

Објекти, типови, вредности...

Променлива?

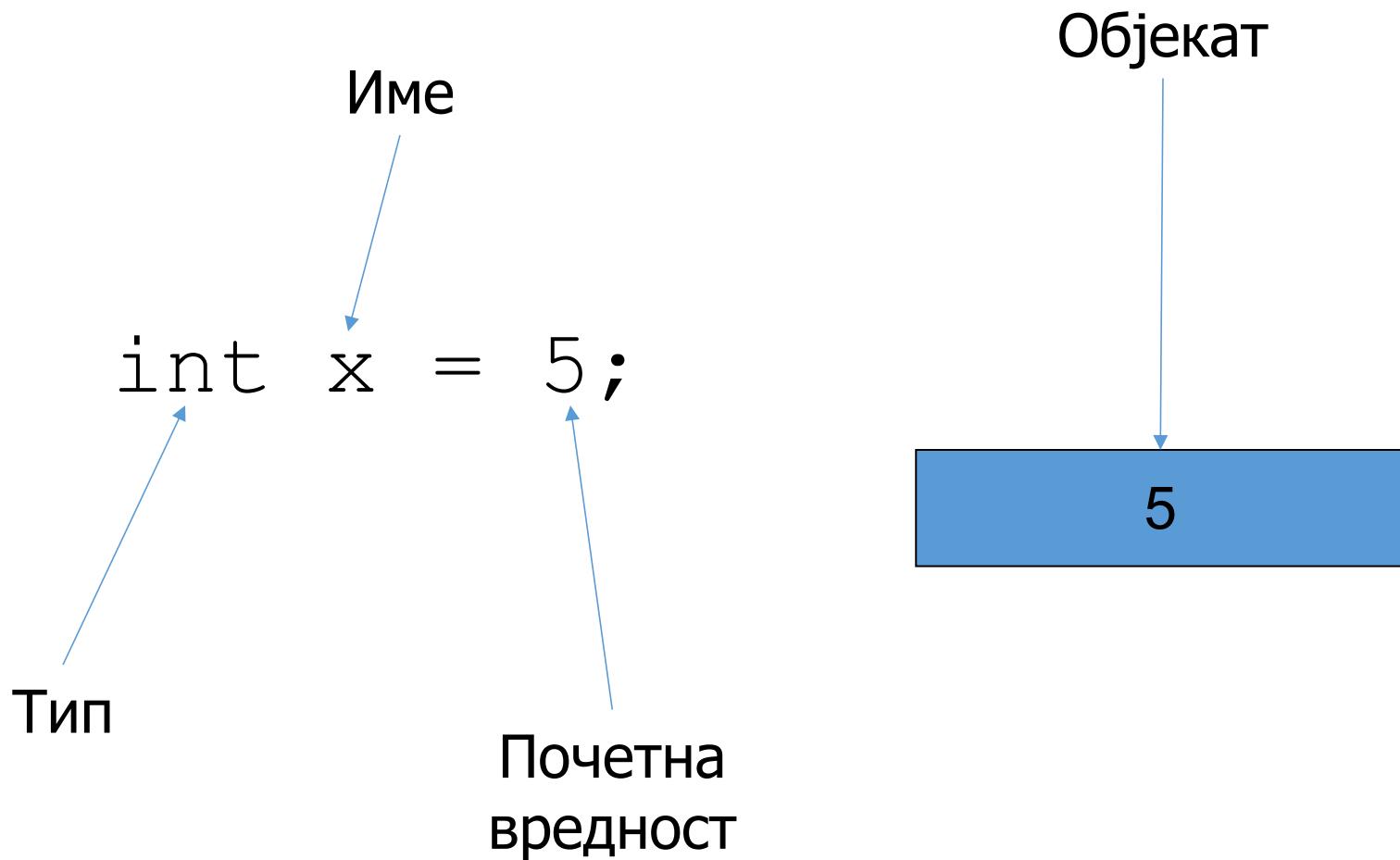
```
int x = 5;
```

Променлива?

```
int x = 5;
```

5

Променълива?



