

Tipos de Datos Abstractos

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

El siguiente desarrollo, detalla las partes pertinentes de la estrategia analizada sobre el TDA de Pila Dinámica Circular.

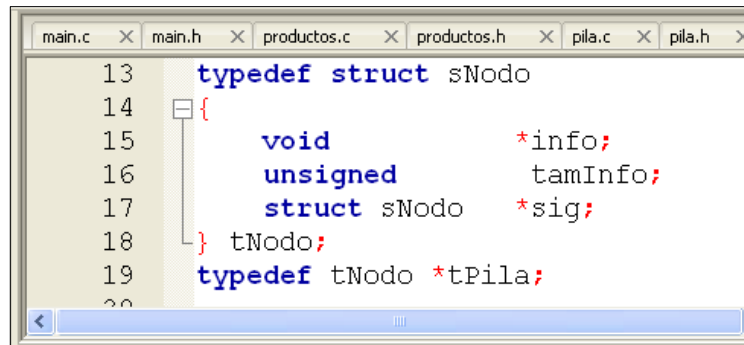
Se tratarán únicamente las primitivas de poner y sacar de Pila Dinámica Circular.

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

`tPila`

`info tam sig`
`tNodo`

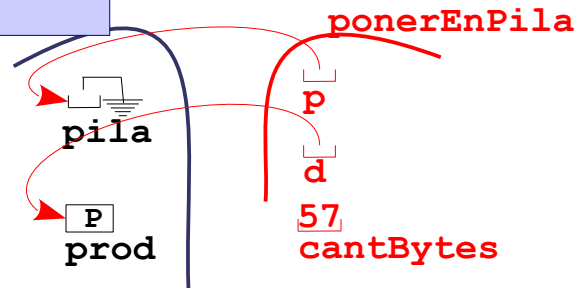


```
main.c x main.h x productos.c x productos.h x pila.c x pila.h x
13 typedef struct sNodo
14 {
15     void *info;
16     unsigned tamInfo;
17     struct sNodo *sig;
18 } tNodo;
19 typedef tNodo *tPila;
20
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner 1er elemento

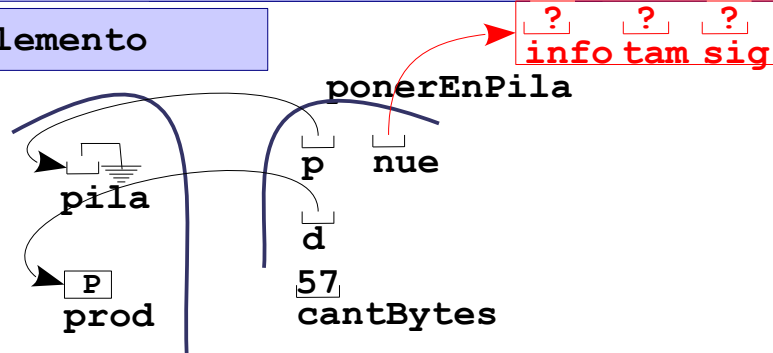


```
main.c x productos/productos.c x lineasDeTexto/lineasTexto.h x pilaCirc/pila.c x pilaCirc/pila.h x
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNodo *nue;
26
27     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner 1er elemento

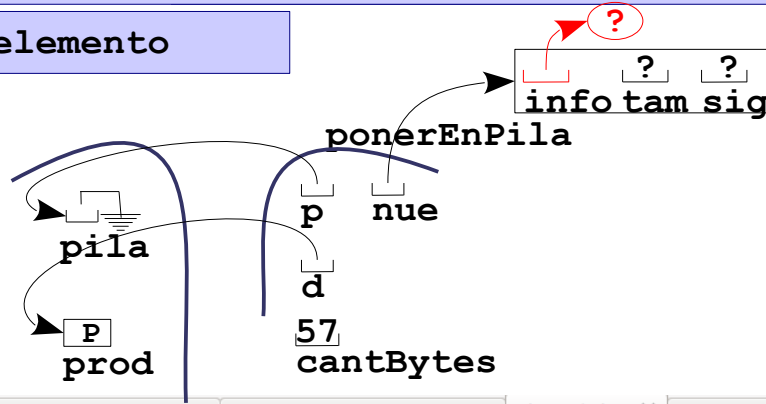


```
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNode *nue;
26
27     if((nue = (tNode *)malloc(sizeof(tNode))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner 1er elemento

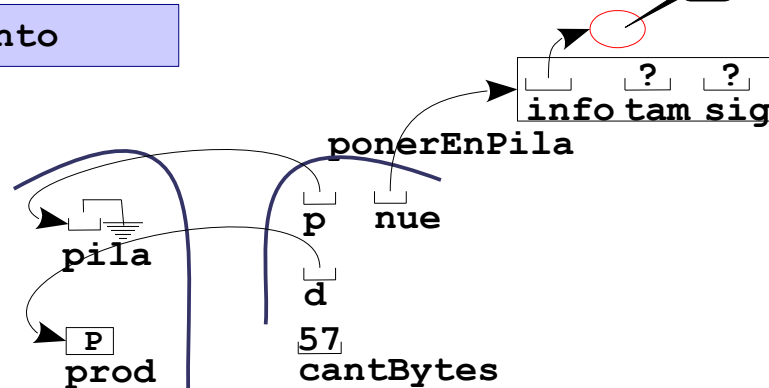


```
productos/productos.c | lineasDeTexto/lineasTexto.h | pilaCirc/pila.c | pilaCirc/pila.h
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNodo *nue;
26
27     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner 1er elemento

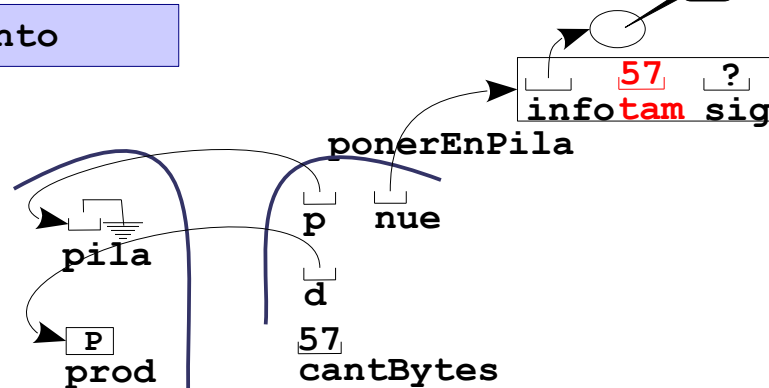


```
main.c
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNode *nue;
26
27     if((nue = (tNode *)malloc(sizeof(tNode))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner 1er elemento

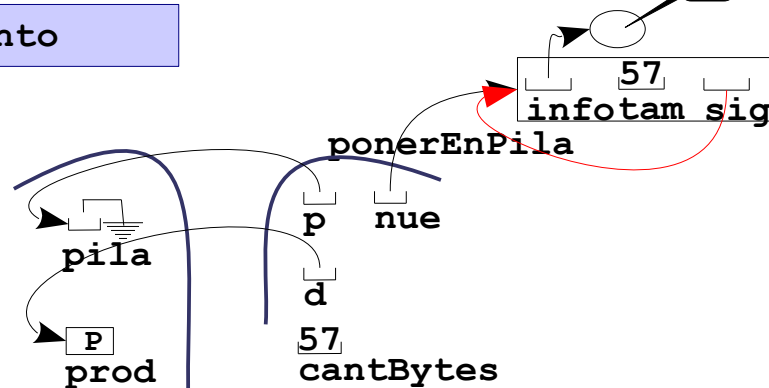


