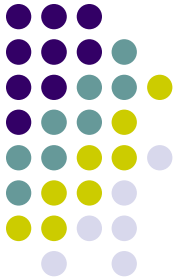
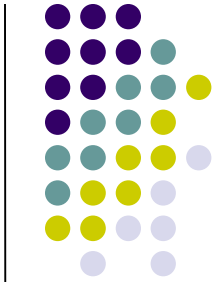
NIT



assert - претпоставке

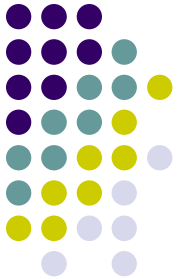
```
#include <assert.h>
assert(expression);
```

- Асерт је макро-функција.
- Израз који јој се прослеђује мора бити логички израз - тврдња. Тврдња представља програмерову претпоставку. Приликом извршавања та претпоставка ће на овај начин бити проверена и ако је нетачна прекинуће се извршавање програма уз одговарајућу информацију.
- Погледајте о асерту у C99 стандарду

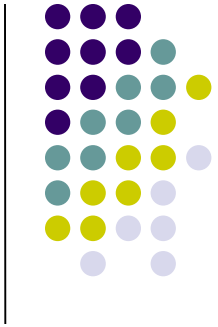


```
int main() {
  int_fast16_t a, b;
  printf("Unesite jedan ceo broj od 1 do 10: ");
  scanf("%"SCNdFAST16_T, &a);

  printf("Unesite jos jedan ceo broj od 1 do 10: ");
  scanf("%"SCNdFAST16_T, &b);

  neka_obrada(a, b);
  return 0;
}
```

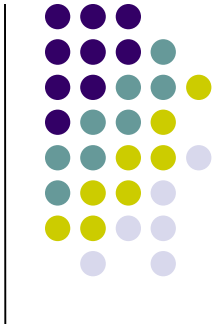
NIT



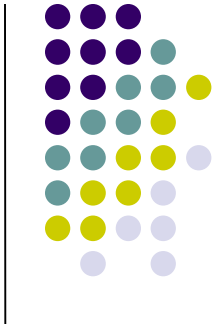
```
int main() {
  int_fast16_t a, b;
  printf("Unesite jedan ceo broj od 1 do 10: ");
  scanf("%"SCNdFAST16_T, &a);
  while (a < 1 || a > 10) {
    printf("Ostav. Od 1 do 10! Ponovo:");
    scanf("%"SCNdFAST16_T, &a);
  }
  printf("Unesite jos jedan ceo broj od 1 do 10: ");
  scanf("%"SCNdFAST16_T, &b);
  while (b < 1 || b > 10) {
    printf("Ostav. Od 1 do 10! Ponovo:");
    scanf("%"SCNdFAST16_T, &b);
  }

  neka_obrada(a, b);
  return 0;
}
```



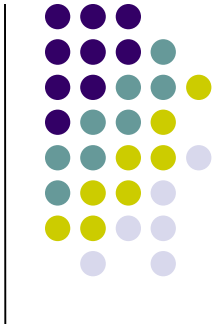
```
// Функција прима два цела броја од 1 до 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
  int_fast16_t p = x * x * x * x;
  int_fast16_t q = y * y * y * y;
  return p + q;
}
```



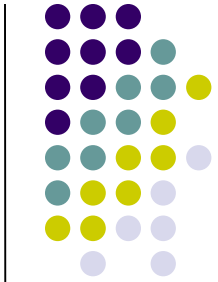
```
// Функција прима два цела броја од 1 до 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
  while (x < 1 || x > 10) {
    printf("Ostav. Od 1 do 10! Ponovo:");
    scanf("%"SCNdFAST16_T, &x);
  }
  while (y < 1 || y > 10) {
    printf("Ostav. Od 1 do 10! Ponovo:");
    scanf("%"SCNdFAST16_T, &y);
  }
  int_fast16_t p = x * x * x * x;
  int_fast16_t q = y * y * y * y;
  return p + q;
}
```



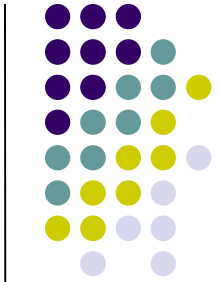
```
// Функција прима два цела броја од 1 до 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
  if (x < 1 || x > 10 || y < 1 || y > 10)
  {
    // ???
  }

  int_fast16_t p = x * x * x * x;
  int_fast16_t q = y * y * y * y;
  return p + q;
}
```



