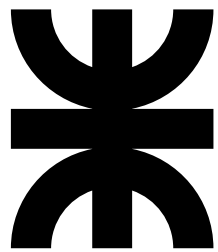
Informática II

Memoria dinámica

Gonzalo F. Pérez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2018 –

Memoria dinámica
Introducción

- Asignación dinámica de memoria es la capacidad de obtener más memoria para datos en tiempo de ejecución de un programa

Memoria dinámica
Introducción

- Asignación dinámica de memoria es la capacidad de obtener más memoria para datos en tiempo de ejecución de un programa
- Se puede solicitar y devolver memoria al sistema operativo

Memoria dinámica
Introducción

- Asignación dinámica de memoria es la capacidad de obtener más memoria para datos en tiempo de ejecución de un programa
- Se puede solicitar y devolver memoria al sistema operativo
- La asignación dinámica de memoria, o memoria dinámica existe en un sector llamado *heap*

Memoria dinámica
Introducción

- Asignación dinámica de memoria es la capacidad de obtener más memoria para datos en tiempo de ejecución de un programa
- Se puede solicitar y devolver memoria al sistema operativo
- La asignación dinámica de memoria, o memoria dinámica existe en un sector llamado *heap*
- Las funciones `malloc` y `free`, junto al operador `sizeof` se utilizan en la asignación dinámica de memoria (archivo de cabecera `stdlib.h`)

Memoria dinámica

Disposición de la memoria en un programas C

Cuando se carga un programa en memoria, esta queda organizada en bloques llamados *segmentos*

Memoria dinámica

Disposición de la memoria en un programas C

Cuando se carga un programa en memoria, esta queda organizada en bloques llamados *segmentos*

- *text* (o código): contiene código binario del programa (solo lectura)

Memoria dinámica

Disposición de la memoria en un programas C

Cuando se carga un programa en memoria, esta queda organizada en bloques llamados *segmentos*

- *text* (o código): contiene código binario del programa (solo lectura)
- *data*: subdividido en dos

Memoria dinámica

Disposición de la memoria en un programas C

Cuando se carga un programa en memoria, esta queda organizada en bloques llamados *segmentos*

- *text* (o código): contiene código binario del programa (solo lectura)
- *data*: subdividido en dos
  - datos inicializados: variables globales, estáticas, y datos constantes

Memoria dinámica

Disposición de la memoria en un programas C

Cuando se carga un programa en memoria, esta queda organizada en bloques llamados *segmentos*

- *text* (o código): contiene código binario del programa (solo lectura)
- *data*: subdividido en dos
  - datos inicializados: variables globales, estáticas, y datos constantes
  - datos no inicializados: almacenados en `bss` (Block Started by Symbol)

Memoria dinámica

Disposición de la memoria en un programas C

Cuando se carga un programa en memoria, esta queda organizada en bloques llamados *segmentos*

- *text* (o código): contiene código binario del programa (solo lectura)
- *data*: subdividido en dos
  - datos inicializados: variables globales, estáticas, y datos constantes
  - datos no inicializados: almacenados en `bss` (Block Started by Symbol)
- *stack*: para almacenar variables locales, argumentos pasados a funciones, y dirección de retorno (crece hacia abajo)

