Reserva dinàmica de memòria

- Fins ara hem utilitzat **emmagatzemament estàtic** de forma que a l'inici del programa declaraven les variables amb la seva mida i així reservàvem espai en memòria. Per exemple, per portar el control de notes d'una assignatura:

```
int assignatura[ 40 ];
```

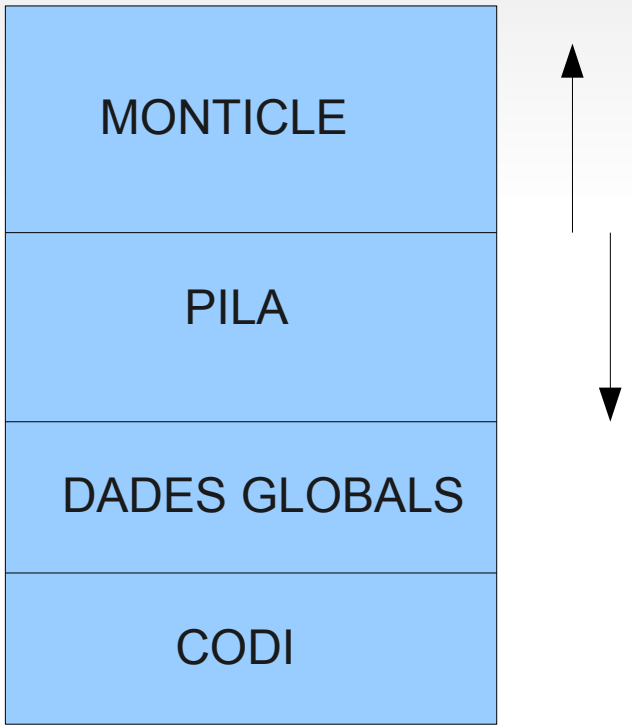
- Hem reservat 40 enters (40x4bytes) fixos. Els arrays són molt útils quan sabem el nombre exacte d'elements a utilitzar en el programa. Però i si la mida del array la sabem sols en el moment de l'execució:

```
printf("Quants alumnes tens? ");
scanf("%d", &num);
int assignatura [ num ];
```

- Aquestes sentències donaran un error de compilació, ja que el compilador requereix que la mida del array siga constant.
- Moltes vegades no sabem la mida que tendrà un array, cadena... i podem optar per fer una reserva no molt gran i quedar-nos curts o declarar una mida molt gran amb el risc de desaprofitar espai. El mètode més convenient per resoldre este problema són els punters i la reserva dinàmica de memòria.
- Al assignar memòria dinàmicament aquesta reserva es realitza en temps d'execució al contrari que fins ara fèiem que es reservava en temps de compilació. Així podrem reservar la mida justa de memòria segons les necessitats del programa.

Mapa de memòria d'un programa

• El codi del programa se situa en segments de memòria anomenats segments de codi. Les dades del programa com són les variables globals es situen en el segment de dades. Les variables locals i de les funcions estan en la pila. La memòria que queda lliure s'anomena memòria del monticle (heap) o memòria lliure. Al sol·licitar memòria dinàmicament s'assigna des d'aquest darrer segment.(monticle).



- La reserva en memòria en la pila es realitza de dalt a baix i al contrari en el monticle.

Assignació dinàmica de memòria

- S'utilitzen dos funcions bàsiques per a l'assignació dinàmica de memòria. Amb la funció **malloc()** se sol·licita memòria del monticle. Amb la funció **free()** es torna memòria al monticle. Aquestes funcions es troben a la llibreria <stdlib.h>
- El prototip per a la funció **malloc()** és

```
void *malloc (size_t grandària);
```

- Com que torna void generalment es realitzarà una conversió (casting) al tipus del punter:

```
tipus *punter;
punter = (tipus *) malloc (mida en bytes);
```

- Per exemple

```
long *p_long;
p_long = (long *) malloc (32);
```

- Per no fer el codi depenent de la màquina se sol utilitzar la funció sizeof():

```
long *p_long;
p_long = (long *) malloc (sizeof(long));
```

- Per reservar un buffer (vector) de 10 enters

```
int *p_buffer;
p_buffer = (int *) malloc (sizeof(int)*10);
```

Assignació dinàmica de memòria

- La funció **malloc()** torna la direcció on s'ha reservat la memòria o si no hi ha lloc torna NULL;

```
float *bloc_mem;
bloc_mem = (float *) malloc (100*sizeof(float));
if (bloc_mem==NULL) puts ("No hi ha espai en memòria");
```

- Reservant la memòria nosaltres mateix ens permetrà ajustar la mida de les cadenes o vectors:

```
int *p_assignatures;  //quasi equivalent a int assignatures[];
printf("Quants alumnes tens? ");
scanf("%d", &num);
p_assignatures= (int *) malloc (sizeof(int)*num);
```

- Com mostrem per pantalla el 3er element del vector (*assignatures[2]*)?

```
printf("%d", *( p_assignatures + 2 ) );
```

