Concepte fundamentale ale limbajelor de programare

Sisteme de tipuri

Curs 08

conf. dr. ing. Ciprian-Bogdan Chirila

Universitatea Politehnica Timisoara
Departamentul de Calculatoare si Tehnologia Informatiei

24 aprilie 2023

Cuprins

Sistemul de tipuri in C

- tipuri predefinite
- tipul enumerare de constante
- tipuri de date structurate
  - tablou
  - structura
  - uniune
- pointeri
- structuri recursive

Tipuri predefinite

- char - un octet pentru setul local de caractere
- int - mulțimea de numere întregi pe mașina gazdă
  - short int, de obicei, pe 16 biți
  - long int pe cel puțin 32 de biți
- lungimea (short) 16 biți
- lungimea (short) <= lungimea (int) <= lungimea (long)
- cu semn și fara semn pot fi aplicate la char sau int
- unsigned char 0..255
- signed char -128..+127
- float, double
- <limits.h> <float.h>

Enumerare de constante

- enum boolean {NO,YES};
- enum days {MO=1,TU,WE,THU,FRI,SAT,SUN};

Tablouri

- Forma generala
  - tip_element nume_tablou[expresie_constanta]
  - dimensiunea tabloului > 0
- Exemplu
  - v[10] – 10 tablou de intregi
  - indecsii incep la zero
  - primul element v[0]
  - ultimul element v[9]
- Initializarea
  - x[]={1,2,3};
- dimensiunea tabloului trebuie cunoscuta la compilare
  - tablourile in C sunt tablouri statice

Tablourile multidimensionale

- este un tablou de tablouri
- int mat[10][10]
  - matrice cu 10 linii si 10 coloane
  - elementul de la indicii (i,j) se pot accesa cu mat[i][j] si nu cu mat[i,j] ca in alte limbaje de programare
- parametrii formali de tip tablou se pot declara incomplet fara a specifica prima dimensiune
- int f(char l[],int m[][10]);

Tablourile multidimensionale

- dimensiunea efectiva a tablourilor poate fi specificata in momentul apelarii functiei
- functiile pot avea un grad mai mare de generalitate decat in Pascal unde
  - dimensiunea parametrului forma si actual trebuie sa fie egale

Structuri

- implementeaza in C produsul cartezian

```
struct point
{
 int x;
 int y;
};
```

- poate fi copiat prin atribuire

```
struct point origin={0,0};
```

Structuri

- accesul la campuri se face prin operatorul de calificare

```
struct point p;
p.x or p.y
```

- pot fi returnate de functii

```
struct point f(int x, int y) { }
```

- pot fi incuibate

```
struct rectangle
{
 struct point p1;
 struct point p2;
};
```

- accesul poate fi si el aplicat succesiv

```
struct rectangle r;
r.p1.x
```

Uniuni

- implementeaza reuniunile variabile

```
union
{
 int i;
 float f;
 char c;
} u;
```

- u poate fi un `int` sau un `float` sau un `char`

Uniuni

- selectia se face prin operatorul de calificare
  - `u.i, u.f, u.c`
- pot fi incuibate in alte uniuni, structuri sau tablouri
- reprezentarea lor in memorie
  - toate campurile au deplasament zero fara de adresa de start a uniunii
  - la un moment dat doar o reprezentare este disponibila

Uniunile

- nu se face verificare de tipuri
- toata responsabilitatea este lasate pe umerii programatorilor
- selectarea unei variante gresite poate cauza erori de rulare severe
- pot fi initializate cu o valoare de tipul primei variante (intreg pentru u)

Pointeri

- o declaratie a unui pointer trebuie sa refere un tip
  - int x=1, y;
  - int *p; /* p este un pointer la intreg */
  - void *p1; /* poate stoca orice tip de pointer */
- poate stoca adresele obiectelor
  - p=&x;
- pentru a accesa obiectul referit de pointer
  - se numeste dereferentiere
  - y=*p; /* y va lua valoarea 1*/
  - *p=0; /* x va lua valoarea 0 */
- variabile sinonime se pot crea cu consecintele cunoscute

Pointeri

- permite accesul direct la locatia de memorie a unui argument

```
void exchange1(int x, int y) /* gresit */
{
 int aux;
 aux=x; x=y; y=aux;
}
exchange1(a,b);
/* interschimba copii ale lui a si b */
```

