Cartilla de código piola

UTN FRSF - El Rejunte

2016

Índice

1. C/C++

1.1. Typedefs

```
typedef long long ll;
typedef long double ld;
typedef pair<int,int> pii;
typedef vector<pii> vpii;
typedef vector<int> vi;
typedef vector<vi> vvi;
```

1.2. Macros

```
//buffers
#define MEMZ(mem) memset(mem, 0, sizeof mem) // memset => <cstring / string.h>
#define MEMX(mem,x) memset(mem, (x), sizeof mem)

//loops
#define forall(i,a,b) for(int (i)=(a);(i)<(b);++(i))
#define foreach(i,v) for(typeof((v).begin()) (i) = (v).begin(); (i) != (v).end(); ++(i)
    )

//map/pairs
#define mp make_pair
#define fi first
#define se second

//vector
#define pb push_back

//queries
#define in(a,v) ((v).find((a))!=(v).end()) // <algorithm>
```

1.3. I/O

1.3.1. scanf Format Strings

`%[*][width][length]specifier`

spec	Tipo	Descripción
`i`	`int`	Dígitos dec. `[0-9]`, oct. `(0)[0-7]`, hexa `(0x|0X)[0-9a-fA-F]`. Con signo.
`d, u`	`int, unsigned`	Dígitos dec. `[+-0-9]`.
`o`	`unsigned`	Dígitos oct. `[+-0-7]`.
`x`	`unsigned`	Dígitos hex. `[+-0-9a-fA-F]`. Prefijo `0x,0X` opcional.
`f, e, g`	`float`	Dígitos dec. c/punto flotante `[+-.0-9]`. Prefijo `0x,0X` y sufijo `e,E` opcionales.

Continuación

spec	Tipo	Descripción
c, [width]c	char, char*	Siguiente carácter. Lee width chars y los almacena contiguamente. No agrega \0.
s	char*	Secuencia de chars hasta primer espacio. Agrega \0.
p	void*	Secuencia de chars que representa un puntero.
[chars]	Scanset, char*	Caracteres especificados entre corchetes.] debe ser primero en la lista, - primero o último. Agrega \0
[^chars]	!Scanset, char*	Caracteres no especificados entre corchetes.
n	int	No consume entrada. Almacena el número de chars leídos hasta el momento.
%		%% consume un %

sub-specifier	Descripción
*	Indica que se leerá el dato pero se ignorará. No necesita argumento.
width	Cantidad máxima de caracteres a leer.
lenght	Uno de hh, h, l, ll, j, z, t, L. Ver tabla siguiente.

length	d i	u o x
(none)	int*	unsigned int*
hh	signed char*	unsigned char*
h	short int*	unsigned short int*
l	long int*	unsigned long int*
ll	long long int*	unsigned long long int*
j	intmax_t*	uintmax_t*
z	size_t*	size_t*
t	ptrdiff_t*	ptrdiff_t*
L		

length	f e g a	c s [ ] [^]	p	n
(none)	float*	char*	void**	int*
hh				signed char*
h				short int*
l	double*	wchar_t*		long int*
ll				long long int*
j				intmax_t*
z				size_t*
t				ptrdiff_t*
L	long double*			

1.3.2. printf Format Strings

```
%[flags][width][.precision][length]specifier
```

specifier	Descripción	Ejemplo
d or i	Entero decimal con signo	392
u	Entero decimal sin signo	7235
o	Entero octal sin signo	610

Continuación

specifier	Descripción	Ejemplo
x	Entero hexadecimal sin signo	7fa
X	Entero hexadecimal sin signo (mayúsculas)	7FA
f	Decimal punto flotante (minúsculas)	392.65
F	Decimal punto flotante (mayúsculas)	392.65
e	Notación científica (mantisa/exponente), (minúsculas)	3.9265e+2
E	Notación científica (mantisa/exponente), (mayúsculas)	3.9265E+2
g	Utilizar la representaciíon más corta: %e ó %f	392.65
G	Utilizar la representaciíon más corta: %E ó %F	392.65
a	Hexadecimal punto flotante (minúsculas)	-0xc.90fep-2
A	Hexadecimal punto flotante (mayúsculas)	-0XC.90FEP-2
c	Caracter	a
s	String de caracteres	sample
p	Dirección de puntero	b8000000
n	No imprime nada. El argumento debe ser int*, almacena el número de caracteres imprimidos hasta el momento.	
%	Un % seguido de otro % imprime un solo %	%

flag	Descripción
-	Justificación a la izquierda dentro del campo width (ver width sub-specifier).
+	Forza a preceder el resultado de texttt+ o texttt-.
(espacio)	Si no se va a escribir un signo, se inserta un espacio antes del valor.
#	Usado con o, x, X specifiers el valor es precedido por 0, 0x, 0X respectivamente para valores distintos de 0.
0	Rellena el número con texttt0 a la izquierda en lugar de espacios cuando se especifica width.