Memoria dinámica

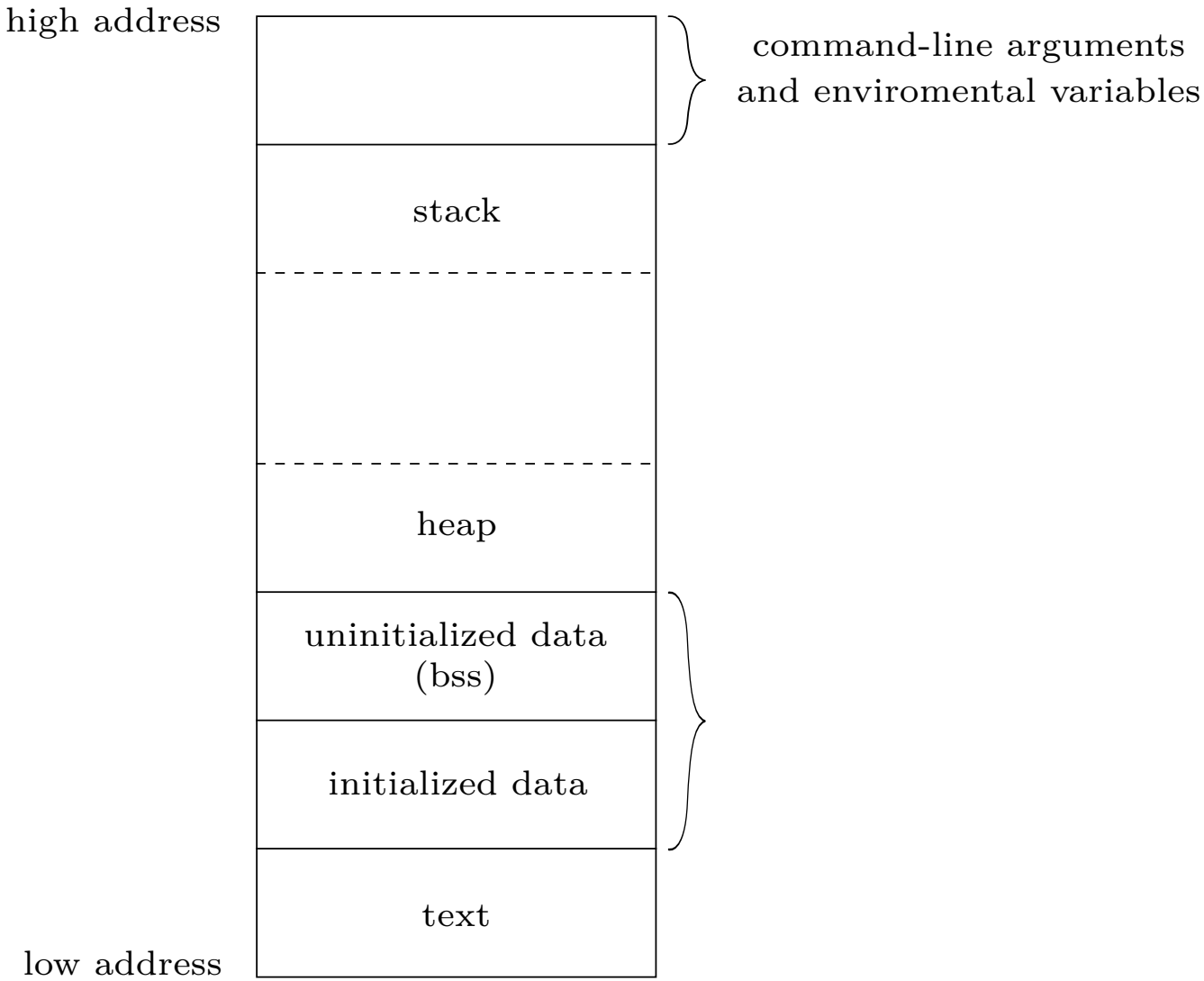
Disposición de la memoria en un programas C

Cuando se carga un programa en memoria, esta queda organizada en bloques llamados *segmentos*

- *text* (o código): contiene código binario del programa (solo lectura)
- *data*: subdividido en dos
  - datos inicializados: variables globales, estáticas, y datos constantes
  - datos no inicializados: almacenados en `bss` (Block Started by Symbol)
- *stack*: para almacenar variables locales, argumentos pasados a funciones, y dirección de retorno (crece hacia abajo)
- *heap*: cuando se asigna memoria (`malloc`) en tiempo de ejecución la memoria se obtiene del *heap* (crece hacia arriba)

Memoria dinámica

Disposición de la memoria en un programas C



Memoria dinámica
Segmento de datos

Una cadena definida como

```
char s[] = "Hola␣mundo";
```

o un enunciado

```
int debug = 1;
```

fuera de **main** (global) almacena las variables en el área de datos inicializados como read-write.

Memoria dinámica
Segmento de datos

Una cadena definida como

```
char s[] = "Hola␣mundo";
```

o un enunciado

```
int debug = 1;
```

fuera de **main** (global) almacena las variables en el área de datos inicializados como read-write.

Un enunciado global como

```
const char *string = "Hola␣mundo";
```

almacena la cadena literal en el área inicializada como read-only, y la variable puntero en el área inicializada como read-write.