Pointeri

```
void exchange2(int *x, int *y)
{
 int aux;
 aux=*x; *x=*y; *y=aux;
}
exchange2(&a,&b); /* apel corect */
/* interschimba valorile lui a si b */
```

Pointeri

- se pot utiliza impreuna cu tablourile

```
int a[10];
int *pa;}
pa=&a[0];
/*pa va retine adresa lui a[0]*/
```

- adresa unui tablou este adresa primului element din acel tablou
- a si pa vor avea aceleasi valori

Pointeri

- *(pa+i) este de fapt elementul a[i]
- *(pa+i) este echivalent cu a[i]
- (pa+i) este echivalent cu &a[i]
- cand un tablou este transmis unei functii
  - este transmisa doar adresa primului element
  - parametrul formal este un pointer
  - se comporta ca o variabila care contine o adresa
- int f(char s[]) { ... }
- int f(char *s) { ... }
- cele doua forme sunt echivalente

Aritmetica pointerilor

- operatiile permise sunt:
  - atribuirea pointerilor de acelasi tip
  - adunarea sau scaderea unui pointer cu un intreg
  - scaderea sau compararea a doi pointeri care refere elemente ale aceluiasi tablou
  - atribuirea sau compararea cu NULL sau 0

Aritmetica pointerilor

- operatii ilegale
  - adunarea a doi pointeri
  - inmultirea sau impartirea pointerilor
  - operatii pe biti sau utilizarea de masti
  - adunarea de pointeri si valori reale

Pointeri la functii

- permise in C
- pot fi atribuiti
- pot fi pusi in tablouri
- pot fi trimisi ca parametri la functii
- pot fi returnate ca valori de functii

Alocarea si realocarea dinamica a memoriei

- alocarea dinamica a obiectelor anonime de dimensiune specificata
  - `malloc(...);`
  - `calloc(...);`
  - `realloc(...);`
- eliberarea memoriei alocate
  - `free()`
- memoria eliberata poate crea referinte false

Structuri recursive

- bazate pe pointeri
- permit descrierea de liste sau arbori

```
struct node
{
 type info;
 struct node *left;
 struct node *right;
}
```

- o structura recursiva trebuie sa utilizeze pointeri
- un tip nu poate sa isi contina propria instantiere

Echivalente de tipuri

- bazat pe echivalente structurale
- exceptii
  - struct
  - union
- sunt tipuri diferite chiar daca au aceeasi structura
- conversiile de tipuri sunt permise prin operatorul de cast
- `(tip) expresie`

Cuprins

Sistemul de tipuri in Python

- tipul text: str
- tipuri numerice: int, float, complex
- tipuri secventa: list, tuple, range
- tipuri mapate: dict
- tipuri multime: set, frozenset
- tipul boolean: bool
- tipuri binare: bytes, bytearray, memoryview
- tipul vid: NoneType

Tipuri predefinite

```
x = 5
print(type(x))
<class 'int'>
---
x = "Hello World" <class 'str'>
x = 20 <class 'int'>
x = 20.5 <class 'float'>
x = 1j <class 'complex'>
x = ["apple", "banana", "cherry"] <class 'list'>
x = ("apple", "banana", "cherry") <class 'tuple'>
x = range(6) <class 'range'>
```

Tipuri predefinite

```
x = {"name" : "John", "age" : 36} <class 'dict'>
x = {"apple", "banana", "cherry"} <class 'set'>
x = frozenset({"apple", "banana", "cherry"})
<class 'frozenset'>
x = True <class 'bool'>
x = b"Hello" <class 'bytes'>
x = bytearray(5) <class 'bytearray'>
x = memoryview(bytes(5)) <class 'memoryview'>
x = None <class 'NoneType'>
```

Conversii

```
# integers
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
# floats
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
```

Conversii

```
# strings
x = str("fcpl") # x will be 'fcpl'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

Siruri de caractere

```
print("Hello FCPL")
print('Hello FCPL')
```

Siruri de caractere pe linii multiple

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""

a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
```

Felierea sirurilor de caractere

```
b = "Hello, World!"
print(b[2:5])
#llo
print(b[:5])
#Hello
print(b[2:])
#llo, World!
print(b[-5:-2])
#orl
```