```
productos/productos.c x lineasDeTexto/lineasTexto.h x pilaCirc/pila.c x pilaCirc/pila.h x
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNode *nue;
26
27     if((nue = (tNode *)malloc(sizeof(tNode))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```


Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner 1er elemento

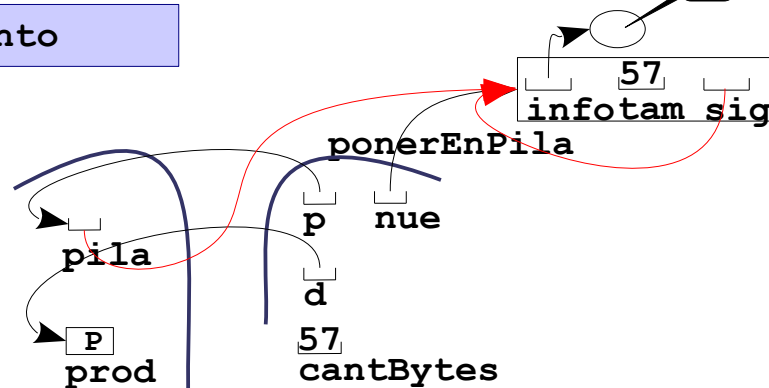


```
productos/productos.c x lineasDeTexto/lineasTexto.h x pilaCirc/pila.c x pilaCirc/pila.h x
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNode *nue;
26
27     if((nue = (tNode *)malloc(sizeof(tNode))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner 1er elemento

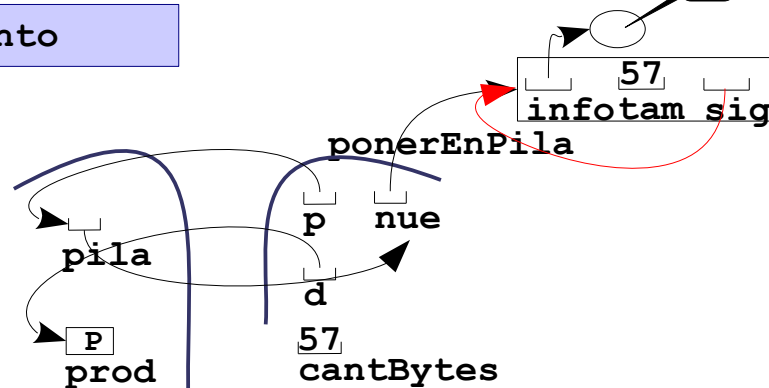


```
productos/productos.c x lineasDeTexto/lineasTexto.h x pilaCirc/pila.c x pilaCirc/pila.h x
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNode *nue;
26
27     if((nue = (tNode *)malloc(sizeof(tNode))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner 1er elemento

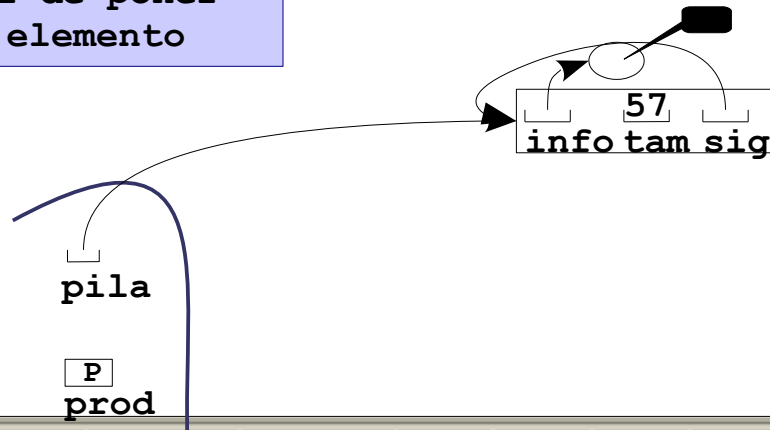


```
productos/productos.c x lineasDeTexto/lineasTexto.h x pilaCirc/pila.c x pilaCirc/pila.h x
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNode *nue;
26
27     if((nue = (tNode *)malloc(sizeof(tNode))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Resultado final de poner
en pila 1er elemento

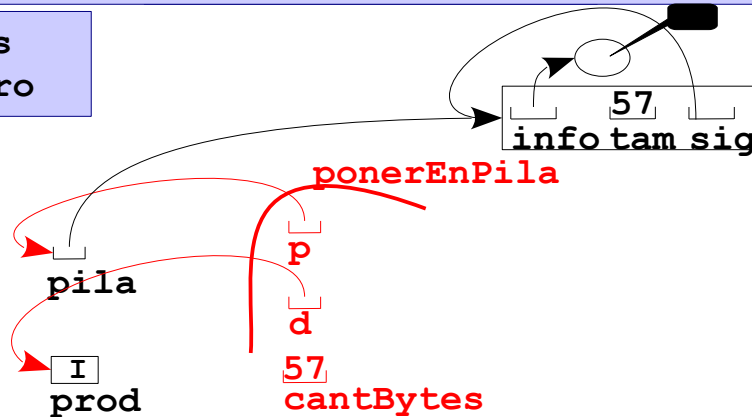


```
main.c x main.h x productos.c x productos.h x pila.c x pila.h x pila.c x pila.h x
86 void probarPonerYSacarDePila(void)
87 {
88     tProd prod;
89     tPila pila;
90
91     crearPila(&pila);
92     while(!pilaLlena(&pila, sizeof(prod)) &&
93         ingresarProducto(&prod))
94         if(!ponerEnPila(&pila, &prod, sizeof(prod)))
95             puts("ERROR - inesperado: pila llena\n");
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner elementos
sucesivos al 1ro