width	Descripción
(número)	Número mínimo de caracteres a imprimir. Si el valor es menor que número, el resultado es rellando con espacios. Si el valor es mayor, no es truncado.
*	No se especifica width, pero se agrega un argumento entero precediendo al argumento a ser formateado. Ej. printf("---%*d----\n", 3, 2); ⇒ "---- 5----".

precision	Descripción
.(número)	Para d, i, o, u, x, X: número mínimo de dígitos a imprimir. Si el valor es más chico que número se rellena con 0. Para a, A, e, E, f, F: número de dígitos a imprimir después de la coma (default 6). Para g, G: Número máximo de cifras significativas a imprimir. Para s: Número máximo de caracteres a imprimir. Trunca.

Continuación	
precision	**Descripción**
.*	No se especifica `precision` pero se agrega un argumento entero precediendo al argumento a ser formateado.

length	**d i**	**u o x X**
(none)	int	unsigned int
hh	signed char	unsigned char
h	short int	unsigned short int
l	long int	unsigned long int
ll	long long int	unsigned long long int
j	intmax_t	uintmax_t
z	size_t	size_t
t	ptrdiff_t	ptrdiff_t
L		

length	**f F e E g G a A**	**c**	**s**	**p**	**n**
(none)	double	int	char*	void*	int*
hh					signed char*
h					short int*
l		wint_t	wchar_t*		long int*
ll					long long int*
j					intmax_t*
z					size_t*
t					ptrdiff_t*
L	long double				

1.3.3. Fast C++ Input

```
ios_base::sync_with_stdio(false); cin.tie(NULL);
```

2. Constantes y Tablas

2.1. Constantes

```
#define INF 1000000000 // 1 billon, entra en int
#define EPS 1e-12
#define PI 3.14159265358979323846
```

3. Estructuras de datos

3.1. Disjoint Sets

3.1.1. Union Find (OOP)

Utiliza: <vector>

Notas: Rangos [i,j] (*0 based*). No recomendable si se tienen que crear y destruir muchos objetos. Probar funcionamiento en casos límites.

```
class UnionFind
{
private:
  vector<int> p, rank, setSize;
  int numSets;
public:
  UnionFind(int N)
  {
    setSize.assign(N, 1);
    numSets = N;
    rank.assign(N, 0);
    p.assign(N, 0);
    for (int i = 0; i < N; i++) p[i] = i;
  }
  int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
  bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
  void unionSet(int i, int j)
  {
    int x = findSet(i), y = findSet(j);
    if (!(x==y))
    {
      numSets--;
      if (rank[x] > rank[y])
      {
        p[y] = x;
        setSize[x] += setSize[y];
      }
      else
      {
        p[x] = y;
        setSize[y] += setSize[x];
        if (rank[x] == rank[y]) rank[y]++;
      }
    }
  }
  int numDisjointSets() { return numSets; }
  int sizeOfSet(int i) { return setSize[findSet(i)]; }
};
```

3.1.2. Union Find (C style/static)

Utiliza: <cstring>

Notas: Rangos [i,j] (*0 based*). En `init(n)`, n $\leq$ MAXN

```
#define MAXN ...
int p[MAXN], rank[MAXN], setSize[MAXN], numSets;

inline void init(int n)
{
    memset(rank,0, sizeof(int)*n);
    for(int i=0;i<n;++i) p[i]=i, setSize[i]=1;
    numSets = n;
}
inline int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
inline bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
void unionSet(int i, int j)
{
```

```
14      int x = findSet(i), y = findSet(j);
15      if (!(x==y))
16      {
17          numSets--;
18          if (rank[x] > rank[y])
19          {
20              p[y] = x;
21              setSize[x] += setSize[y];
22          }
23          else
24          {
25              p[x] = y;
26              setSize[y] += setSize[x];
27              if (rank[x] == rank[y]) rank[y]++;
28          }
29      }
30  }
31  inline int numDisjointSets() { return numSets; }
32  inline int sizeOfSet(int i) { return setSize[findSet(i)]; }
```