Modificarea sirurilor de caractere

```
s=" Hello FCPL "
s.upper()
s.lower()
s.strip()
s.replace("H", "J")
s.split(",")
a="Alfa"
b="Romeo"
c=a+" "+b
```

Booleeni

```
print(10 > 9)
print(10 == 9)
print(10 < 9)

bool("abc")
bool(123)
bool(["apple", "cherry", "banana"])
# vor returna True
```

Booleeni

```
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
# vor returna False
```

Liste

```
mylist = ["alfa", "beta", "gamma"]
print(len(thislist))
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
list4 = ["abc", 34, True, 40, "male"]
nextlist = list(("alfa", "beta", "gamma"))
print(nextlist[1]) # beta
print(nextlist[-1]) # gamma
print(nextlist[1:2]) # ["beta", "gamma"]
```

Accesarea listelor

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)

islist = ["apple", "banana", "cherry", "orange", "kiwi", "mang
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist)
```

Adaugarea de elemente la liste

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

Stergerea de elemente din liste

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)

thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

Iterarea listelor

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

Tupluri

```
mytuple = ("apple", "banana", "cherry")
print(mytuple[1])
print(mytuple[-1])
print(mytuple[1:2])
```

Actualizarea tuplurilor

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

Despachetarea tuplurilor

```
fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

Parcurgerea tuplurilor

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

Multimi

```
myset = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
set4 = {"abc", 34, True, 40, "male"}
```

Operatii cu multimi

- add clear copy
- difference
- discard intersection
- isdisjoint issubset issupperset
- pop remove
- union update

Dictionar

```
thisdict =
{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
thisdict["color"] = "red"
thisdict.update({"color": "blue"})
```

Iterarea dictionarelor

```
for x in thisdict:
  print(thisdict[x])

for x in thisdict.values():
  print(x)

for x in thisdict.keys():
  print(x)

for x, y in thisdict.items():
  print(x, y)
```

Cuprins

Sistemul de tipuri in Lisp

- contine tipuri de date
- nu exista variabile in sensul clasic
- variabilele sunt inlocuite cu atomi simbolici sau simboluri
- simbolurile au un nume care este un sir de litere si nu reprezinta un numar
- limbajul Lisp este proiectat pentru calcul simbolic

Sistemul de tipuri in Lisp

- in limbajele imperative
  - la o variabila noi atribuim o valoare de un anumit tip
  - referirea valorii se face prin acel nume de variabila
- in Lisp
  - un simbol este un nume atasat la o entitate pentru o aumita perioada de timp
  - tipurile de date nu se refera la simboluri ci la valorile legate de acestea
  - un simbol poate reprezenta la diferite momente de timp valori diferite apartinand diferitelor tipuri

Sistemul de tipuri in Lisp

- din punct de vedere al implementarii
  - este posibila legarea dinamica a diferitelor tipuri la o aceeasi variabila
  - variabilele Lisp sunt referinte (pointeri) la entitati ce pot fi de diverse tipuri
- in limbajele imperative
  - variabila este un nume dat unei locatii de memorie
  - de dimensiune fixa
  - egala cu cea a tipului variabilei

Legarea unei valori la un atom

- inlocuieste operatia de atribuire
- este implementata de forma functionala `setq` si `setf`

```
> (setq x 10)
10

> (setq x ‘Lisp)
LISP

> (setq x ‘(a b c))
(A B C)
```

Legarea unei valori la un atom

- tipul este specific obiectului reprezentat de simbol
- dar nu simbolului in sine
- este cazul limbajelor slab tipizate
- la compilare este imposibil de spus care este tipul variabilei
- facilitatile de procesare dinamica sunt preferate in schimbul verificarii corespondentei de tipuri de la compilare

Tipuri simple predefinite

- Numerice
  - Integer
    - Fixnum
    - Bignum
  - Ratio
    - 10/3
    - 10/2
    - 10/4
    - (* 5/2 5/3)
    - 25/6

Tipuri simple predefinite

- Numerice (continuate)
  - float
    - short-float
    - single-float
    - double-float
    - long-float
  - complex
    a+bi -> #c(a b)
    > (sqrt -1)
    #c(0 1)
    > (* #c(01) #c(0 2))
    -2
- Non-numerice
  - character

