

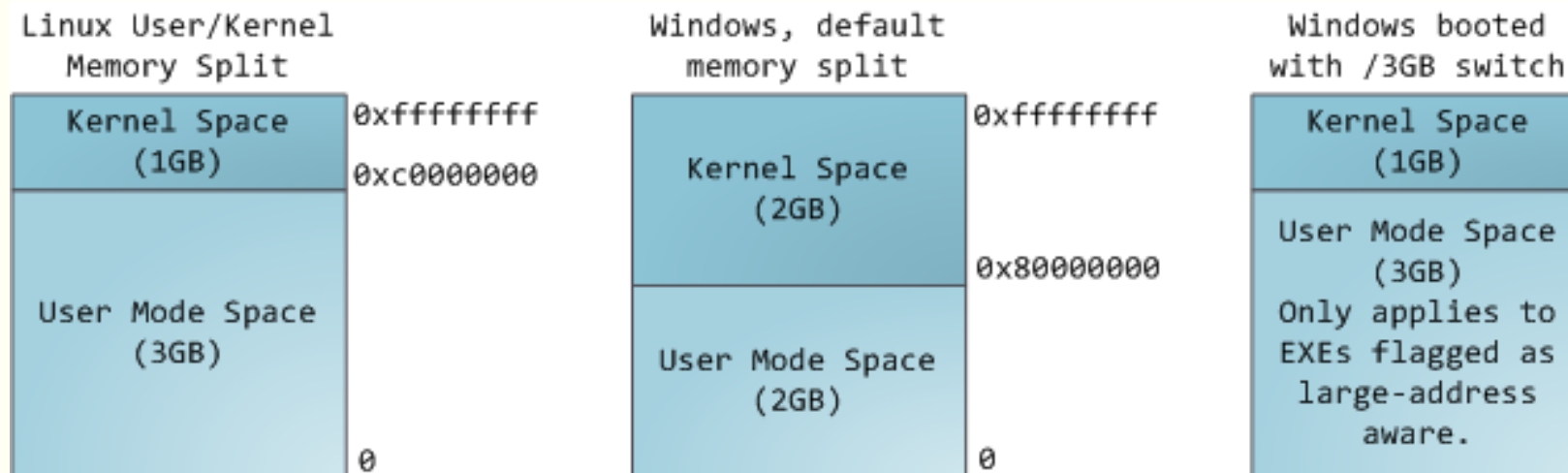


РОЗПОДІЛ ПАМ'ЯТІ ПРИ РОБОТІ ПРОГРАМИ

Питання 3.5.