Променљива

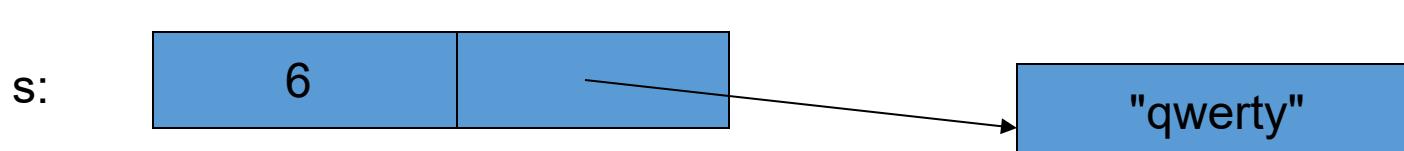
- Објекат је парче меморије (ресурса) које може садржати вредност одговарајућег типа
- Променљива је објекат који има име
- Декларација придржује име објекту

```
int a = 7;
```

```
char c = 'x';
```

```
complex<double> z(1.0,2.0);
```

```
string s = "qwerty";
```



Име

- Имена у Це++ програму
 - Почињу словом, садрже слова, цифре и доњу црту
 - **x, number_of_elements, Fourier_transform, z2**
 - Ово нису имена:
 - **12x**
 - **time\$to\$market**
 - **main line**
 - Иако то јесте могуће, не треба почињати имена доњом цртом: **_foo**
 - таква имена су резервисана за системске називе
 - Кључне речи не могу бити имена
 - На пример:
 - **int**
 - **if**
 - **while**
 - **double**

Типови

- Це++ нуди основни скуп типова
 - Нпр. `bool`, `char`, `int`, `double`
 - Зову се „уграђени типови“
- Програмери могу увести нове типове
 - Зову се „кориснички дефинисани типови“ (или само „кориснички типови“) или „изведени типови“
 - О томе касније у предмету
- Стандардна Це++ библиотека нуди још неке типове
 - Нпр. `string`, `vector`, `complex`
 - У суштини, ово су кориснички типови. (Сваки корисник може направити свој вектор и сл.)
 - Али ћемо на овом предмету сматрати да су увек присутни, тј. користићемо их од самог почетка и кад год нам је потребно

Објекти и променљиве: Це++ наспрам Јаве

- Нема имплицитне индирекције. Променљива директно представља објекат.

К x;

К x;

Објекти и променљиве: Це++ наспрам Јаве

- Нема имплицитне индирекције. Променљива директно представља објекат.

```
class K {  
public: int a;  
}  
  
K x;
```

```
x.a = 5;
```

```
class K {  
public int a;  
}  
  
K x = new K();  
x.a = 5;
```

Објекти и променљиве: Це++ наспрам Јаве

- Нема имплицитне индирекције. Променљива директно представља објекат.
- Директан рад са објектима је обично бржи.
- Али некада није довољно „флексибилилан“.
- У Це++-у можемо и индиректно приступати објектима, али за то се користи другачија синтакса, тј. индиректан рад је експлицитан.
- То значи да Це++ пружа више могућности.
- Али то значи и да је Це++ „тежи“, јер програмер мора да доносе одлуку.

Тип

- Тип променљиве одређује које вредности она може да има и које операције можемо обавити над њом (и шта је њихово значење)

Интеџери и стрингови

- Стингови
 - `cin >>` учитава реч
 - `cout <<` исписује цео стринг
 - + спаја стрингове
 - `+= s` додаје стринг `s` на крај
 - `++` није валидно
 - - није валидно
 - ...
- Интеџери
 - `cin >>` учитава број
 - `cout <<` исписује број (у одређеном формату)
 - + сабира
 - `+= n` увећава за `n`
 - `++` увећава за `1`
 - - одузима
 - ...

Тип одређује које су операције валидне и шта је њихов смисао

Декларација, дефиниција и иницијализација

int a = 7;

a: 7

int b = 9;

b: 9

char c = 'a';

c: 'a'

double x = 1.2;

x: 1.2

string s1 = "Hello, world";

s1: 12 | "Hello, world"

string s2 = "1.2";

s2: 3 | "1.2"

Декларација, дефиниција и иницијализација

int a;

a:

?

int b;

b:

?

char c;

c:

?

double x;

x:

?

string s1;

s1:

???

string s2;

s2:

???

