnie dla wskaźników:

```
wskaźnik->metoda(arg1, arg2, …);
```

Na końcu deklaracji klasy należy pamiętać o średniku!

Konstruktory

Konstruktor to specjalna funkcja składowa, która nazywa się tak samo, jak klasa. Jest wywoływana przy tworzeniu („konstrukcji”) obiektu. Wewnątrz konstruktora możemy zamieścić **instrukcje nadające wartości początkowe składnikom definiowanego właśnie obiektu.** Konstruktory:

- Nie zwracają nic, nawet typu „void”! (Uwaga, po napisaniu konstruktorów należy pamiętać, że wszystkie inne metody muszą zwracać jakiś typ (chociażby „void”). Z rozpędu łatwo o tym zapomnieć).

Konstruktorem dla klasy nowy_typ mógłby być:

```
class nowy_typ{ public: int a, b;
      nowy_typ(){a=0;b=0;} - konstruktor domyślny
      nowy_typ(int A, int B){ a=A; b=B;} - konstruktor dwuparametrowy
}
```

Konstruktory wywoływane są automatycznie, gdy program napotyka definicję danego obiektu.

`nowy_typ C;` - w tym momencie wywołany jest konstruktor domyślny

`nowy_typ C(2,3);` - w tym momencie wywołany jest konstruktor dwuparametrowy

Lub też, gdy używamy operatora new:

```
nowy_typ *C = new nowy_typ();
nowy_typ *C = new nowy_typ(2,3);
```

Metody “set” i “get”:

Zadaniem funkcji typu „set” jest ustawienie odpowiednich pól składowych, tzn.:

załóżmy, że mamy w naszej klasie pole `int fIlosc`. Wtedy metody „set” i „get” dla takiego pola:

```
void SetIlosc(int ilosc) //metoda typu „set” / „setter”
{
  fIlosc = ilosc;
}
int GetIlosc() //metoda typu „get” / „getter”
{
  return fIlosc;
}
```

Kompilacja z linii poleceń:

```
g++ <opcje> <lista plików cpp; nigdy pliki h!> –o <nazwa programu>
```

np.

```
g++ –Wall klasa1.cpp klasa2.cpp program.cpp –o program
```

Wskaźnik *this

- Mówi: "teraz odwołasz sie do składowej >TEJ< klasy" (w której jesteś)

- Jego używanie nie jest obligatoryjne (przydaje sie, jeśli z jakiegoś powodu chcemy posiadać takie same nazwy dla argumentów funkcji jak i składników klasy)

Pola prywatne i publiczne:

prywatne: private

Jest dostępne tylko dla funkcji składowych klasy (i funkcji zaprzyjaźnionych z daną klasą). Czasem chcemy ukryć informacje, by nie były dostępne "na zewnątrz" (jest to powszechna praktyka przy tworzeniu dużych programów). Jeśli nie wyszczególnimy etykiety, to wszystkie składniki klasy będą domyślnie prywatne.

publiczne: public

Publiczne składniki mogą być używane zarówno we wnętrzu klasy, jak również spoza jej zakresu.

Funkcje zaprzyjaźnione - funkcje, które mimo, że nie są składnikami klasy maja dostęp do wszystkich (nawet prywatnych i chronionych) składników klasy.

To nie funkcja ma twierdzić, ze przyjaźni sie z klasa, ale sama klasa daje jej dostęp do swoich prywatnych i chronionych składników.

- Funkcja zaprzyjaźniona może być przyjacielem więcej niż jednej klasy!

- Jeśli klasa deklaruje przyjaźń ze wszystkimi funkcjami innej klasy, możemy mieć klasę zaprzyjaźnioną.

- słowo kluczowe: **friend**

Przesyłanie przez wartość

- funkcja pracuje na kopii

Jeśli klasa jest spora, zużywamy dużo zasobów na utworzenie kopii. Lecz czasem właśnie tego chcemy.

```
nowy_typ FunkcjaPrzezWartosc(nowy_typ obiekt);
```

Przesyłanie przez referencje

- funkcja pracuje na oryginale

```
void FunkcjaPrzezReferencje(nowy_typ& obiekt);
```

Wyobraźmy sobie, ze chcemy np zmienic ktoryś z parametrów istniejącego obiektu.

Przeładowywanie funkcji - nadanie jej wielu znaczeń, gdyż istnieje kilka różnych funkcji o tej samej nazwie.

Jako, że obie funkcje robią dokładnie to samo, chcielibyśmy zachować rownież taką samą nazwę. W C++ można tak zrobic - kompilator sam odkryje, czy podawany przez nas obiekt to integer (enum) czy tez string i bazujac na tej informacji wywoła odpowiednią funkcję.

Typy wyliczeniowe: enum. Jest to osobny typ dla wybranego zestawu stałych całkowitych.

Przydaje sie często - jeśli mamy ograniczana liczbę jakiejś opcji. Jeśli to jest jakieś “słowo”, a nie chcemy marnować miejsca na masę takich samych stringów myślimy: skoro mamy np. trzy zawody, to przypiszmy im numery. Informayk - 1, sprzątaczka - 2, ... Jednak używanie nic nie znaczących intów w miejsce konkretnych rzeczy jest zupełnie nieczytelne. Używamy wiec typu wyliczeniowego enum. Przykład użycia:

```
enum zawod{ informatyk=1, sprzataczka=2, manager=3};
```

W ten sposób, jeśli zadeklarujemy taki typ globalnie, to za każdym razem, jeśli napiszemy w kodzie "manager" to program będzie wiedział, ze chodzi o 3. Możemy również zadeklarować enum wewnątrz klasy. Takie postępowanie jest bardzo naturalne - niektóre wyliczenia wiążą sie tylko i wyłącznie z jedną, konkretną klasą. Po prostu wklejamy deklarację enum do wnętrza klasy. Wtedy wewnątrz składników klasy używamy zwyczajnie slow "informatyk" tak, jakby był to dowolny int. Na zewnątrz klasy musimy jednak określić, z jakiego zakresu słowo "informatyk" ma pochodzić: Pracownik::informatyk.