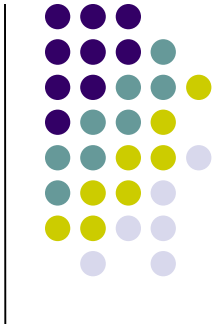
```
// Функција прима два цела броја од 1 до 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
  if (x >= 1 && x <= 10 && y >= 1 && y <= 10)
  {
    int_fast16_t p = x * x * x * x;
    int_fast16_t q = y * y * y * y;
    return p + q;
  }
  else
  {
    // ????
  }
}
```



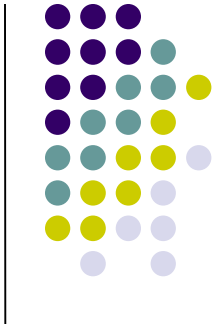
```
// Функција прима два цела броја од 1 до 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
  if (x >= 1 && x <= 10 && y >= 1 && y <= 10)
  {
    int_fast16_t p = x * x * x * x;
    int_fast16_t q = y * y * y * y;
    return p + q;
  }
  else
  {
    printf("Neka poruka o gresci");
    exit(1);
  }
}
```



```
// Функција прима два цела броја од 1 до 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
  if (!(x >= 1 && x <= 10 && y >= 1 && y <= 10))
  {
    printf("Neka poruka o gresci");
    exit(1);
  }

  int_fast16_t p = x * x * x * x;
  int_fast16_t q = y * y * y * y;
  return p + q;
}
```



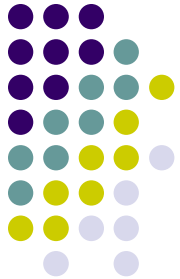
```c
#include <assert.h>

// Функција прима два цела броја од 1 до 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
  assert(x >= 1 && x <= 10 && y >= 1 && y <= 10);

  int_fast16_t p = x * x * x * x;
  int_fast16_t q = y * y * y * y;
  return p + q;
}
```



```c
#include <assert.h>

// Функција прима два цела броја од 1 до 10
int_fast16_t foo(int_fast16_t x, int_fast16_t y)
{
  assert(x >= 1 && x <= 10);
  assert(y >= 1 && y <= 10);

  int_fast16_t p = x * x * x * x;
  int_fast16_t q = y * y * y * y;
  return p + q;
}
```

NIT

Чему то служи



- Асерти су механизам који олакшава хватање грешака које настају услед појаве неодговарајућих ситуација током извршавања
- Рецимо, додавање елемента на стек функционише само ако стек већ није скороз попуњен. Дакле, претпоставка приликом извршавања рутине за додавање елемента је да стек није попуњен.

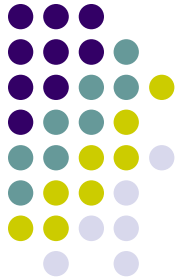
NIT

Асерти наспрам руковања грешкама

- Асертима се хватају случајеви када се деси нешто што ни теоријски не би требало да се деси. То јест, ако се деси то онда значи да неки део програма није добро написан и треба бити поправљен да се тај случај не дешава.
- Руковање грешкама покрива случајеве који се могу десити али знамо да нису подржани од стране програма. На пример, корисник унесе негативан број, а наш програм ради само за позитивне.
- Не треба мешати ове две ствари.

NIT

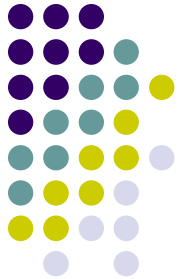
Асерти раде само током развоја



- Пошто хватају ситуације које теоретски не би требале да се десе, а и кад се десе нема смисленог начина реаговања. Провера претпоставки механизмом асерта игра улогу само током развоја и испитивања програма.
- Зато је assert макро-функција имплементирана тако да не ради ништа ако је симбол NDEBUG дефинисан.
- Асерти се на тај начин практично искључују у release верзији програма.

NIT

Статички асерт



- Статички проверљива тврдња, тј. тврдња која се може проверити током превођења.
- _Static_assert
- Отприлике уместо #error са проверама пре тога

- _Static_assert(_Alignof(char) == 1, "alignment of char must be 1");
- Мора да има текст који га прати.