Аутоматско закључивање типа

- Може се препустити компјултеру да закључи тип променљиве из типа иницијализационог израза
 - `auto x = 1;`
 - `auto y = 'c';`
 - `auto d = 1.2;`
- Али:
 - `auto s = "Howdy";` // шта је тип "Howdy"? Није string :-/
- Кад је стварно корисно:
 - `auto sq = sqrt(2);`
- Наравоученије: **auto** не служи да ви не бисте морали да знате ког типа треба да буду неке променљиве, него да вам скрати писање и учини код лакшим за прераду (а посебно је корисно код генеричког програмирања).
- Реч **auto** се не сме користити на овом предмету, осим на индивидуалном пројекту.

Типови и литерали

- Уграђени типови
 - Логички (Бул) тип
 - `bool`
 - Знаковни тип
 - `char`
 - Интеџер тип
 - `int`
 - и `short` и `long`
 - Реални тип у покретном зарезу
 - `double`
 - и `float`
 - Типови из стандардне библиотеке:
 - `string`
 - `vector<int>`
- Логички литерали
 - `true` `false`
 - Знаковни литерали
 - `'a'`, `'x'`, `'4'`, `'\n'`, `'$'`
 - Целобројни литерали
 - `0`, `1`, `123`, `-6`, `034`, `0xa3`
 - Литерали реалног типа
 - `1.2`, `13.345`, `.3`, `-0.54`, `1.2e3`, `.3F`
 - Стинг литерали `"asdf"`,
`"Zdravo svete!"`

Додела и увећање

a:

`int a = 7 или int a{7}//иницијализација на 7`

7

`a = 9; // додела: сада је вредност 9`

9

`a = a + a; // додела: сада је вредност 2a`

18

`a += 2; // увећање за 2`

20

`++a; // увећање за 1 (инкрементирање)`

21

Типска безбедност (енгл. type-safety)

- Правила која намеће језик:
 - Сваки објекат може безбедно бити коришћен само у складу са својим типом
 - Објекат сме безбедно бити коришћен само након иницијализације
 - Над објектом могу бити безбедно коришћени само операнди који су за његов тип дефинисани
 - Свака операција која је дефинисана над типом оставља валидну вредност у објекту тог типа
- Идеал 1: статичка типска безбедност
 - Програм који небезбедно рукује објектима са становишта њихових типова неће се ни превести
- Идеал 2: динамичка типска безбедност
 - Небезбедно руковање објектима са становишта њиховог типа биће откријено током извршавања

Типска безбедност (енгл. type-safety)

- Типска безбедност је важно
 - Трудите се да је не нарушите
 - Компајлер ће вам много помоћи
 - иако ћете можда гунђати када не преведе код за који сте ви сигури да ваља
- Це++ није (у потпуности) статички типски безбедан језик
 - Ниједан језик у широј употреби није такав
 - Потпуна статичка типска безбедност може значајно смањити изражајност језика
- Це++ није (у потпуности) ни динамички типски безбедан језик
 - Многи језици јесу у потпуности динамички типски безбедни
 - Али и то у одређеној мери смањује изражајност и често чини код већим споријим
- (Скоро) све што будемо учили на овом курсу ће бити типски безбедно

Нарушавање типске безбедности (имплицитно сужавање)

// Пажња: Це++ не спречава програмера да смести вредност из велике
// променљиве у малу променљиву (мада ће компајлер издати упозорење)

```
int main()
{
    int a = 20000;
    char c = a;
    int b = c;
    if (a != b)
        cout << "Ups!: " << a << "!=" << b << '\n';
    else
        cout << "OK";
}
```

a 20000

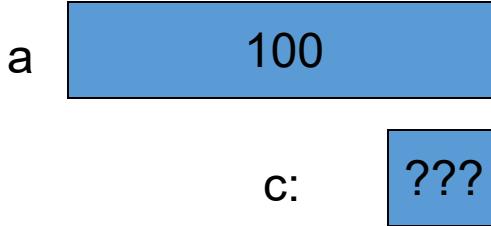
c: ???

