SFML - Vrijeme i satovi

Objektno programiranje - 10. vježbe (1. dio)

dr. sc. Sebastijan Horvat

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

22. svibnja 2024. godine

Vrijeme i satovi

- prije nego promotrimo rad s vremenom u SFML-u, nije loše pogledati kako se može mjeriti vrijeme pomoću prog. jezika C++
- prije C++11 standarda: strukture i funkcije za rad s vremenom naslijeđene iz programskog jezika C (C-ova biblioteka `time` $\Rightarrow$ uključujemo **`ctime`**)

Primjer. Ispis broja sekundi od epohe (pokrenuti više puta):

```
#include <iostream>
#include <ctime>
using namespace std;

int main() {
    time_t s;
    s = time(nullptr);
    cout << s << endl;    //npr. 1685367686
    return 0;
}
```

Tip `time_h` i funkcija `time`

- tip **`time_h`** - iako nije definirano C standardom, gotovo uvijek drži cjelobrojnu vrijednost - funkcija **`time`** predstavlja broj sekundi od **epohe** (00:00h, 1. siječnja 1970, UTC[1])
- UNIX inženjeri odabrali tu epohu jer im se činila prikladnom, a preciznost je promijenjena u sekunde kako ubrzo ne bi došlo do *overflowa* (vidi tzv. problem 2038. godine)
- argument funkcije je null-pokazivač (često se piše i: `time(0)`) ili pokazivač na `time_t` varijablu u koju želimo spremiti sekunde od epohe

⇒ kod u prethodnom primjeru mogli smo napisati ovako:

```
time_t s;
time(&s);
cout << s << endl;
```

[1] Svjetsko vrijeme (*Universal Time Coordinated*) - vrijeme nulte vremenske zone

Dobivanje vremena u čitljivom formatu (za čovjeka)

```
char *ctime(const time_t *time);
```

- vraća pokazivač na C-ovski string koji predstavlja lokalno vrijeme za dani broj sekundi od epohe u obliku:

  dan_tjedna mjesec datum sati:minute:sekunde godina

Primjer.

```
time_t s = time(nullptr);
char* t = ctime(&s);
cout << t << endl;
```

Primjer ispisa:

```
Mon May 29 16:18:02 2023
```

Primjer. Objasniti ispis ako prvu gornju liniju primjera zamijenimo s: `time_t s = 100;` (podsjetnik: Hrvatska je na UTC+1!)

```
Thu Jan 1 01:01:40 1970
```

Struktura tipa `tm`

```
struct tm {
    int tm_sec;   //sekunde (0 do 61)
    int tm_min;   //minute (0 do 59)
    int tm_hour;  //sati (0 do 24)
    int tm_mday;  //dani (1 do 31)
    int tm_mon;   //mjesec (0 do 11)
    int tm_year;  //godina (od 1900)
    int tm_wday;  //dani od nedjelje
    int tm_yday;  //dani od 1. siječnja
    int tm_isdst; //zastavica: ljetno rač. vremena
}
```

```
struct tm *gmtime(const time_t *time);
```

- vraća pokazivač na popunjenu `tm` strukturu s podacima o trenutnom UTC vremenu - pretvaranje u (pokazivač na) C-ovski string dobivamo funkcijom:

```
char * asctime (const struct tm * time);
```

Struktura tipa `tm` - primjer

Primjer.

```
time_t s = time(0);
char *t = ctime(&s);
cout << t; //Sat Feb 25 16:47:46 2023
tm *v = gmtime(&s);
cout << v->tm_sec   << endl  //46
     << v->tm_min   << endl  //47
     << v->tm_hour  << endl  //15
     << v->tm_mday  << endl  //25
     << v->tm_mon   << endl  //1
     << v->tm_year  << endl  //123
     << v->tm_wday  << endl  //6
     << v->tm_yday  << endl  //55
     << v->tm_isdst << endl; //0
char *str = asctime(v);
cout << str; //Sat Feb 25 15:47:46 2023
```

Napomene

- `tm_sec` - do 61 (noviji standardi 60) zbog prijestupnih sekundi
- `tm_hour` - vrijednost 24 je zapravo 00:00 idućeg dana
- *Daylight Saving Time* zastavica (`tm_isdst`) - veća od nule (ljetno računanje vremena), nula (ljetnog računanja vremena nije na snazi), manje od nule ako info o tome nije dostupna
- ljetno vrijeme (Hrvatska 2024.): od 02:00 31. ožujka do 03:00 27. listopada (tj. zadnja nedjelja ožujka do zadnja nedjelja listopada)
- za razliku od prethodnog, funkcija

```
struct tm *localtime(const time_t *time);
```

vraća pokazivač na `tm` strukturu s podacima o lokalnom vremenu

Lokalno vs. UTC vrijeme

Lijevo je kao u prethodnom primjeru, a desno je prikaz izlaza za primjer koji umjesto `gmtime` koristi `localtime`.