Memoria dinámica

Funciones de la biblioteca estándar

Ciclo de vida de las variables en el *heap* son:

1. asignar espacio a la variable usando `malloc()` o `calloc()`

Memoria dinámica

Funciones de la biblioteca estándar

Ciclo de vida de las variables en el *heap* son:

1. asignar espacio a la variable usando `malloc()` o `calloc()`
2. redimensionar la variable utilizando `realloc()` [Opcional]

Memoria dinámica

Funciones de la biblioteca estándar

Ciclo de vida de las variables en el *heap* son:

1. asignar espacio a la variable usando `malloc()` o `calloc()`
2. redimensionar la variable utilizando `realloc()` [Opcional]
3. liberar la memoria con `free()`

Memoria dinámica

Funciones de la biblioteca estándar

Ciclo de vida de las variables en el *heap* son:

1. asignar espacio a la variable usando `malloc()` o `calloc()`
2. redimensionar la variable utilizando `realloc()` [Opcional]
3. liberar la memoria con `free()`

Prototipos de funciones – archivo de cabecera `stdlib.h`

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

Memoria dinámica

Funciones de la biblioteca estándar

Ciclo de vida de las variables en el *heap* son:

1. asignar espacio a la variable usando `malloc()` o `calloc()`
2. redimensionar la variable utilizando `realloc()` [Opcional]
3. liberar la memoria con `free()`

Prototipos de funciones – archivo de cabecera `stdlib.h`

```
void *malloc(size_t size);

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

Memoria dinámica

Funciones de la biblioteca estándar

Ciclo de vida de las variables en el *heap* son:

1. asignar espacio a la variable usando `malloc()` o `calloc()`
2. redimensionar la variable utilizando `realloc()` [Opcional]
3. liberar la memoria con `free()`

Prototipos de funciones – archivo de cabecera `stdlib.h`

```
void *malloc(size_t size);

void *calloc(size_t nmemb, size_t size);

void *realloc(void *ptr, size_t size);
```

Memoria dinámica

Funciones de la biblioteca estándar

Ciclo de vida de las variables en el *heap* son:

1. asignar espacio a la variable usando `malloc()` o `calloc()`
2. redimensionar la variable utilizando `realloc()` [Opcional]
3. liberar la memoria con `free()`

Prototipos de funciones – archivo de cabecera `stdlib.h`

```
void *malloc(size_t size);

void *calloc(size_t nmemb, size_t size);

void *realloc(void *ptr, size_t size);

void free(void *ptr);
```

Memoria dinámica

Ejemplos

Tipos básicos y arreglos

```
int *edad = malloc(sizeof(int));
```

Memoria dinámica

Ejemplos

Tipos básicos y arreglos

```
int *edad = malloc(sizeof(int));

double *promedio = malloc(sizeof(double));
```

Memoria dinámica

Ejemplos

Tipos básicos y arreglos

```
int *edad = malloc(sizeof(int));

double *promedio = malloc(sizeof(double));

char *cadena = malloc(20 * sizeof(char));
```

Memoria dinámica

Ejemplos

Tipos básicos y arreglos

```
int *edad = malloc(sizeof(int));

double *promedio = malloc(sizeof(double));

char *cadena = malloc(20 * sizeof(char));

float *lista = malloc(3 * sizeof(float));
```

Memoria dinámica

Ejemplos

Tipos básicos y arreglos

```
int *edad = malloc(sizeof(int));

double *promedio = malloc(sizeof(double));

char *cadena = malloc(20 * sizeof(char));

float *lista = malloc(3 * sizeof(float));
```

Estructuras

```
struct paciente {
  char apellido[20];
  char nombre[20];
  int edad;
  float peso;
  float altura;
};
struct paciente *jperez = malloc(sizeof(struct paciente));
```

Memoria dinámica

Ejemplos

Tipos básicos y arreglos

```
int *edad = malloc(sizeof(int));

double *promedio = malloc(sizeof(double));

char *cadena = malloc(20 * sizeof(char));

float *lista = malloc(3 * sizeof(float));
```

Estructuras

```
struct paciente {
  char apellido[20];
  char nombre[20];
  int edad;
  float peso;
  float altura;
};
struct paciente *jperez = malloc(sizeof(struct paciente));
```

No olvidar liberar la memoria con **`free()`** [Memory leak]

Memoria dinámica

Ejercicios

1. Escribir un programa que reserve espacio en memoria para las variables de tipos básicos del ejemplo, le asigne valores, imprima los valores, y libere la memoria.