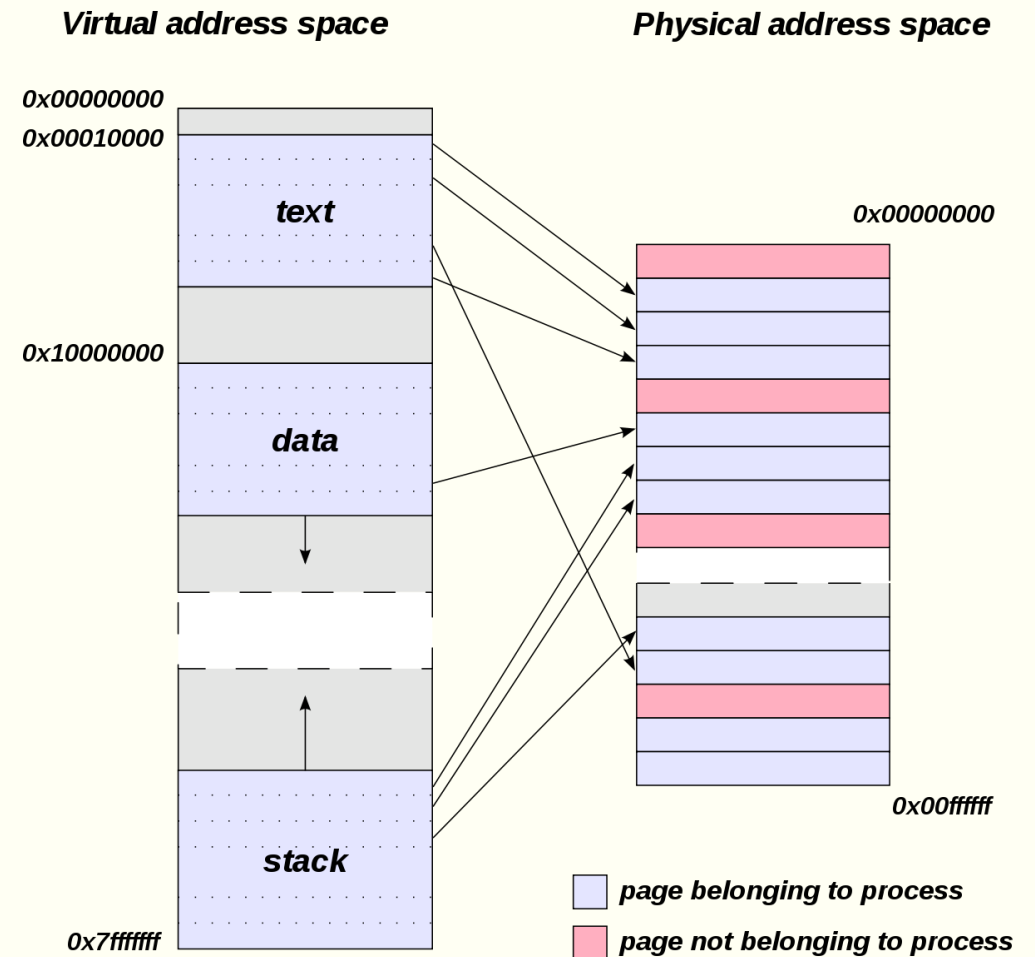
Програма в пам'яті

- Кожен процес у багатозадачній ОС працює у власному віртуальному **адресному просторі**.
 - У 32-бітному режимі завжди доступний блок з **4ГБ** адрес у пам'яті.
 - Віртуальні адреси відображаються на фізичну пам'ять за допомогою **page tables**, які підтримуються ядром ОС та consulted процесором.
 - Особливість: як тільки задіюються віртуальні адреси, вони застосовуються до всього програмного забезпечення, працюючого на машині, в т. ч. ядра ОС.



Віртуальна пам'ять та вказівники

- Додаток розбивається на сторінки / фрейми (pages/frames), які відображають області основної пам'яті.
 - Сторінки додатку розміщені в різних, потенційно непослідовних областях пам'яті.
 - У пам'ять можуть завантажуватись не всі сторінки відразу.
 - Якщо ОС потребує пам'ять, зайняту сторінкою, інформацію можна тимчасово «перекинути» (swap out) на вторинний носій, а пізніше перезавантажити в інше місце в пам'яті.
- Кожна програма передбачає, що вона має доступ до всього простору фізичної пам'яті.
 - Насправді це не так – використовувані програмою адреси є віртуальними.
 - ОС за потреби відображає (maps) віртуальну адресу на адресу реальної фізичної пам'яті.



Поняття класу пам'яті

- Описує характеристики змінної/функції.
 - В основному, область видимості (scope), зв'язування (binding) та тривалість зберігання (lifetime).
- Область видимості (scope):** описує частину або частини програми, з яких можна отримати доступ до конкретного ідентифікатора.

```
#include <stdio.h>
```

Глобальна область видимості
доступні всі функції з файлу `stdio.h`

```
int main(void) {
```

```
    int x[10] = {1,4,34,53,1,92,4,8,2,44};  
    int min = 99999;  
    int max = -99999;
```

Локальна область видимості:
переменные `x`, `min` и `max` видны везде ниже
их объявления

```
    for(int i = 0; i<10; i++){  
        if (x[i] < min)  
            min = x[i];  
    }
```

Область видимости переменной `i`
за пределами тела цикла переменная `i` не
существует

```
    for(int i = 0; i<10; i++){  
        if (x[i] > max)  
            max = x[i];  
    }
```

Область видимости переменной `i`
это новая переменная `i`, она никак не
связана с той, что выше

```
    printf("max: %d\tmin: %d\n",max,min);
```

```
    return 0;
```

```
}
```

Область
видимості в
межах файлу

Область
видимості в
межах блоку
(між
фігурними
дужками)

Області видимості змінних

▪ *У межах блоку*

- Блоком є все тіло функції та тіла складених операторів МП.
- Змінну видно тільки до кінця блоку.

▪ *У межах функції*

- Відноситься лише до міток, що застосовуються з оператором goto.
- Якщо мітка вперше з'являється у внутрішньому блоці функції, її область видимості – вся функція.

▪ *У межах прототипу функції*

- Застосовується до формальних параметрів у прототипі функції.
- `int mighty(int mouse, double large);`
- Область видимості – від місця визначення параметру до кінця оголошення прототипу (імена параметрів ролі не грають).

▪ *У межах файлу*

- Змінну видно від місця її оголошення до кінця файлу.
- Такі змінні називають глобальними.

Зв'язування змінних

- Змінна в мові С має одне з наступних видів зв'язування:
 - *Зовнішнє зв'язування*
 - *Внутрішнє зв'язування*
 - *Відсутність зв'язування.* Характерне для області видимості в межах блоку, функції або прототипу функції.
- Змінні з областю видимості в межах файлу можуть мати внутрішнє або зовнішнє зв'язування.
 - Змінна з внутрішнім зв'язуванням може використовуватись будь-де в одиниці трансляції (файлі з кодом + його заголовкові файли)
 - Змінні з зовнішнім зв'язуванням може застосовуватись будь-де в програмі.
- Часто загальні терміни скорочуються:
 - Область видимості в межах файлу із внутрішнім зв'язуванням = область видимості в межах файлу (статичні глобальні змінні)
 - Область видимості в межах файлу із зовнішнім зв'язуванням = область видимості в межах програми = глобальна область видимості

Тривалість зберігання (lifetime)

- Характеризує *постійність* доступних через ідентифікатори об'єктів.
 - **Статична тривалість зберігання** – об'єкт існує протягом виконання програми. Характерна для змінних з областю видимості в межах файлу.
 - Ключове слово `static` стосується не тривалості зберігання, а зв'язування.
 - **Потокова тривалість зберігання** – об'єкт існує з моменту його оголошення та до завершення потоку. Працює в багатопоточних додатках.
 - Специфікатор `_Thread_local` вказує на те, що кожен потік отримує власну копію змінної.
 - **Автоматична тривалість зберігання** – пам'ять для автоматичних змінних є тимчасовою та може застосовуватись багатократно. Характерна для змінних з областю видимості в межах блоку.
 - Незначний виняток: масиви змінної довжини – існують не від початку блоку, а від місця оголошення.
 - Раніше використовувані локальні змінні мають автоматичну тривалість зберігання.
 - Змінна може мати область видимості в межах блоку, проте статичну тривалість зберігання:
 - Змінна `ct` зберігається в статичній пам'яті: існує від запуску і до завершення програми,
 - Проте область видимості обмежена функцією `more()`
 - **Виділена тривалість зберігання** – для динамічно виділеної пам'яті.

```
void more(int number)
{
    int index;
    static int ct = 0;
    ...
    return 0;
}
```