- Podrem ajustar la mida de les cadenes però ens farà falta un vector capaç de guardar-la inicialment:

```
char cad_aux[300], *ptr;
fgets(cad_aux,300,stdin);  //Llegim la cadena per teclat
longitud = strlen(cad_aux) + 1 //Longitud de cadena + caràcter '\0'
ptr = (char *) malloc (sizeof(char)*longitud));
strcpy(ptr,cad_aux);
```

Assignació dinàmica de memòria

- La funció **free()** té com a prototip

***void free (void** *bloc);*

- Esta funció torna al monticle el bloc de memòria apuntat per *bloc*, sent *bloc* un punter proporcionat per una cridà a **malloc()**. La memòria tornada per **free()** pot tornar a ser assignada per una altra crida a **malloc()**.
- **Després de finalitzar el bloc de memòria reservat es convenient alliberar-lo per evitar consumir massa recursos.**
- Per exemple

```
int *p_buffer,i = 0;
p_buffer = (int *) malloc (sizeof(int)*10);
...
while (i<10){
  printf("%d", *p_buffer);
  p_buffer++;
  i++;
}
free(p_buffer);   //Alliberem tot el bloc de memòria
```

Assignació dinàmica de memòria

- **La funció calloc()**

- Calloc és una altra funció de reserva de memòria que permet reservar un bloc de memòria prou gran per a un array de “n” elements de la grandària especificada i inicialitza el bloc de memòria al valor zero.
- Tornarà un punter al nou bloc de memòria reservat, o NULL en el cas que no puga reservar-ho.

```
void *calloc(size_t nitems, size_t size);
```

- *Generalment farem una conversió al tipus del punter:*

```
tipus *punter;
punter = (tipus *) calloc (nombre_elements, mida en bytes);
```

- Per exemple:

```
int *p_buffer;
p_buffer = (int *) calloc (10,sizeof(int));
if ( p_buffer == NULL )
  puts (“Error assignant memòria”);
```

- Aquesta funció estarà indicada per a reservar blocs de memòria per a arrays.

Assignació dinàmica de memòria

- **La funció realloc()**

- La funció realloc() permet ampliar un bloc de memòria reservat anteriorment.
- Tornarà un punter al nou bloc de memòria reservat, o NULL en el cas que no puga reservar-ho.

```
void *realloc(void *bloc, size_t mida_total_final);
```

- Generalment farem una conversió al tipus del punter:

```
tipus *punter;
punter = (tipus *) realloc (punter, mida final del bloc);
```

- Per exemple:

```
int *p_buffer;
int tam=10;
p_buffer = (int *) calloc (tam,sizeof(int));
...
tam= tam +20  //Ampliarem el bloc reservat en 20 més
p_buffer = (int *) realloc(p_buffer,tam*sizeof(int));
```

- **Si el segon argument que li passem és 0 alliberarà la memòria reservada i per tant la seva funcionalitat serà la mateixa a free().**

Regles de l'assignació dinàmica de memòria

- El prototip de les funcions està a stdlib.h
- Les funcions malloc(), realloc() i calloc() tornen el tipus void*, per tant requereixen una conversió al tipus del punter:

```
int *p_buffer;
p_buffer = (int *) malloc (sizeof(int)*10);
```

- Les funcions d'assignació de memòria tenen com argument la mida en bytes per tant serà molt útil la funció sizeof().

```
p_buffer = (int *) realloc(p_buffer,t20*sizeof(int));
```

- La funció realloc() permet expandir el bloc de memòria reservat.
- Les funcions d'assignació tornen NULL si no s'ha pogut reservar la memòria.

```
if ( p_buffer == NULL )
  puts ("Error assignant memòria");
```

- Es pot utilitzar qualssevol funció d'assignació per reservar diferents tipus de bloc com són els arrays, les estructures, tipus primitius...

```
struct alumne *p_struct;
p_struct = (struct alumne *) calloc(10, sizeof(struct alumne);
```

- La memòria reservada convé alliberar-la nosaltres mateixos amb la funció free:

```
free(p_buffer);
```

Assignació dinàmica de memòria

- **La funció realloc()**

- La funció realloc() permet ampliar un bloc de memòria reservat anteriorment.
- Tornarà un punter al nou bloc de memòria reservat, o NULL en el cas que no puga reservar-ho.

```
void *realloc(void *bloc, size_t mida_total_final);
```

- Generalment farem una conversió al tipus del punter:

```
tipus *punter;
punter = (tipus *) realloc (punter, mida final en bytes);
```

- Per exemple:

```
int *p_buffer;
int tam=10;
p_buffer = (int *) calloc (tam,sizeof(int));
...
tam= tam +20  //Ampliarem el bloc reservat en 20 més
p_buffer = (int *) realloc(p_buffer,tam);
```

- **Si el segon argument que li passem és 0 alliberarà la memòria reservada i per tant la seva funcionalitat serà la mateixa a free().**