	`gmtime` Mon May 29 **17**:09:06 2023	`localtime` Mon May 29 **19**:09:06 2023
v->tm_sec	6	6
v->tm_min	9	9
v->tm_hour	**17**	**19**
v->tm_mday	29	29
v->tm_mon	4	4
v->tm_year	123	123
v->tm_wday	1	1
v->tm_yday	148	148
v->tm_isdst	**0**	**1**

Pomoću `ctime` dobivamo: "Mon May 29 19:09:06 2023"

Funkcija `mktime`

```
time_t mktime (struct tm *timeptr);
```

- obratno od `localtime`
- vraća `time_t` vrijednost koja predstavlja lokalno vrijeme čiji podaci se nalaze u `tm` strukturi na koju pokazuje `timeptr`
- pri pretvorbi se članovi `tm_wday` i `tm_yday` zanemaruju
- ostali članovi se interpretiraju čak i izvan njihovih smislenih raspona

Primjer.

```
time_t s = time(0);
cout << ctime(&s); //Mon May 29 19:34:46 2023
tm *v = localtime(&s);
s = mktime(v);
cout << ctime(&s); //Mon May 29 19:34:46 2023
```

Primjer. Ispis dana u tjednu za zadani datum

- funkcija `mktime` će prilagoditi vrijednosti koje ne odgovaraju ostalim članovima (`tm_wday` i `tm_yday`)

Primjer.

```
time_t sek = time(0);
struct tm *info = localtime(&sek);
vector<string> tjedan = {"nedjelja","ponedjeljak",
   "utorak","srijeda","cetvrtak","petak","subota"};
cout << "Unesite dan:  " << endl;
cin >> info->tm_mday;
cout << "Unesite mjesec:  " << endl;
cin >> info->tm_mon;
info->tm_mon--;
cout << "Unesite godinu:  " << endl;
cin >> info->tm_year;
info->tm_year -= 1900;
mktime(info);
cout << tjedan[info->tm_wday] << endl;
```

Funkcija `difftime`

```
double difftime(time_t time1, time_t time2);
```

- vraća razliku u sekundama između `time1` i `time2` (tj. vraća `time1` - `time2`)

Primjer.

```
time_t tstart, tkraj;
time(&tstart);
tkraj = tstart + 5;
cout << difftime(tstart,tkraj) << ", "
     << difftime(tkraj,tstart) << endl;
```

Ispis: -5, 5

Biblioteka `chrono`

- kolekcija tipova i funkcija za rad s datumom i vremenom
- dio STL-a, uključeno u sve verzije od C++11
- zašto `chrono`: satovi za mjerenje vremena različiti na različitim sustavima $\Rightarrow$ potrebno unaprijediti preciznost mjerenja vremena

Primjer. Izmjerimo vrijeme izvršavanja funkcije:

```
#include <iostream>
#include <chrono>
#include <ctime>
using namespace std;

size_t funkcija(){
    size_t br = 0;
    for(int i = 0; i < 2000; i++)
        for(int j = 0; j < 1500; j++)
            br++;
    return br;
}
```

Satovi

- služe za dobivanje vremena
- sastoje se od početne točke (epohe) i stopa otkucaja
- tri tipa satova - koristimo **system_clock** ako želimo raditi sa satom sustava
- imaju funkciju članicu **now()** - vraća koliko je vremena prošlo od početne točke sata - to predstavlja predložak klase **time_point** (parametar predloška je tip sata koji koristimo) - primjer:

```
time_point<system_clock> t = system_clock::now();
```

- možemo računati s tim tipom - primjerice, možemo zbrojiti dva takva vremena (ali moraju se odnositi na isti tip sata!)
- **duration** - predložak koji predstavlja vremenski interval (ne ovisi o tipu sata!)
- po *defaultu* vrijednost u sekundama (vrijednost vraća funkcija članica **count**) - tip navodimo kao prvi parametar predloška