Класи пам'яті (Storage Classes) мови C

Модифікатор	Застосування	Область дії	Тривалість життя
auto	Локальне	Блок	Тимчасова
register	Локальне	Блок	Тимчасова
extern	Глобальне	Блок	Тимчасова
static	Локальне	Блок	Постійна
	Глобальне	Файл	
volatile	Глобальне	Файл	Постійна

Класи пам'яті. Автоматичні змінні

- Модифікатор `auto` використовується при описі локальних змінних.
 - На практиці його пропускають, оскільки він застосовується за замовчуванням.
 - Застосовується тільки до змінних, які видно виключно в блоці, в якому вони оголошені.

- Приклад:

```
{  
    int mount;  
    auto int month;  
}
```

- defines two variables with in the same storage class.
- 'auto' can only be used within functions, i.e., local variables.

Класи пам'яті. Регістрові змінні

- Модифікатор `register` вказує компілятору спробувати розмістити відповідну змінну в регістрах процесора, а не в оперативній пам'яті.
 - Якщо спроба невдала, змінна поводить себе як автоматична.
 - Розміщення в регістрах оптимізує програмний код за швидкістю.
 - Проте через невелику кількість регістрів таких змінних може бути мало.
 - Розмір такої змінної обмежується розміром регістра (зазвичай одне машинне слово).
 - До змінної не можна застосувати оператор взяття адреси `&`.

- Приклад

```
{  
    register int miles;  
}
```

- Регістровими мають бути тільки ті змінні, які потребують швидкого доступу, зокрема лічильники.

Класи пам'яті. Зовнішні змінні

- Використовуються, якщо програма складається з кількох модулів, між якими потрібно передавати інформацію.
 - Зовнішній клас пам'яті використовується, щоб надати посилання на глобальну змінну, видиму в UCIX програмних файлах.
 - В одному модулі змінна **оголошується** глобальною, а в інших файлах – з модифікатором `extern`.

main.c

```
#include <stdio.h>
```

```
int count;
```

```
extern void write_extern();
```

```
main() {  
    count = 5;  
    write_extern();  
}
```

support.c

```
#include <stdio.h>
```

```
extern int count;
```

```
void write_extern(void) {  
    printf("count is %d\n", count);  
}
```

Класи пам'яті. Статичні змінні

- Схожі на глобальні змінні.
 - Для опису використовують модифікатор `static`.
 - Компілятору наказується підтримувати існування локальної змінної протягом життя програми замість створення та знищення цієї змінної кожного разу, при вході та виході з області видимості.
- Якщо статична змінна оголошена глобально,
 - ініціалізується при запуску програми,
 - область видимості – від оголошення до кінця файлу.
- Якщо статична змінна оголошена локально (у функції або блоці),
 - ініціалізується при першому вході у відповідну функцію або блок,
 - статичні локальні змінні здатні тримати свої значення між викликами функцій.
- Статичні змінні не можуть бути оголошеними в інших файлах, як зовнішні.
 - Модифікатор `static` також може застосовуватись до глобальних змінних.
 - Область видимості такої змінної – в межах файлу, в якому вона була оголошена.

Класи пам'яті. Статичні змінні

```
#include <stdio.h>

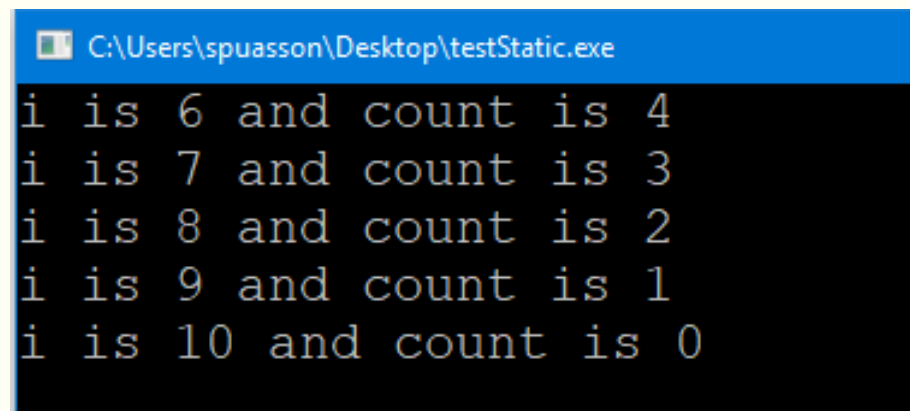
void func(void); /* function declaration */

static int count = 5; /* global variable */

main() {
    while(count-->0) {
        func();
    }
    return 0;
}

/* function definition */
void func( void ) {
    static int i = 5; /* local static variable */
    i++;
    printf("i is %d and count is %d\n", i, count);
}
```

- У мові С статична глобальна змінна вказує на існування єдиного її екземпляру, спільного для всіх об'єктів, що її використовують.



```
C:\Users\spuasson\Desktop\testStatic.exe
i is 6 and count is 4
i is 7 and count is 3
i is 8 and count is 2
i is 9 and count is 1
i is 10 and count is 0
```