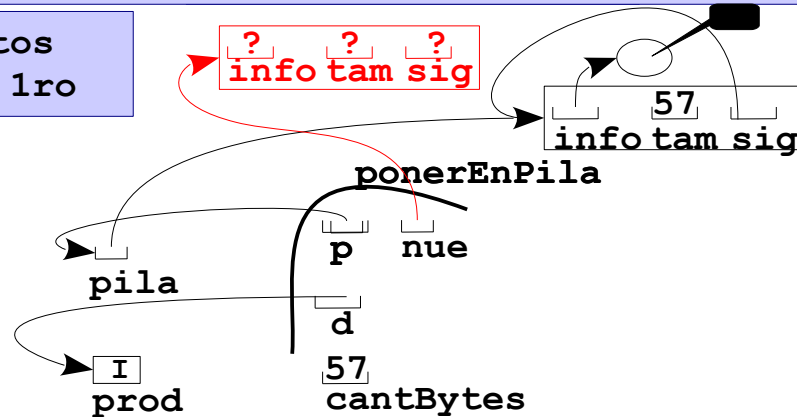


```
main.c  productos/productos.c  lineasDeTexto/lineasTexto.h  pilaCirc/pila.c  pilaCirc/pila.h
23  int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24
25      tNodo *nue;
26
27      if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
28         (nue->info = malloc(cantBytes)) == NULL)
29      {
30          free(nue);
31          return 0;
32      }
33      memcpy(nue->info, d, cantBytes);
34      nue->tamInfo = cantBytes;
35      if(*p == NULL)
36      {
37          nue->sig = nue;
38          *p = nue;
39      }
40      else
41      {
42          nue->sig = (*p)->sig;
43          (*p)->sig = nue;
44      }
45      return 1;
46
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner elementos
sucesivos al 1ro

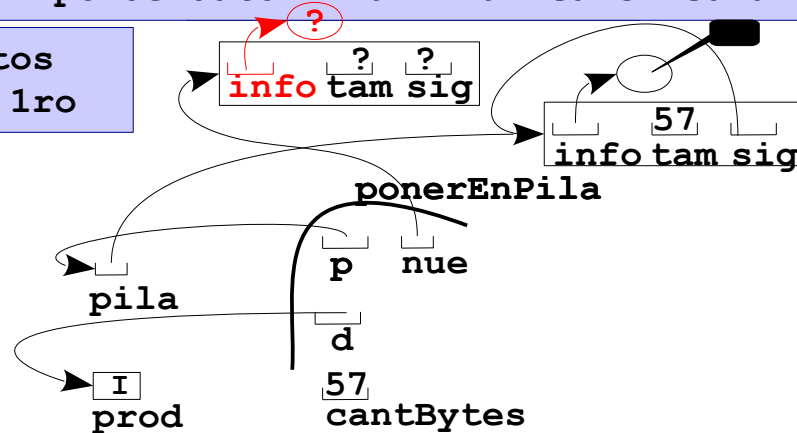


```
main.c x productos/productos.c x lineasDeTexto/lineasTexto.h x pilaCirc/pila.c x pilaCirc/pila.h x
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNodo *nue;
26
27     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner elementos
sucesivos al 1ro

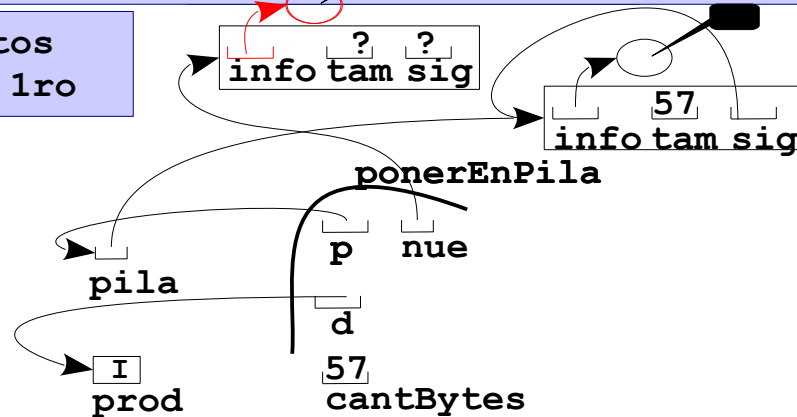


```
productos/productos.c x lineasDeTexto/lineasTexto.h x pilaCirc/pila.c x pilaCirc/pila.h x
main.c x
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNodo *nue;
26
27     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner elementos
sucesivos al 1ro

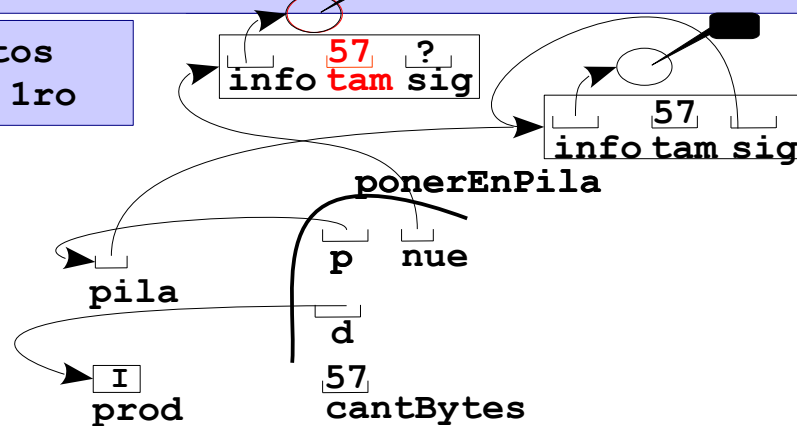


```
products/productos.c x lineasDeTexto/lineasTexto.h x pilaCirc/pila.c x pilaCirc/pila.h x
main.c x
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNodo *nue;
26
27     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```


Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner elementos
sucesivos al 1ro