Liste

- listele sunt expresii compuse non-atomice
- (red yellow blue)
- (1 2 -4 1.5)
- ((red yellow blue) (1 2 -4 1.5))
- organizarea este liniara, secventiala
- sunt implementate ca si structuri de date dinamice
- in limbajele imperative
  - alocarea dinamica si de-alocarea listelor
  - se face manual de catre programator
- in Lisp alocarea si de-alocarea se face automat

Liste

- adaugarea unui element intr-o lista folosind `cons`
  - (cons 'd '(a b c))
  - (d a b c)
- alocarea dinamica pentru atomul "d"
- legarea lui "d" in lista
- sunt operatiuni invizibile pentru programator
- au doua campuri
  - car – pointeaza spre primul element al listei
  - cdr – pointeaza spre restul elementelor listei (de la al doilea element incepand)

Vectori si matrici

```
> (setq mat (make-array
‘(2 3 2):initial-contents
‘(((1 2)(3 4)(5 6)) ((7 8)(9 10)(11 12)))))

#3A(((1 2)(3 4)(5 6)((7 8)(9 10)(11 12))))
```

Vectori si matrici

```
> (setq vect (vector 0 1 2 3 4 5 6 7 8 9))
#(0 1 2 3 4 5 6 7 8)

> (aref mat 0 0 0)
1

> (aref mat 1 2 0)
11
```

Vectori de biti si matrici de biti

```
> (setq matbits (make-array '(2 3 2)
initial-element 0:element-type 'bit))
#3A ((#*00 #*00 #*00) (#*00 #*00 #*00))

> (setq (aref matbits 1 2 0) 1))
1
```

Vectori de biti si matrici de biti

```
> (setq vbits #*01010101)
#* 01010101

> (bit-not vbits)
#* 10101010
```

- bit-not
- bit-and
- bit-ior, bit-xor
- bit-eqv - equivalence
- bit-orcl - implication

Siruri de caractere

- Subtype of vectors

```
>(length "abcd")
4
>(aref "abcd" 2)
#\c
```

- compararea de siruri de caractere

```
> (string = "abcd" "abcd")
T
> (string < "abcd" "abdd")
2
```

Siruri de caractere

- transformarea unui atom in string

```
> (string 'abcd)
"ABCD"
```

- cautarea unui subsir intr-un sir

```
> (search "cd" "abcd")
2
```

Echivalenta de tipuri. Subtipuri

- programatorul Lisp nu trebuie sa fie constient de tipurile de date
- in versiunile mai vechi tipurile nici nu existau
- legarea dinamica a tipurilor evita verificarea statica
- singura verificare se face cand un operator isi executa operanzii >(+ 1 "5")

Subtipuri

- Tipuri numerice
  - rationale
    - intregi:fixnum,bignum
    - fractii
  - float
    - short-float
    - single-float
    - double-float
    - long-float
  - complex

Subtipuri

- tipuri vectoriale
  - vector
    - string
    - bit-vector
- operatori
  - `type-of 1 arg`
  - `type-p 2 args`
  - `subtype-p 2 args`

Exemple cu tipuri

```
> (type-of 1)
FIXNUM

> (type-of #*01000111)
(SIMPLE-BIT-VECTOR 8)

> (type-of #\a)
CHARACTER

> (type-of "abcd")
SIMPLE-STRING
```

Exemple cu tipuri

```
> (typep 1 'number)
T
> (typep 1 'integer)
T
> (typep 1 'fixnum)
T
> (typep 1 'bignum)
NIL
```

Exemple cu tipuri

```
> (typep (a b c) ‘sequence)
T
> (typep (a b c) ‘list)
T
> (subtypep ‘integer ‘number)
T
> (subtypep ‘array ‘sequence)
NIL
```

Bibliography

1. Brian Kernighan, Dennis Ritchie, C Programming Language, second edition, Prentice Hall, 1978.
2. Carlo Ghezzi, Mehdi Jarayeri – Programming Languages, John Wiley, 1987.
3. Horia Ciocarlie – Universul limbajelor de programare, editia 2-a, editura Orizonturi Universitare, Timisoara, 2013.