Модель пам'яті для мови програмування C

```
char* s = "hello";
int i;
char* foo(void)
{
    char* p;
    static int x = 0;
    x += 1;
    i = 1;
    p = malloc( sizeof(char) * strlen(s) + i * x);

    for(int j=0; j < x; ++j)
        strcat(p,s);
    return p;
}
```

.data: assigned "hello" at start
Available until termination

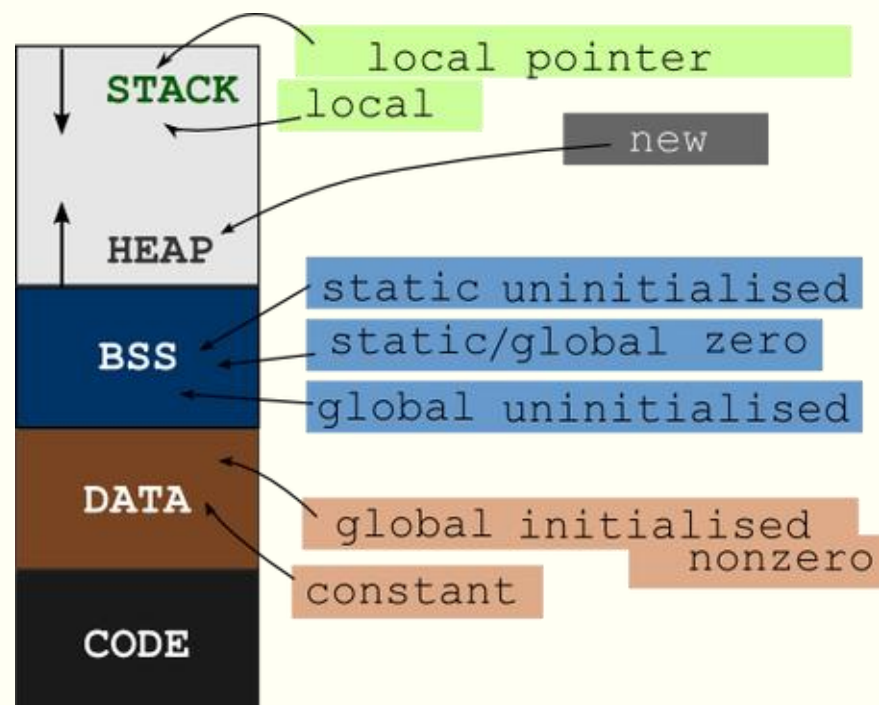
.bss: assigned 0 at start
Available until termination

Stack: allocated at function call
Deallocated on return from foo()

.data: assigned 0 at start
Available until termination

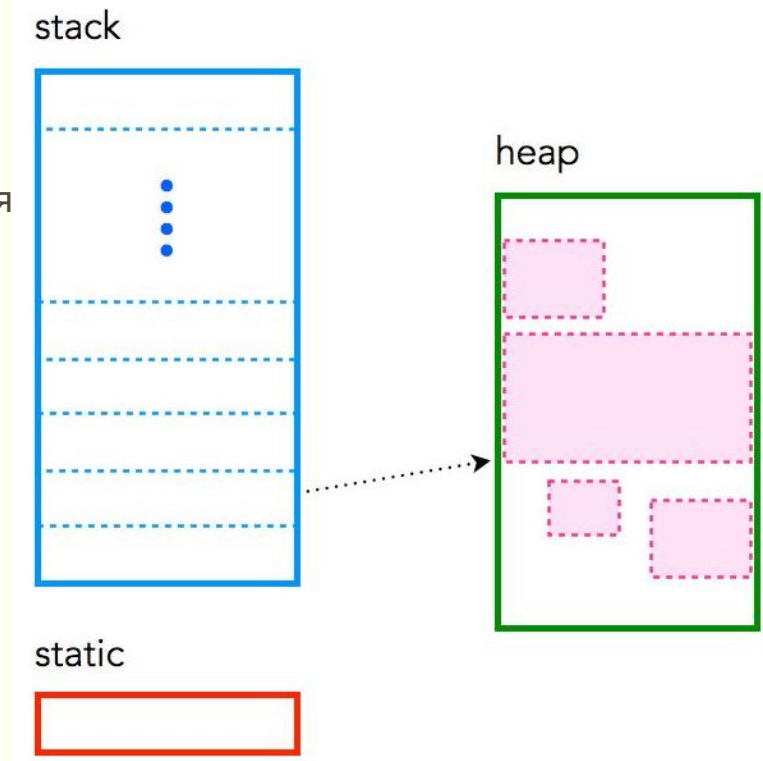
Heap: allocates 6 bytes at malloc()
Deallocated on free()

- Код програми
- Змінні функцій: аргументи, локальні змінні, розташування даних для повернення з функції
- Глобальні змінні / статично аллоковані
- Динамічно аллоковані змінні



Пам'ять та вказівники

- При компіляції робота відбувається з 3 типами пам'яті:
 - **Static/Global – у стековій пам'яті**
 - Статично оголошені змінні оголошуються в цьому виді пам'яті
 - Глобальні змінні також використовують цю область пам'яті, розміщуються в ній при запуску програми та залишаються до її завершення.
 - Доступ до глобальних змінних мають усі функції, а область видимості статичних змінних обмежується функцією, в якій вони оголошені.
 - **Automatic – у стековій пам'яті**
 - Такі змінні оголошуються всередині функції та створюються при виклику функції.
 - Їх область видимості обмежується функцією, а тривалість життя (lifetime) – періодом виконання функції.
 - **Dynamic – у кучі (heap)**
 - Пам'ять виділяється з кучі та може вивільнитись (release) за необхідності.
 - Вказівник указує на виділену (allocated) пам'ять.
 - Область видимості обмежена видимістю відповідних вказівників.
 - Тривалість життя обмежена періодом від алокації до вивільнення пам'яті, на яку вказує відповідний вказівник.