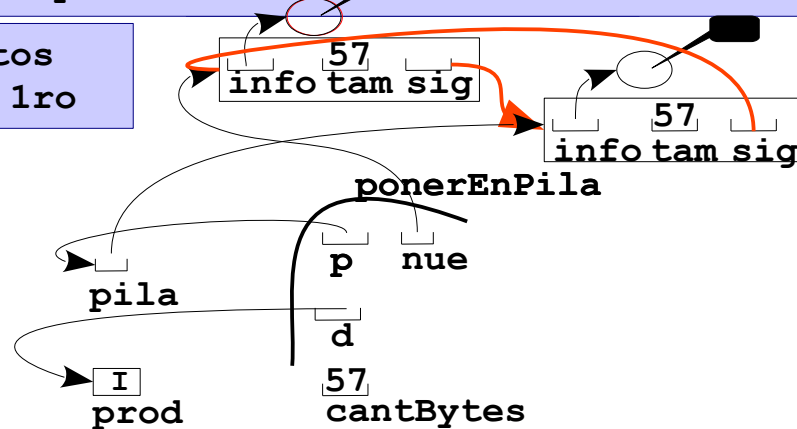


```
products/productos.c x linesDeTexto/lineasTexto.h x pilaCirc/pila.c x pilaCirc/pila.h x
main.c x
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNodo *nue;
26
27     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner elementos
sucesivos al 1ro

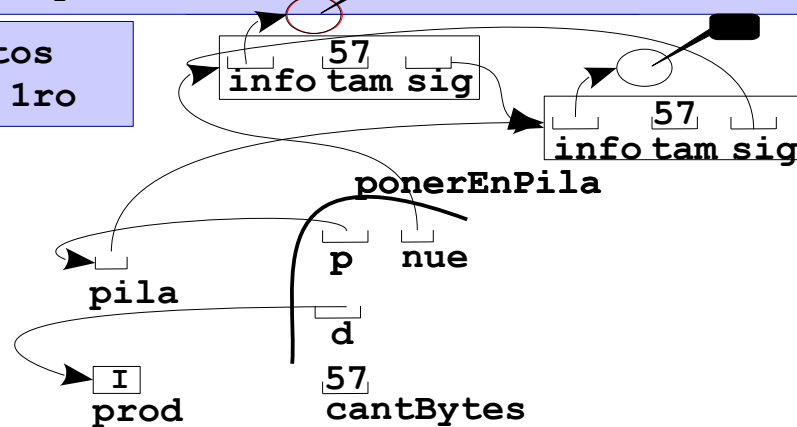


```
products/productos.c x linesDeTexto/lineasTexto.h x pilaCirc/pila.c x pilaCirc/pila.h x
main.c x
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNodo *nue;
26
27     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Poner elementos
sucesivos al 1ro

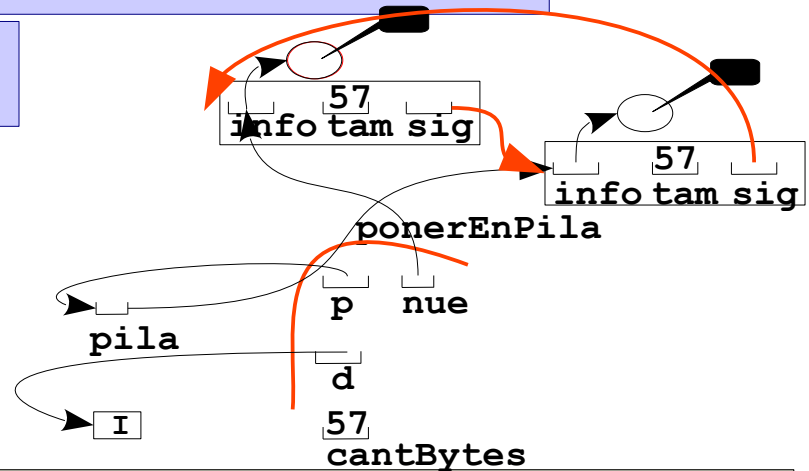


```
products/productos.c x lineasTexto/lineasTexto.h x pilaCirc/pila.c x pilaCirc/pila.h x
main.c x
23 int ponerEnPila(tPila *p, const void *d, unsigned cantBytes)
24 {
25     tNodo *nue;
26
27     if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
28        (nue->info = malloc(cantBytes)) == NULL)
29     {
30         free(nue);
31         return 0;
32     }
33     memcpy(nue->info, d, cantBytes);
34     nue->tamInfo = cantBytes;
35     if(*p == NULL)
36     {
37         nue->sig = nue;
38         *p = nue;
39     }
40     else
41     {
42         nue->sig = (*p)->sig;
43         (*p)->sig = nue;
44     }
45     return 1;
46 }
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Resultado final de poner
un 2do elemento en pila

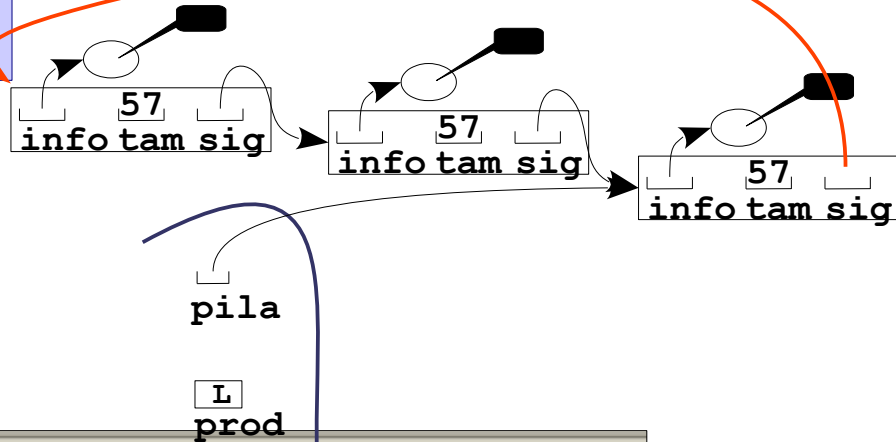


```
main.c x main.h x productos.c x productos.h x pila.c x pila.h x pila.c x pila.h x
86 void probarPonerYSacarDePila(void)
87 {
88     tProd  prod;
89     tPila  pila;
90
91     crearPila(&pila);
92     while(!pilaLlena(&pila, sizeof(prod)) &&
93           ingresarProducto(&prod))
94         if(!ponerEnPila(&pila, &prod, sizeof(prod)))
95             puts("ERROR - inesperado: pila llena\n");
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular

Sacar un elemento de pila
Distinto al último

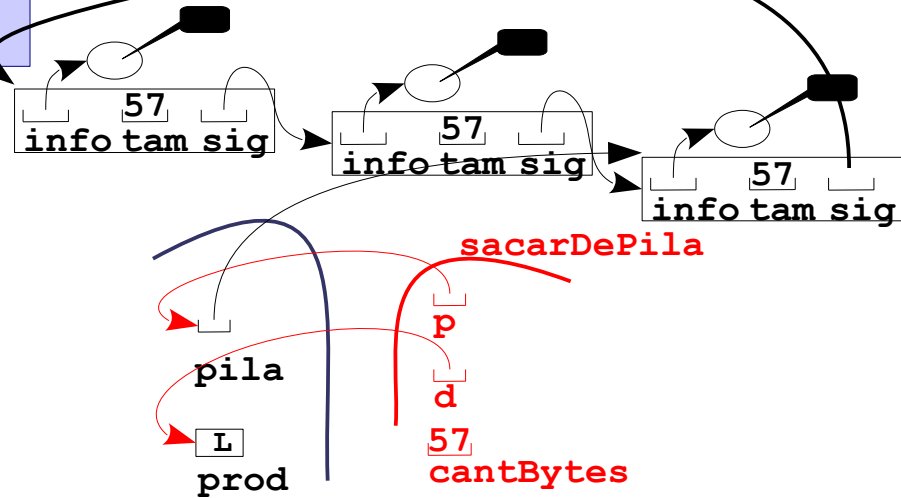


```
main.c x main.h x productos.c x productos.h x pila.c x pila.h x pila.c x pila.h x
115 while(sacarDePila(&pila, &prod, sizeof(tProd)))
116     mostrarProducto(&prod);
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar un elemento de pila
distinto al último

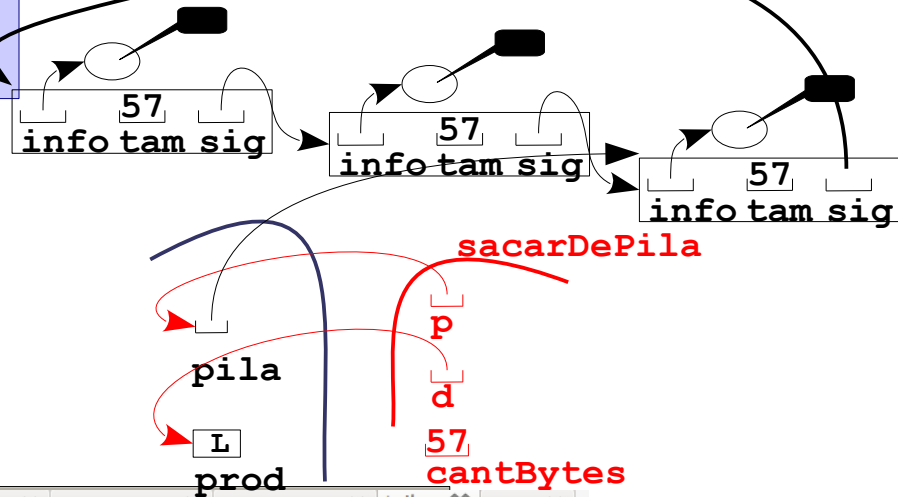


```
main.c x main.h x productos.c x productos.h x pila.c x pila.h x pila.c x pila.h x
115 while(sacarDePila(&pila, &prod, sizeof(tProd)))
116     mostrarProducto(&prod);
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar un elemento de pila
distinto al último

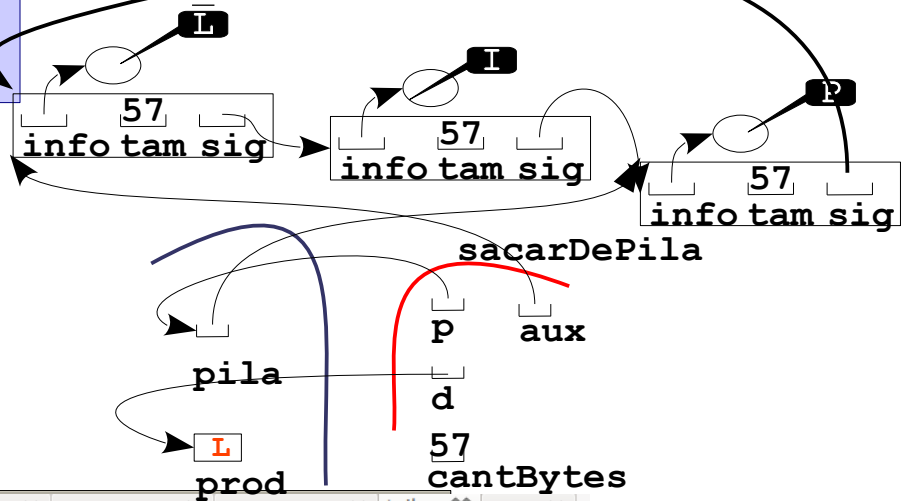


```
main.c x main.c x main.h x productos.c x productos.h x lineasTexto.c x lineasTexto.h x *pila.c x pila.h x
115
116
117
68
69 int sacarDePila(tPila *p, void *d, unsigned cantBytes)
70 {
71     tNodo *aux;
72
73     if(*p == NULL)
74         return 0;
75     aux = (*p)->sig;
76     memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
77     if(aux == *p)
78         *p = NULL;
79     else
80         (*p)->sig = aux->sig;
81     free(aux->info);
82     free(aux);
83     return 1;
84 }
85
```


Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar un elemento de pila
Distinto al último

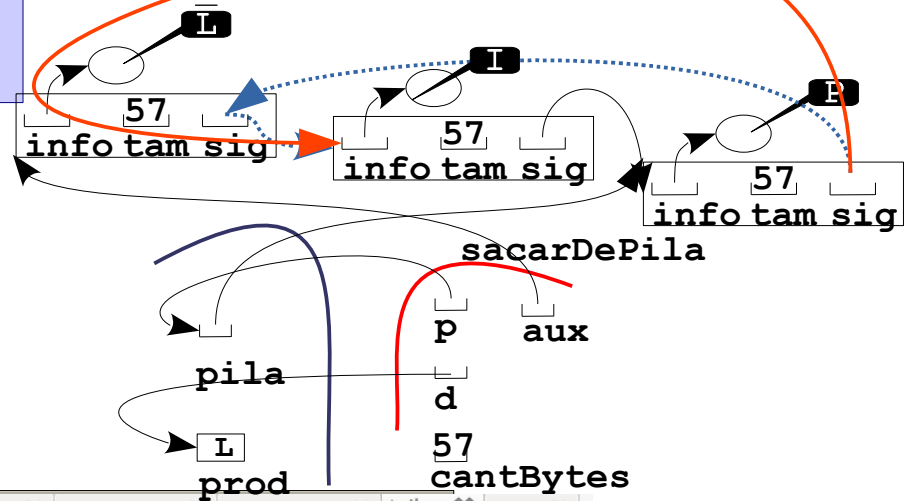


```
main.c x main.c x main.h x productos.c x productos.h x lineasTexto.c x lineasTexto.h x *pila.c x pila.h x
115
116
117
68
69  int sacarDePila(tPila *p, void *d, unsigned cantBytes)
70  {
71      tNodo *aux;
72
73      if(*p == NULL)
74          return 0;
75      aux = (*p)->sig;
76      memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
77      if(aux == *p)
78          *p = NULL;
79      else
80          (*p)->sig = aux->sig;
81      free(aux->info);
82      free(aux);
83      return 1;
84  }
85
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar un elemento de pila
distinto al último

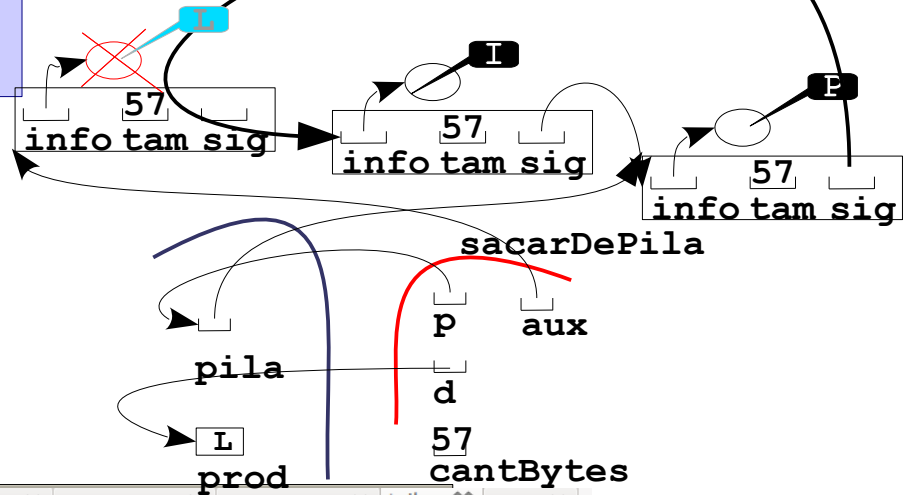