- Проверите шта ће бити вредност у **b** на вашем рачунару

Нарушавање типске безбедности (имплицитно сужавање)

// Пажња: Це++ не спречава програмера да смести вредност из велике
// променљиве у малу променљиву (мада ће компајлер издати упозорење)

```
int main()
{
    int a = 100;
    char c = a;
    int b = c;
    if (a != b)
        cout << "Ups!: " << a << "!=" << b << '\n';
    else
        cout << "OK";
}
```



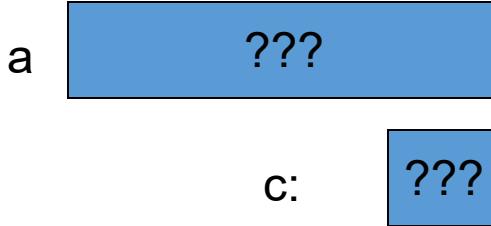
a 100

c: ???

Нарушавање типске безбедности (имплицитно сужавање)

// Пажња: Це++ не спречава програмера да смести вредност из велике
// променљиве у малу променљиву (мада ће компајлер издати упозорење)

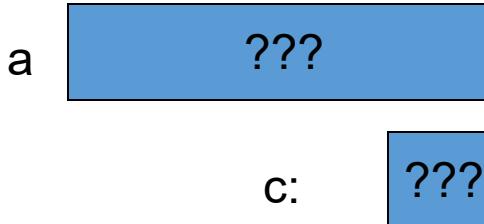
```
void foo(int a)
{
    char c = a;
    int b = c;
    if (a != b)
        cout << "Ups!: " << a << "!=" << b << '\n';
    else
        cout << "OK";
}
```



Нарушавање типске безбедности (имплицитно сужавање)

// Динамичка типска безбедност кошта!

```
void foo(int a)
{
    char c = check(a);
    int b = c;
    if (a != b)
        cout << "Ups!: " << a << "!=" << b << '\n';
    else
        cout << "OK";
}
char check(int a) {
    if (a > numeric_limits<char>::max()
        && a < numeric_limits<char>::min()) return 0;
    else return a; }
```



a ???

c: ???

Нарушавање типске безбедности

(Неиницијализоване променљиве)

// Пажња: Це++ не спречава програмера да користи вредност променљиве
// пре него што је она иницијализована (мада обично пријављује
// упозорење)

```
int main()
{
    int x; // x добија неодређену почетну вредност
    char c; // c добија неодређену почетну вредност
    double d; // d добија неодређену почетну вредност
    // Није сваки 32битни податак валидна double вредност!!!
    double dd = d; // могућа грешка током извршавања!
    cout << " x: " << x << " c: " << c << " d: " << d << '\n';
}
```

- Увек иницијализујте своје променљиве, осим ако немате јако добар разлог да то не урадите. Имајте на уму: „дебаг мод“ може иницијализовати променљиве, али „рилис мод“ то неће урадити.

Технички детаљи

- У рачунару (меморији) све су бити (0 или 1); тип је оно што даје смисао битима

(бинарно) **01100001** је int **97** а char '**a**'

(бинарно) **01000001** је int **65**, а char '**A**'

(бинарно) **00110000** је int **48**, а char '**0**'

```
char c = 'a';
cout << c;
int i = c;
cout << i;
```

- Исто као и у свакодневном животу:
 - Шта значи „42“?
 - Недостаје контекст, односно јединице:
 - Дужина? Висина? У метрима, или нешто друго? Температура? Кућни број?

Мало о ефикасности

- За почетак се нећемо бавити ефикасношћу
 - Усредсредите се на исправност и једноставност
- Це++ је наследник Цеа, који је системски програмски језик
 - Це++-ови уgraђени типови се директно пресликају на главне елементе рачунарске архитектуре
 - **char** је величине бајта (Пажња: бајт **није** 8 бита у општем случају)
 - **int** се смешта у реч (реч је „најприроднија“ величина циљне машине)
 - **double** одговара величини наменског регистра
 - Це++-ове уgraђене операције директно се пресликају на уобичајене операције подржане машинским инструкцијама
 - Це++ омогућава директан приступ великим делу могућности које нуди већина модерног хардвера
- Це++ помаже програмерима да направе безбедније, елегантније и ефикасније нове типове и операције коришћењем основних уgraђених типова и операција.
 - Пример: **string**

Мало философирања

- Програмирање, као и остале гране инжењерства, захтева прављење компромиса.
- Идеални захтеви су врло често, практично увек, међусобно супротстављени. Програмер мора одлучити шта је од тога важно (важније) за дати програм.
 - Типска безбедност
 - Перформансе (ефикасност)
 - Преносивост
 - Компатибилност (са другим системима и кодом)
 - Лакоћа прављења
 - Лакоћа одржавања
- И немојмо занемарити исправност и испитивање
- Типска безбедност и преносивост су подразумевано пожељне

Кратак поглед на Це++11, 14, 17, 20...