Режими виділення пам'яті в мові С: статичний, автоматичний, динамічний

- **Статична аллокація передбачає, що пам'ять для таких змінних буде виділена при запуску програми**
 - Розмір змінних фіксований.
 - Застосовується до глобальних змінних,
- **Переваги статичного виділення пам'яті**
 - Простота використання.
 - Висока швидкодія процесу виділення.
 - Відсутність потреби аллокації/реаллокації/деаллокації пам'яті.
 - Змінні залишаються перманентно аллокованими.
- **Недоліки статичної аллокації пам'яті**
 - Даремні витрати пам'яті.
 - Неможливість вивільнення пам'яті після завершення її використання.
 - Статично виділена пам'ять очищується на основі області видимості (scope) автоматично.

Динамічне виділення пам'яті (бібліотека `stdlib.h`)

Функція	Що виконує
malloc	Виділяє (allocate) пам'ять заданого розміру та повертає вказівник на перший байт аллокованого простору
calloc	Виділяє пам'ять для елементів масиву. Ініціалізує елементи нулями та повертає вказівник на масив у пам'яті
realloc	Використовується для зміни розміру попередньо виділеного обсягу пам'яті
free	Очищає (вивільняє) попередньо аллоковану пам'ять

- Функція `malloc()` повертає значення типу `void *`.
 - `int *sieve = (int *) malloc(sizeof(int) * length);`
 - Раніше зведення типу було обов'язковим, тепер код без нього вже безпечний:
`int *sieve = malloc(sizeof(int) * length);`
 - Ще один варіант запису:
`int *sieve = malloc(length * sizeof *sieve);`

Функція free()

- При динамічній аллокації виконувати очищення пам'яті потрібно явно.
 - Інакше можуть трапитись помилки роботи з пам'яттю.
 - Функція free() викликається, щоб очистити (release/deallocate) пам'яті.

```
#include <stdio.h>

int main() {
    int* ptr = malloc(10 * sizeof(*ptr));
    if (ptr != NULL) {
        *(ptr + 2) = 50;
        printf("Value of the 2nd integer is %d", *(ptr + 2));
    }
    free(ptr);
}
```

- Відсутність free() призведе до витоку пам'яті у зв'язку з недосяжними аллокаціями (Unreachable allocations)

Функція calloc()

- Скорочена назва від «contiguous allocation» - неперервний розподіл пам'яті.
 - Функція використовується для виділення **кількох** (на відміну від malloc) блоків пам'яті.
 - Використовується для аллокації пам'яті для складних структур даних: масивів, структур тощо.
 - Кожен виділений блок матиме однаковий розмір.
- Синтаксис:
 - `ptr = (cast_type *) calloc (n, size);`
 - Виділяється n блоків пам'яті з розміром кожного рівним size.
 - Після аллокації всі байти зануляються.
 - Повертається вказівник на перший байт виділеної пам'яті.
 - При помилці аллокації повертається null-вказівник.

Приклад використання функції calloc()

```
1 #include <stdio.h>
2 int main() {
3     int i, * ptr, sum = 0;
4     ptr = calloc(10, sizeof(int));
5     if (ptr == NULL) {
6         printf("Error! memory not allocated.");
7         exit(0);
8     }
9     printf("Building and calculating the sequence sum of the first 10 terms \n ");
10
11     for (i = 0; i < 10; ++i) { * (ptr + i) = i;
12         sum += * (ptr + i);
13     }
14     printf("Sum = %d", sum);
15
16     free(ptr);
17
18     return 0;
19 }
```

E:\GDisk\[College]\[фёютш яёюёрьстрээ ёр рыуюёшсь|ўэ| ёутш]\тхър 03. 4юї|фэ| ёшяш фрэші є ёют| т\Code\test_calloc.exe

building and calculating the sequence sum of the first 10 terms n Sum = 45

Process exited after 0.04416 seconds with return value 0

для продолжения нажмите любую клавишу . . .

Функція realloc()

