DCC024 Linguagens de Programação

2022.1

Tipos Abstratos de Dados

Haniel Barbosa

UFMG | ICEx
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG

Abstração

- Bárbara Liskov, recebedora do *Turing Award* em 2008, foi pioneira no uso de abstrações em linguagens de programação, nos anos 1970

- Um tipo abstrato de dado é definido por
  - operações
  - dados

The **use** which may be made of an abstraction is **relevant.**
The **implemented** of the abstraction is **irrelevant.**

Abstração

- Diferentes níveis de abstração facilitam *modularização*
  - Separar um problema em diferentes partes e entendê-las individualmente
    - Definições locais
  - Entender o relacionamento entre as diferentes partes
    - Contratos
  - Combinar as diferentes partes para a resolução do problema em questão

Conjuntos em C

```
typedef struct
{ ... } set;

void new(set* s);
void add(set* s, unsigned e);
void del(set* s, unsigned e);
int contains(set* s, unsigned e);
```

Conjuntos em C

```
typedef struct
{ ... } set;

void new(set* s);
void add(set* s, unsigned e);
void del(set* s, unsigned e);
int contains(set* s, unsigned e);

int main()
{
  set s; new(&s); add(&s, 2); add(&s, 3); add(&s, 5);
  printf("Contains %d? %d\n", 1, contains(&s, 1));
  printf("Contains %d? %d\n", 2, contains(&s, 2));
  printf("Contains %d? %d\n", 5, contains(&s, 5));
  del(&s, 5);
  printf("Contains %d? %d\n", 5, contains(&s, 5));
}
```

Implementando conjuntos em C

```
typedef struct
{ unsigned* vector; unsigned size; unsigned capacity; } set;

void new(set* s)
{
  s->vector = (unsigned*)malloc(2 * sizeof(unsigned));
  s->size = 0; s->capacity = 2;
}
void add(set* s, unsigned e)
{
  if (contains(s, e)) return;
  if (s->size == s->capacity) {
    s->capacity *= 2;
    s->vector = realloc(s->vector, s->capacity * sizeof(int));
  }
  s->vector[s->size++] = e;
}
```

Implementando conjuntos em C

```
void del(set* s, unsigned e)
{
  unsigned deleted = s->size;
  for (unsigned i = 0; i < s->size; ++i)
    if (s->vector[i] == e)
    {
      deleted = i; break;
    }
  if (deleted == s->size) return;
  for (unsigned i = deleted; i < s->size - 1; ++i)
    s->vector[i] = s->vector[i+1];
  s->size--;
}

int contains(set* s, unsigned e)
{
  for (unsigned i = 0; i < s->size; ++i)
    if (s->vector[i] == e) return 1;
  return 0;
}
```

Implementando conjuntos em C *com bitsets*

```
typedef struct
{ unsigned* vector; unsigned capacity; } set;

#define INT_BITS 32

void new(set* s)
{
  s->vector = (unsigned*)malloc((1 + (60 / INT_BITS)) * sizeof(unsigned));
  s->capacity = 60;
}

void add(set* s, unsigned e)
{
  unsigned index = e / INT_BITS;
  unsigned offset = e % INT_BITS;
  unsigned bit = 1 << offset;
  s->vector[index] |= bit;
}
```

Implementando conjuntos em C *com bitsets*

```
void del(set* s, unsigned e)
{
  unsigned index = e / INT_BITS;
  unsigned offset = e % INT_BITS;
  unsigned bit = 1 << offset;
  s->vector[index] &= ~bit;
}

int contains(set* s, unsigned e)
{
  unsigned index = e / INT_BITS;
  unsigned offset = e % INT_BITS;
  unsigned bit = 1 << offset;
  return s->vector[index] & bit;
}
```

```
int main()
{
  set s; new(&s);
  add(&s, 2); add(&s, 3); add(&s, 5);
  printf("Contains %d? %d\n", 1,
         contains(&s, 1));
  printf("Contains %d? %d\n", 2,
         contains(&s, 2));
  printf("Contains %d? %d\n", 3,
         contains(&s, 3));
  printf("Contains %d? %d\n", 5,
         contains(&s, 5));
  del(&s, 5);
  printf("Contains %d? %d\n", 5,
         contains(&s, 5));

}
```

Implementando conjuntos em C *com bitsets*

```
int main()
{
  set s; new(&s);
  add(&s, 2); add(&s, 3); add(&s, 5);
  printf("Contains %d? %d\n", 1, contains(&s, 1));
  printf("Contains %d? %d\n", 2, contains(&s, 2));
  printf("Contains %d? %d\n", 3, contains(&s, 3));
  printf("Contains %d? %d\n", 5, contains(&s, 5));
  del(&s, 5);
  printf("Contains %d? %d\n", 5, contains(&s, 5));

  s.vector[0] = 16;
  printf("Contains %d? %d\n", 2, contains(&s, 2));
  printf("Contains %d? %d\n", 3, contains(&s, 3));
  printf("Contains %d? %d\n", 4, contains(&s, 4));
 }
```

Implementando conjuntos em SML com módulos

```
signature SET =
sig
  type set
  val new : set
  val add : set -> int -> set
  val contains : set -> int -> bool
  val remove : set -> int -> set
end ;
```

```
structure S = FunSet ;
val s = S.new ;
val s = S.add s 3 ;
val contains1 = S.contains s 1 ;
val contains3 = S.contains s 3 ;
val s = S.remove s 3 ;
val contains3 = S.contains s 3 ;
```

```
structure FunSet :> SET =
struct
  type set = int -> bool
  val new = fn x => false
  fun add s i = fn (x : int) => if x = i then true else s x
  fun contains s i = s i
  fun remove s i = fn (x : int) => if x = i then false else s x
end;
```

Implementando conjuntos em SML com módulos

Selamento opaco (:>) ou transparante (:)

```
structure FunSet : SET =
struct
  type set = int -> bool
  val new = fn x => false
  fun add s i = fn (x : int) => if x = i then true else s x
  fun contains s i = s i
  fun remove s i = fn (x : int) => if x = i then false else s
end;
```