- Це++ је жив језик и мења се.
- Постоји неколико стандарда:
 - Из 1998. (Це++98)
 - Из 2003. (Це++03)
 - Из 2011. (Це++11)
 - Из 2014. (Це++14)
 - Из 2017. (Це++17)
 - Из 2020. (Це++20)
- Велика већина кода који будемо писали у ООП2 користи ствари које су важиле и у Це++03 стандарду, али ослањаћемо се на све ново што нам је потребно, а уведено је у каснијим верзијама стандарда.

Улаз и излаз

```
// учитај име:  
#include "std_lib_facilities.h" // заглавље на нашем курсу  
  
int main()  
{  
    cout << "Please enter your first name (followed " << "by 'enter'):\n";  
    string first_name;  
    cin >> first_name;  
    cout << "Hello, " << first_name << '\n';  
    return 0;  
}
```

Учитавање стринга

```
// учитај име и презиме:  
int main()  
{  
    cout << "please enter your first and second names\n";  
    string first;  
    string second;  
    cin >> first >> second; // чита два стринга  
    string name = first + " " + second; // спаја стрингове  
                                // убацујући размак између њих  
    cout << "Hello, "<< name << '\n';  
    return 0;  
}
```

Улаз и тип

- Читамо у променљиву
 - `first_name`
- Свака променљива има тип
 - `string`
- Тип променљиве одређује које вредности она може да има и које операције можемо обавити над њом (и шта је њихово значење)
 - `cin>>first_name;` учитава све знакове до празног знака (енгл. *white space*) што се сматра јеном речју
 - Празан знак: размак, табулатор, нови ред, ...

Интецери

```
// учитај име и године:  
  
int main()  
{  
    cout << "please enter your first name and age\n";  
    string first_name; // стринг променљива  
    int age; // интецер променљива  
    cin >> first_name >> age; // учитавање  
    cout << "Hello, " << first_name << " age " << age << '\n';  
}
```

Интеџери и стрингови

- Стингови
 - `cin >>` учитава реч
 - `cout <<` исписује цео стринг
 - + спаја стрингове
 - `+= s` додаје стринг `s` на крај
 - `++` није валидно
 - - није валидно
 - ...
- Интеџери
 - `cin >>` учитава број
 - `cout <<` исписује број (у одређеном формату)
 - + сабира
 - `+= n` увећава за `n`
 - `++` увећава за `1`
 - - одузима
 - ...

Тип одређује које су операције валидне и шта је њихов смисао

Имена

- Имена у Це++ програму
 - Почињу словом, садрже слова, цифре и доњу црту
 - **x, number_of_elements, Fourier_transform, z2**
 - Ово нису имена:
 - **12x**
 - **time\$to\$market**
 - **main line**
 - Иако то јесте могуће, не треба почињати имена доњом цртом: **_foo**
 - таква имена су резервисана за системске називе
 - Кључне речи не могу бити имена
 - На пример:
 - **int**
 - **if**
 - **while**
 - **double**

Једноставна аритметика

```
int main()
{
    cout << "please enter a floating-point number: ";
    double n;
    cin >> n;
    cout << "n == " << n
        << "\nn+1 == " << n+1 // '\n' значи "нова линија"
        << "\nthree times n == " << 3*n
        << "\ntwice n == " << n+n
        << "\nn squared == " << n*n
        << "\nhalf of n == " << n/2
        << "\nsquare root of n == " << sqrt(n) // библиотечка функција
        << endl; // алтерантивни начин за саопштавање нове линије
}
```

Једноставна аритметика

```
int main() // конверзија инча у сантиметре
{
    const double cm_per_inch = 2.54; // сантиметри по инчуу
    int length = 1; // дужина у инчима
    while (length != 0) // length == 0 служи за излазак
    {
        cout << "Please enter a length in inches: ";
        cin >> length;
        cout << length << "in. = " << cm_per_inch*length << "cm.\n";
    }
}
```

- Блок наредби се понавља докле год је услов тачан