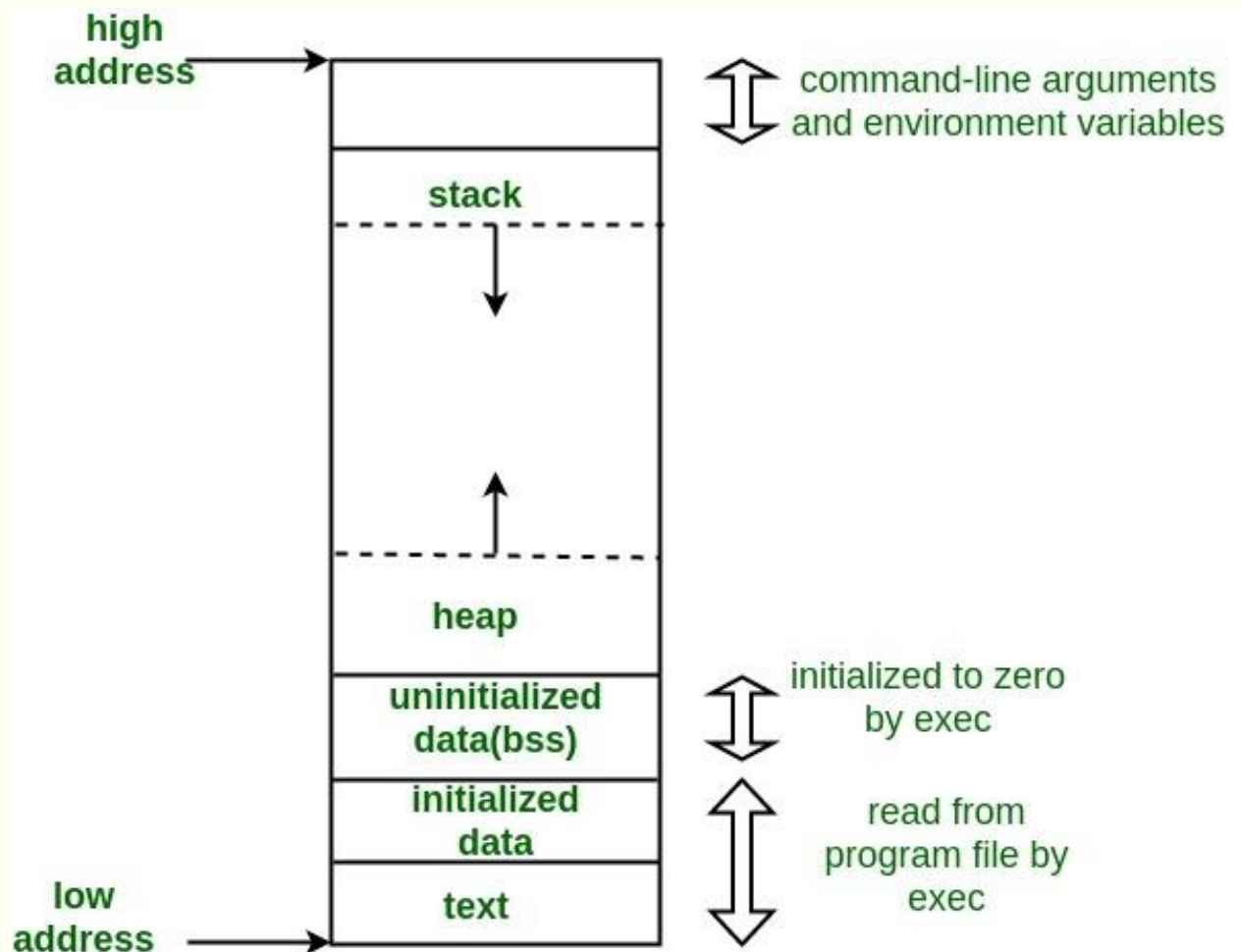
- Використовується для розширення або урізання вже виділеної пам'яті.
- Синтаксис
 - `ptr = realloc (ptr, newsize);`
 - Виділяється новий простір пам'яті заданого розміру `newsize`.
 - Після виконання функції повертається вказівник на перший байт блоку пам'яті.
 - Немає гарантії, що новий виділений блок буде за тою ж адресою, що й раніше.
 - Функція скопіює всі попередні дані в нову область пам'яті та переконається, що вони в безпеці.

```
1  #include <stdio.h>
2  int main () {
3      char *ptr;
4      ptr = (char *) malloc(10);
5      strcpy(ptr, "Programming");
6      printf(" %s, Address = %u\n", ptr, ptr);
7
8      ptr = (char *) realloc(ptr, 20); //ptr is reallocated with new size
9      strcat(ptr, " In 'C'");
10     printf(" %s, Address = %u\n", ptr, ptr);
11     free(ptr);
12     return 0;
13 }
```

Статичне та динамічне виділення пам'яті

Статична аллокація пам'яті	Динамічна аллокація пам'яті
Пам'ять виділяється ДО початку виконання програми	Пам'ять виділяється ПІД ЧАС виконання програми
Дій з аллокації або деаллокації під час виконання програми не відбувається	Зв'язки в пам'яті (memory bindings) встановлюються та руйнуються під час виконання програми
Змінні залишаються перманентно аллокованими	Пам'ять аллокована, поки активний відповідний програмний модуль
Реалізується через стеки та кучі	Реалізується через сегменти даних
Вища швидкодія	Нижча швидкодія
Вимагає більше пам'яті	Потребує менше пам'яті