```
main.c x main.c x main.h x productos.c x productos.h x lineasTexto.c x lineasTexto.h x *pila.c x pila.h x
115
116
117
68
69  int sacarDePila(tPila *p, void *d, unsigned cantBytes)
70  {
71      tNodo *aux;
72
73      if(*p == NULL)
74          return 0;
75      aux = (*p)->sig;
76      memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
77      if(aux == *p)
78          *p = NULL;
79      else
80          (*p)->sig = aux->sig;
81      free(aux->info);
82      free(aux);
83      return 1;
84  }
85
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar un elemento de pila
distinto al último

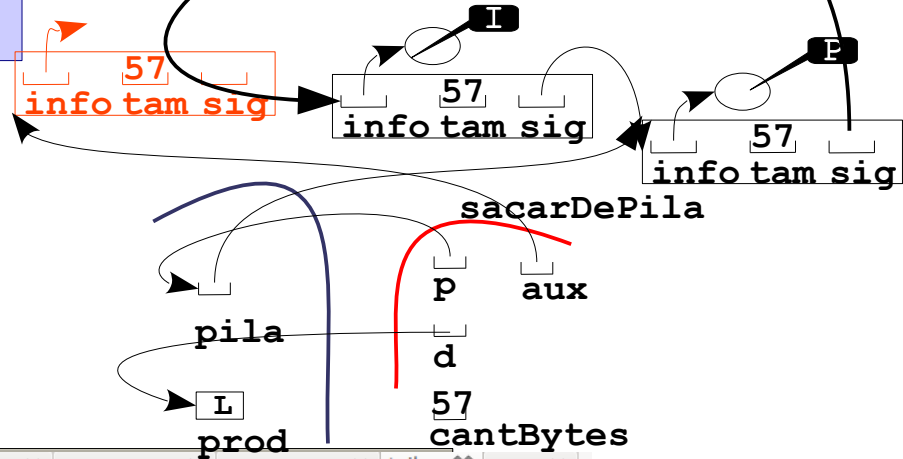


```
main.c x main.c x main.h x productos.c x productos.h x lineasTexto.c x lineasTexto.h x *pila.c x pila.h x
115
116
117
68
69  int sacarDePila(tPila *p, void *d, unsigned cantBytes)
70  {
71      tNodo *aux;
72
73      if(*p == NULL)
74          return 0;
75      aux = (*p)->sig;
76      memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
77      if(aux == *p)
78          *p = NULL;
79      else
80          (*p)->sig = aux->sig;
81      free(aux->info);
82      free(aux);
83      return 1;
84  }
85
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar un elemento de pila
distinto al último

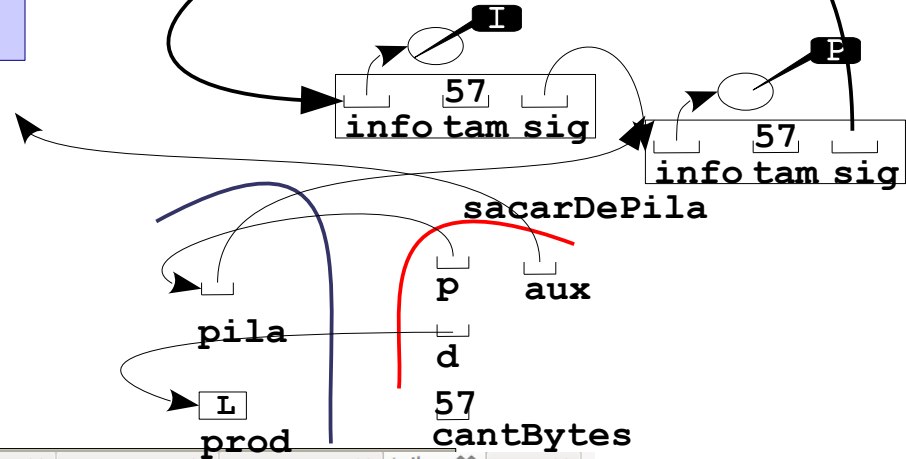


```
main.c x main.c x main.h x productos.c x productos.h x lineasTexto.c x lineasTexto.h x *pila.c x pila.h x
115
116
117
68
69  int sacarDePila(tPila *p, void *d, unsigned cantBytes)
70  {
71      tNodo *aux;
72
73      if(*p == NULL)
74          return 0;
75      aux = (*p)->sig;
76      memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
77      if(aux == *p)
78          *p = NULL;
79      else
80          (*p)->sig = aux->sig;
81      free(aux->info);
82      free(aux);
83      return 1;
84  }
85
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar un elemento de pila
Distinto al último