Nastavak prethodnog primjera

```
int main() {
  chrono::time_point<chrono::system_clock> start,
      kraj;
  start = chrono::system_clock::now();
  cout << funkcija() << endl;
  kraj = chrono::system_clock::now();
  chrono::duration<double> uk = kraj - start;
  time_t t = chrono::system_clock::to_time_t(kraj);
  std::cout << "Zavrseno " << ctime(&t)
      << "ukupno vrijeme: " << uk.count()
      << " sekundi." << endl;
  return 0;
}
```

- koristimo imenički prostor **std::chrono**
- u gornjem kodu koristili **to_time_t** kako bi `duration` pretvorili u `time_t` tip (za kasniji ispis podataka pomoću `ctime`)

Milisekunde

- primjer ispisa za prethodni kod:

```
3000000
Zavrseno Mon May 29 22:36:03 2023
ukupno vrijeme: 0.0056243 sekundi.
```

- zajedno s vrijednosti, `duration` sadrži i omjer (zaglavlje **`ratio`**) koji određuje jedinice vremena koje vrijednost predstavlja
- *default* je 1:1 za sekunde (za milisekunde je 1:1000, a za mikrosekunde 1:1000000 - razni omjeru mogu se vidjeti ovdje)

Primjer. Promjene u prethodnom primjeru za broj milisekundi:

```
#include <ratio>

...

chrono::duration<double, milli> uk = kraj - start;
```

Rad s vremenom u SFML-u

- sve klase i funkcije koje rade s vremenom koriste klasu **`sf::Time`** - predstavlja vremenski interval (tj. vrijeme između dvije vremenske točke)
- vrijednost se može konstruirati iz sekundi, milisekundi i mikrosekundi

Primjer. Sva sljedeća tri objekta imaju istu vrijednost:

```
sf::Time t1 = sf::microseconds(10000);
sf::Time t2 = sf::milliseconds(10);
sf::Time t3 = sf::seconds(0.01f);
```

Ovako dobivamo obratne pretvorbe (iz vrijednosti vremena u sekunde, milisekunde ili mikrosekunde):

```
sf::Time t = ...;

sf::Int64 us = t.asMicroseconds();
sf::Int32 ms = t.asMilliseconds();
float s = t.asSeconds();
```

`sf::Time` podržava uobičajene aritmetičke operacije

Iz implementacije (datoteka `Time.hpp`):

```
bool operator ==(Time left, Time right);
bool operator !=(Time left, Time right);
bool operator <(Time left, Time right);
bool operator >(Time left, Time right);
bool operator <=(Time left, Time right);
bool operator >=(Time left, Time right);
Time operator -(Time right);
Time operator +(Time left, Time right);
Time& operator +=(Time& left, Time right);
Time operator -(Time left, Time right);
Time& operator -=(Time& left, Time right);
Time operator *(Time left, float right);
Time operator *(Time left, Int64 right);
Time operator *(float left, Time right);
Time operator *(Int64 left, Time right);
```

Nastavak s prethodnog slajda

```
Time& operator *=(Time& left, float right);
Time& operator *=(Time& left, Int64 right);
Time operator /(Time left, float right);
Time operator /(Time left, Int64 right);
Time& operator /=(Time& left, float right);
Time& operator /=(Time& left, Int64 right);
float operator /(Time left, Time right);
Time operator %(Time left, Time right);
Time& operator %=(Time& left, Time right);
```

Primjer.

```
sf::Time t1 = ...;
sf::Time t2 = t1 * 2;
sf::Time t3 = t1 + t2;
sf::Time t4 = -t3;
bool b1 = (t1 == t2);
bool b2 = (t3 > t4);
```

Mjerenje vremena u SFML-u

- jednostavna klasa za mjerenje vremena: **`sf::Clock`**
- daje najpreciznije moguće mjerenje vremena koje OS podržava (općenito u mikrosekundama ili nanosekundama)
- osigurana je monotonost (u smislu da vrijeme ne može ići unatrag čak i ako se promijeni vrijeme sustava)
- konstruktor pokreće sat - nakon toga imamo dvije funkcije na raspolaganju:

```
Time sf::Clock::restart()
```

  - stavlja brojač vremena na nulu
  - vraća vrijeme koje je proteklo od posljednjeg (ponovnog) pokretanja sata (tj. konstrukcije ili posljednjeg restarta)

```
Time sf::Clock::getElapsedTime() const
```

  - vraća vrijeme proteklo od (posljednjeg) pokretanja sata

Primjer mjerenja vremena

Primjer.

```
sf::Clock sat;
...
sf::Time v1 = sat.getElapsedTime();
std::cout << v1.asSeconds() << std::endl;
sat.restart();
...
sf::Time v2 = sat.getElapsedTime();
std::cout << v2.asSeconds() << std::endl;
```

Napomena: mogli smo izbaciti gornji `sat.restart();` te napisati ovako: `sf::Time v1 = sat.restart();` (no, tada `v2` uključuje i vrijeme potrebno za ispis `v1` vremena).