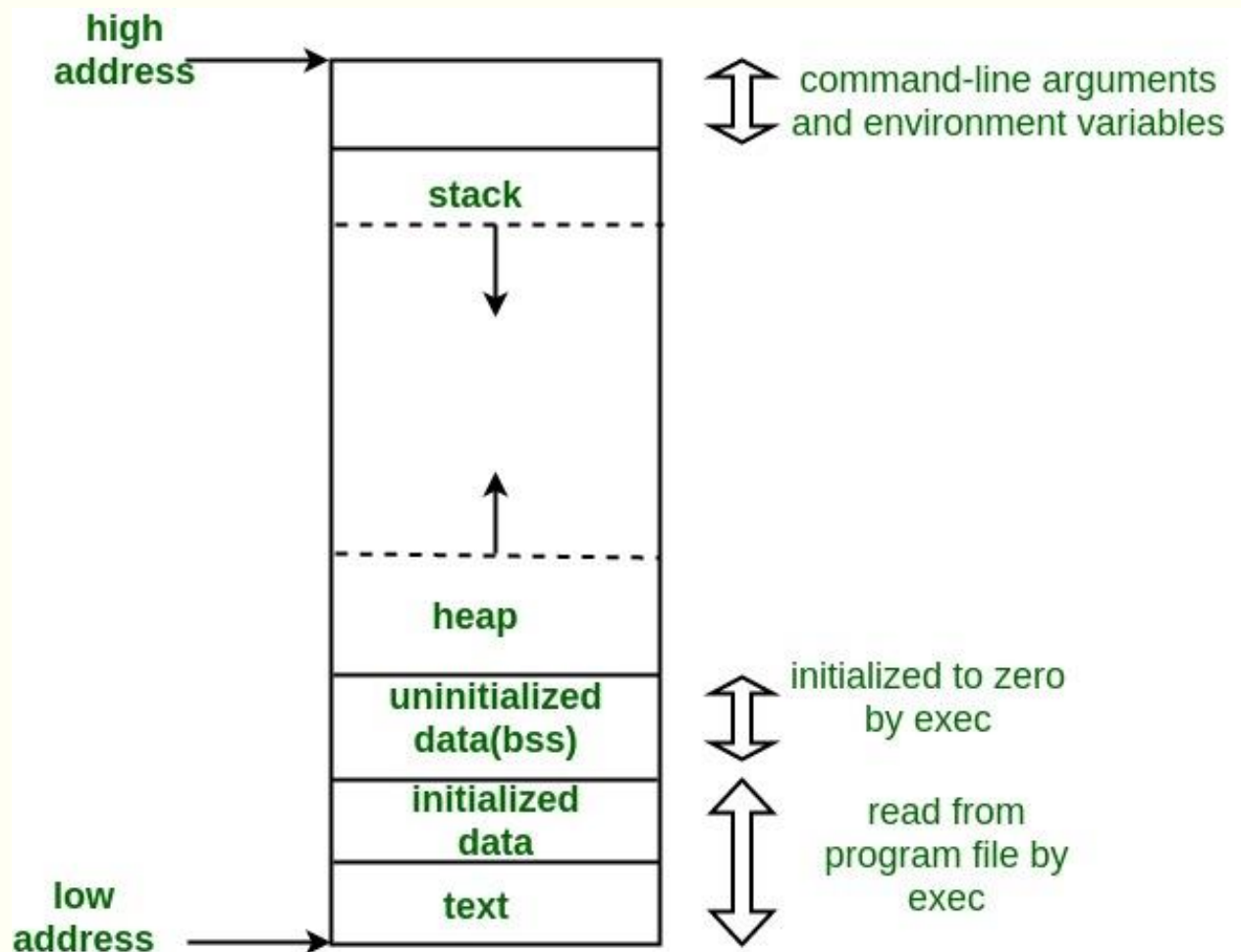
Представлення програми мовою С в пам'яті



1. Текстовий сегмент (Text Segment):

- Також відомий як **сегмент коду** або просто **текст**.
- представляє частини програми в об'єктному файлі або пам'яті, які містять виконувані інструкції.
- У пам'яті може розташовуватись перед кучою або стеком, щоб попереджувати їх переповнення через свій перезапис.
- Зазвичай до текстового сегменту існує спільний доступ (sharable), тому існує потреба лише в одній копії сегменту в пам'яті для часто виконуваних програм, зокрема редакторів тексту, компілятора, консолей тощо.
- Часто сегмент доступний лише для зчитування, що запобігти випадковій зміні інструкцій програми.

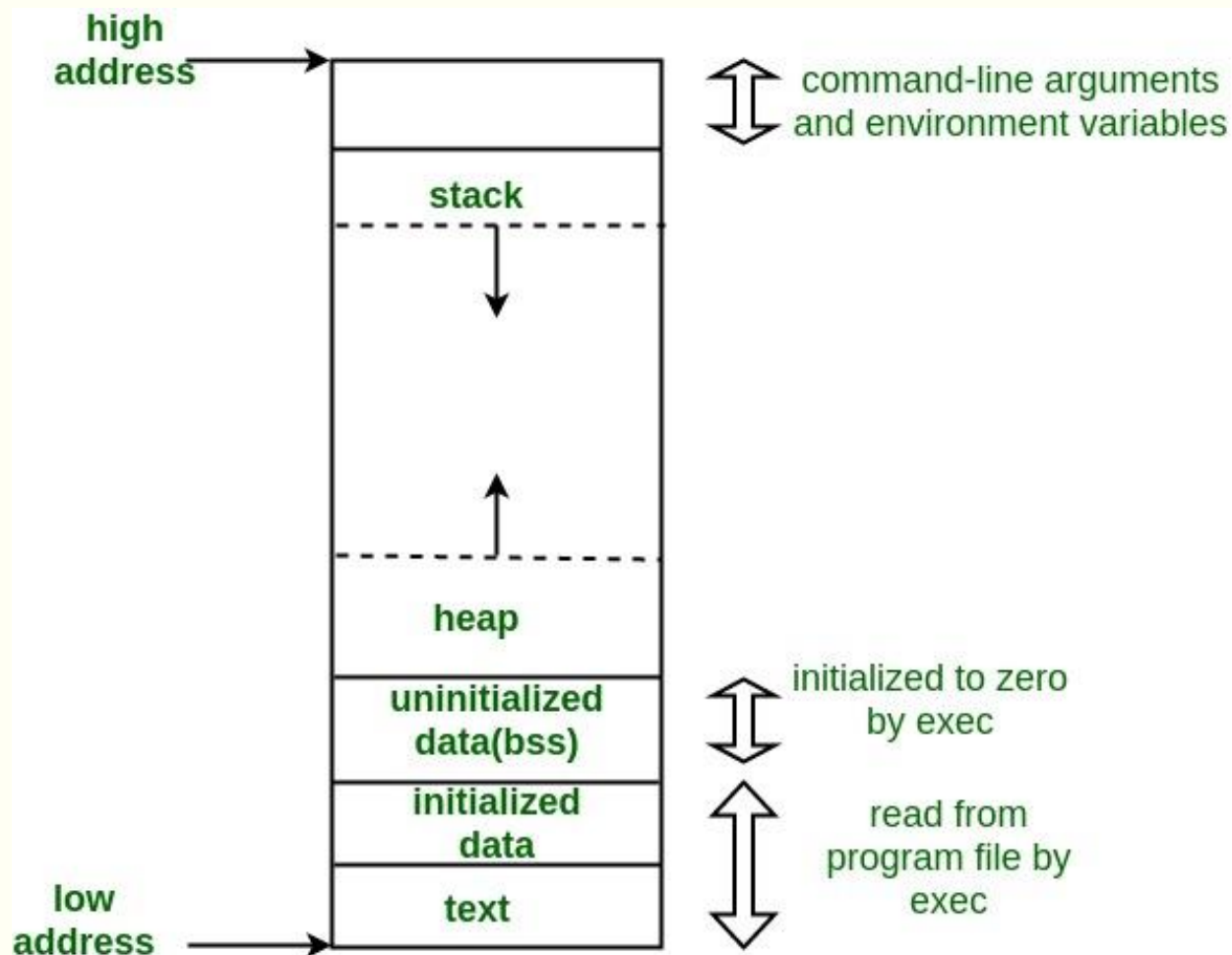
Представлення програми мовою С в пам'яті