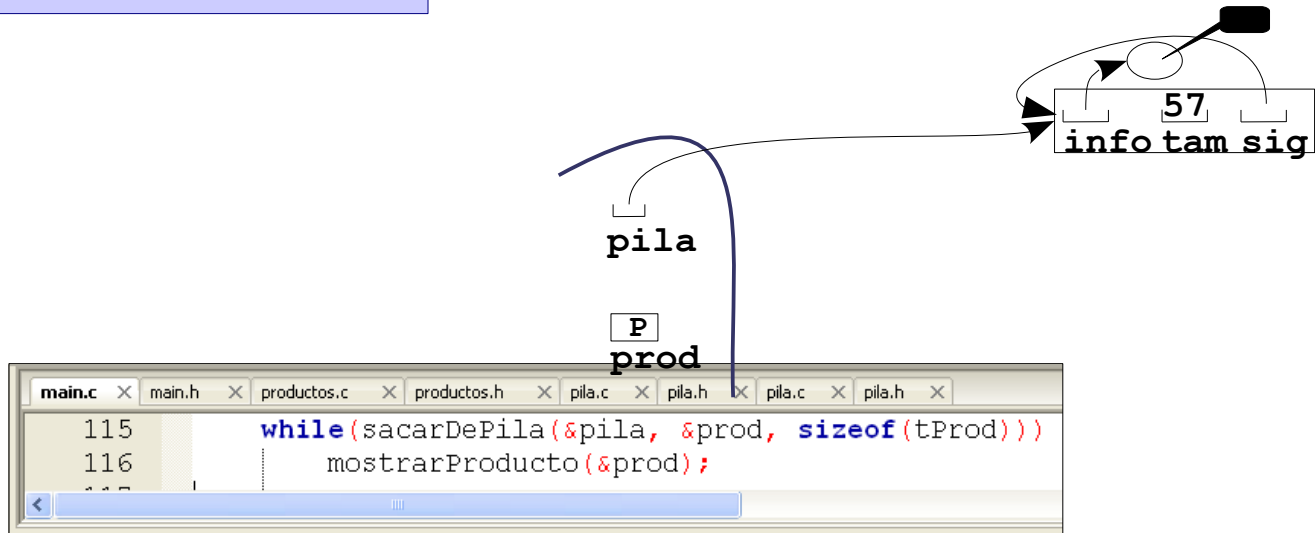


```
main.c x main.c x main.h x productos.c x productos.h x lineasTexto.c x lineasTexto.h x *pila.c x pila.h x
115
116
117
68
69  int sacarDePila(tPila *p, void *d, unsigned cantBytes)
70  {
71      tNodo *aux;
72
73      if(*p == NULL)
74          return 0;
75      aux = (*p)->sig;
76      memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
77      if(aux == *p)
78          *p = NULL;
79      else
80          (*p)->sig = aux->sig;
81      free(aux->info);
82      free(aux);
83      return 1;
84  }
85
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

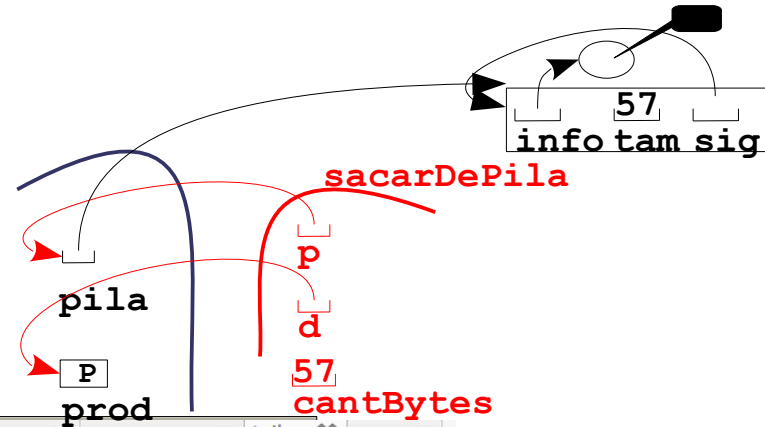
Sacar último elemento
de pila



Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar último elemento
de pila

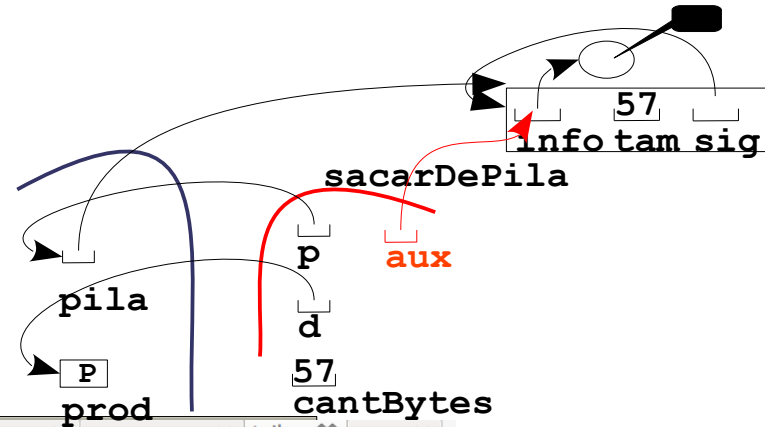


```
main.c x main.c x main.h x productos.c x productos.h x lineasTexto.c x lineasTexto.h x *pila.c x pila.h x
115
116
117
68
69 int sacarDePila(tPila *p, void *d, unsigned cantBytes)
70 {
71     tNodo *aux;
72
73     if(*p == NULL)
74         return 0;
75     aux = (*p)->sig;
76     memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
77     if(aux == *p)
78         *p = NULL;
79     else
80         (*p)->sig = aux->sig;
81     free(aux->info);
82     free(aux);
83     return 1;
84 }
85
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar último elemento
de pila

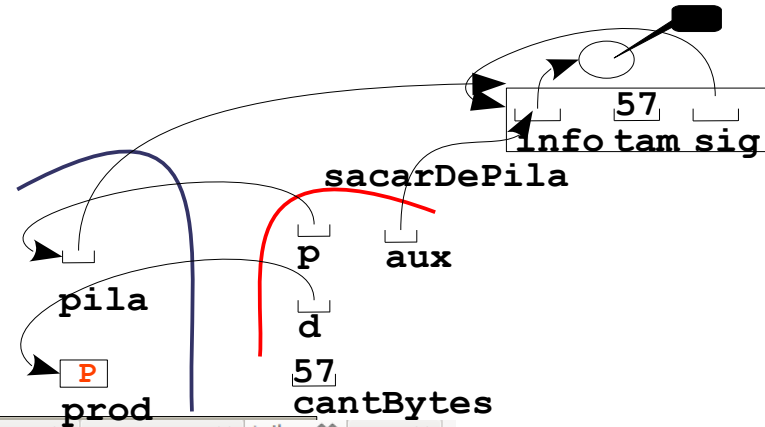


```
main.c x main.c x main.h x productos.c x productos.h x lineasTexto.c x lineasTexto.h x *pila.c x pila.h x
115
116
117
68
69  int sacarDePila(tPila *p, void *d, unsigned cantBytes)
70  {
71      tNodo *aux;
72
73      if(*p == NULL)
74          return 0;
75      aux = (*p)->sig;
76      memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
77      if(aux == *p)
78          *p = NULL;
79      else
80          (*p)->sig = aux->sig;
81      free(aux->info);
82      free(aux);
83      return 1;
84  }
85
```


Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar último elemento
de pila

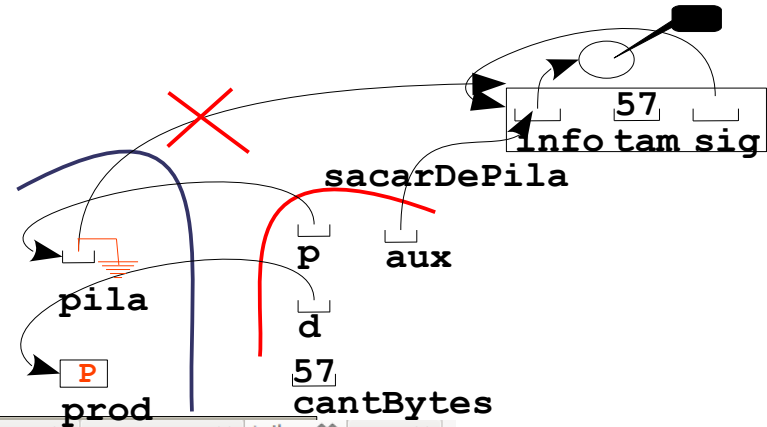


```
main.c x main.c x main.h x productos.c x productos.h x lineasTexto.c x lineasTexto.h x *pila.c x pila.h x
115
116
117
68
69  int sacarDePila(tPila *p, void *d, unsigned cantBytes)
70  {
71      tNodo *aux;
72
73      if(*p == NULL)
74          return 0;
75      aux = (*p)->sig;
76      memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
77      if(aux == *p)
78          *p = NULL;
79      else
80          (*p)->sig = aux->sig;
81      free(aux->info);
82      free(aux);
83      return 1;
84  }
85
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar último elemento
de pila

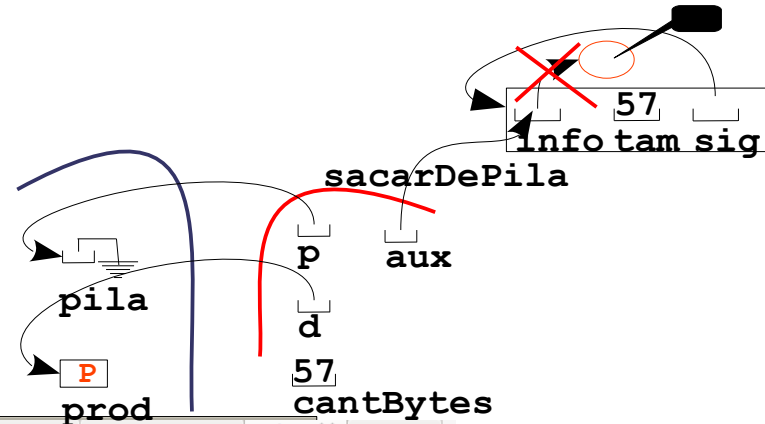


```
main.c x main.c x main.h x productos.c x productos.h x lineasTexto.c x lineasTexto.h x *pila.c x pila.h x
115
116
117
68
69  int sacarDePila(tPila *p, void *d, unsigned cantBytes)
70  {
71      tNodo *aux;
72
73      if(*p == NULL)
74          return 0;
75      aux = (*p)->sig;
76      memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
77      if(aux == *p)
78          *p = NULL;
79      else
80          (*p)->sig = aux->sig;
81      free(aux->info);
82      free(aux);
83      return 1;
84  }
85
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar último elemento
de pila

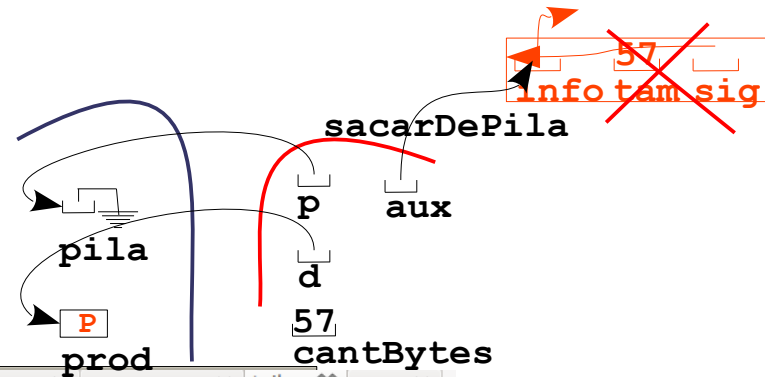


```
main.c x main.c x main.h x productos.c x productos.h x lineasTexto.c x lineasTexto.h x *pila.c x pila.h x
115
116
117
68
69  int sacarDePila(tPila *p, void *d, unsigned cantBytes)
70  {
71      tNodo *aux;
72
73      if(*p == NULL)
74          return 0;
75      aux = (*p)->sig;
76      memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
77      if(aux == *p)
78          *p = NULL;
79      else
80          (*p)->sig = aux->sig;
81      free(aux->info);
82      free(aux);
83      return 1;
84  }
85
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar último elemento
de pila

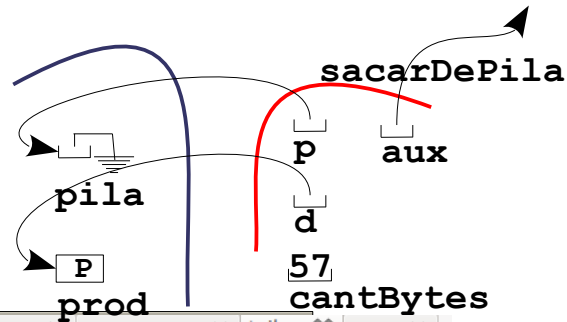


```
main.c x main.c x main.h x productos.c x productos.h x lineasTexto.c x lineasTexto.h x *pila.c x pila.h x
115
116
117
68
69  int  sacarDePila(tPila *p, void *d, unsigned cantBytes)
70  {
71      tNodo *aux;
72
73      if(*p == NULL)
74          return 0;
75      aux = (*p)->sig;
76      memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
77      if(aux == *p)
78          *p = NULL;
79      else
80          (*p)->sig = aux->sig;
81      free(aux->info);
82      free(aux);
83      return 1;
84  }
85
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar último elemento
de pila

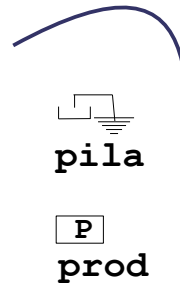


```
main.c x main.c x main.h x productos.c x productos.h x lineasTexto.c x lineasTexto.h x *pila.c x pila.h x
115
116
117
68
69  int sacarDePila(tPila *p, void *d, unsigned cantBytes)
70  {
71      tNodo *aux;
72
73      if(*p == NULL)
74          return 0;
75      aux = (*p)->sig;
76      memcpy(d, aux->info, minimo(cantBytes, aux->tamInfo));
77      if(aux == *p)
78          *p = NULL;
79      else
80          (*p)->sig = aux->sig;
81      free(aux->info);
82      free(aux);
83      return 1;
84  }
85
```

Tipos de Datos Abstractos

Tipo de dato Pila Dinámica Circular.

Sacar último elemento
de pila



```
main.c x main.h x productos.c x productos.h x pila.c x pila.h x pila.c x pila.h x
115 while(sacarDePila(&pila, &prod, sizeof(tProd)))
116     mostrarProducto(&prod);
```