Primjena: Igra na sporijim računalima

- prošla prezentacija: kako ograničiti FPS na prebrzim računalima

```
void Prozor::Stvori() {
  auto stil = (cijeliZaslon ? sf::Style::Fullscreen
                : sf::Style::Default);
  prozor.create(sf::VideoMode(velicina.x,
                velicina.y,32), naslov, stil);
  prozor.setFramerateLimit(500);
}
```

- Pitanje: Što sa sporijim računalima?

Određivanje brzine formule

- kod koji trenutno imamo:

```
void Igra::pomakniFormulu() {
    ...
    sprite.move(pomak * smjer);
}
```

- rješenje problema: izmjerit ćemo vrijeme potrebno za jedan ciklus igre (jedan prolaz `while` petljom iz `main` funkcije) i prema tome prilagoditi **brzinu** formule
- podsjetnik:

$$\text{brzina} = \frac{\text{put}}{\text{vrijeme}}$$

$$\Rightarrow \quad \text{put} = \text{brzina} \cdot \text{vrijeme}$$

Izmjene u klasi **Igra** (datoteka `Igra.h`)

```
class Igra {
  public:
    ...
    sf::Time protekloVrijeme();
    void restartSata();
  private:
    ...
    sf::Clock sat;
    sf::Time vrijeme;
};

sf::Time Igra::protekloVrijeme() {
    return vrijeme;
}

void Igra::restartSata() {
    vrijeme = sat.restart();
}
```

Mjerenje vremena izvrš. jedne iteracije glavne petlje

```
int main() {
    Igra igra;
    while (!igra.dohvatiProzor()->jelGotov()) {
        igra.obradiUlaz();
        igra.update();
        igra.renderiraj();
        igra.restartSata();
    }

    return 0;
}
```

Brzina formule

```
class Igra {
    ...
  private:
    ...
    float brzina = 200; //br. piksela u sekundi
    float kutnaBrzina = 90; //br. stupnjeva u sek.
};
```

- u konstruktoru za `Igra::Igra()`:

```
pomak = pocetni_pomak = sf::Vector2f(0.f, -1.f);
```

- u funkciji `Igra::obradiUlaz()` (za jedinični vektor):

```
kut -= kutnaBrzina * protekloVrijeme().asSeconds();
kut += kutnaBrzina * protekloVrijeme().asSeconds();
```


Brzina formule (nastavak)

- u funkciji `Igra::pomakniFormulu()`:

```
sprite.move(pomak * smjer
    * brzina * protekloVrijeme().asSeconds());
```

Zadatak. Testirati dobiveno za različite FPS-ove:

```
void Prozor::Stvori() {
    ...
    prozor.setFramerateLimit(100);  ←
}
```

Kako dobiti fiksan broj iteracija u sekundi?

1. promjena

- umjesto da imamo samo vrijeme posljednje iteracije, promatramo ukupno vrijeme dosadašnjih iteracija

```
void Igra::restartSata() {
    vrijeme += sat.restart();
}
```

- ako želimo `brIteracija` iteracija u sekundi tada jedna iteracija treba trajati:

$$\text{trajanjeIteracije} = \frac{1.f}{\text{željeni\_broj\_iteracija\_u\_sekundi}}$$

Fiksan broj iteracija u sekundi

2. promjena

- Ako nešto želimo napraviti npr. 60 puta u sekundi, tada koristimo sljedeći kod:

```
float vrijemeIteracije = 1.0f / 60.0f;
if(vrijeme.asSeconds() >= vrijemeIteracije) {
    ...  //ovdje ono što želimo 60x u sek
    vrijeme -= sf::seconds(vrijemeIteracije);
}
```

- oduzimanje u gornjem kodu predstavlja resetiranje ciklusa i omogućava simuliranje izvršavanja konstantnom brzinom