2. Сегмент ініціалізованих даних (Initialized Data Segment):

- Часто називають сегментом даних (Data Segment).
- Частина віртуального адресного простору, яка містить глобальні змінні та ініціалізовані програмістом статичні змінні.
- Доступний для читання (initialized read-only area) та запису (initialized read-write area).

Представлення програми мовою С в пам'яті



3. Сегмент неініціалізованих даних (Uninitialized Data Segment):

- Також називають bss-сегментом на честь старовинного асемблерного оператора - "block started by symbol".
 - Дані в цьому сегменті ініціалізуються ядром як арифметичний 0 перед запуском програми на виконання.
-
- Неініціалізовані дані починаються після сегменту даних та містять усі глобальні змінні та статичні змінні, що ініціалізовані нулем або не мають явної ініціалізації в первинному коді.
 - Наприклад, `static int i;`
 - Наприклад, глобально оголошена `int j;`

Демонстрація наповнення сегментів даних (компілятор GCC, 64-бітний)

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     return 0;
6 }
```

```
C:\Program Files (x86)\Dev-Cpp\MinGW64\bin>size testExe/test_segments_1.exe
   text    data     bss     dec     hex filename
 10284    2316    2656   15256   3b98 testExe/test_segments_1.exe
```

```
1 #include <stdio.h>
2
3 //Uninitialized variable stored in bss
4 int global;
5
6 int main(void)
7 {
8     return 0;
9 }
```

```
C:\Program Files (x86)\Dev-Cpp\MinGW64\bin>size testExe/test_segments_2.exe
   text    data     bss     dec     hex filename
 10284    2316    2672   15272   3ba8 testExe/test_segments_2.exe
```

```
1 #include <stdio.h>
2
3 //Uninitialized variable stored in bss
4 int global;
5
6 int main(void)
7 {
8     //Uninitialized static variable stored in bss
9     static int i;
10
11     return 0;
12 }
```

```
C:\Program Files (x86)\Dev-Cpp\MinGW64\bin>size testExe/test_segments_3.exe
   text    data     bss     dec     hex filename
 10284    2316    2672   15272   3ba8 testExe/test_segments_3.exe
```

Демонстрація наповнення сегментів даних (компілятор GCC, 64-бітний)

```
1 #include <stdio.h>
2
3 //Uninitialized variable stored in bss
4 int global;
5
6 int main(void)
7 {
8     //Initialized static variable stored in DS
9     static int i = 100;
10
11     return 0;
12 }
```

```
C:\Program Files (x86)\Dev-Cpp\MinGW64\bin>size testExe/test_segments_4.exe
   text    data     bss      dec     hex filename
  10284    2332    2672   15288   3bb8 testExe/test_segments_4.exe
```

```
1 #include <stdio.h>
2
3 //Initialized variable stored in DS
4 int global = 10;
5
6 int main(void)
7 {
8     //Initialized static variable stored in DS
9     static int i = 100;
10
11     return 0;
12 }
```

```
C:\Program Files (x86)\Dev-Cpp\MinGW64\bin>size testExe/test_segments_5.exe
   text    data     bss      dec     hex filename
  10284    2332    2656   15272   3ba8 testExe/test_segments_5.exe
```

Демонстрація наповнення сегментів даних (компілятор GCC, 32-бітний)

```
C:\Program Files (x86)\Dev-Cpp\MinGW64\bin>size testExe/test_segments_1.exe
  text    data     bss     dec      hex filename
  6836    1488     1104    9428    24d4 testExe/test_segments_1.exe

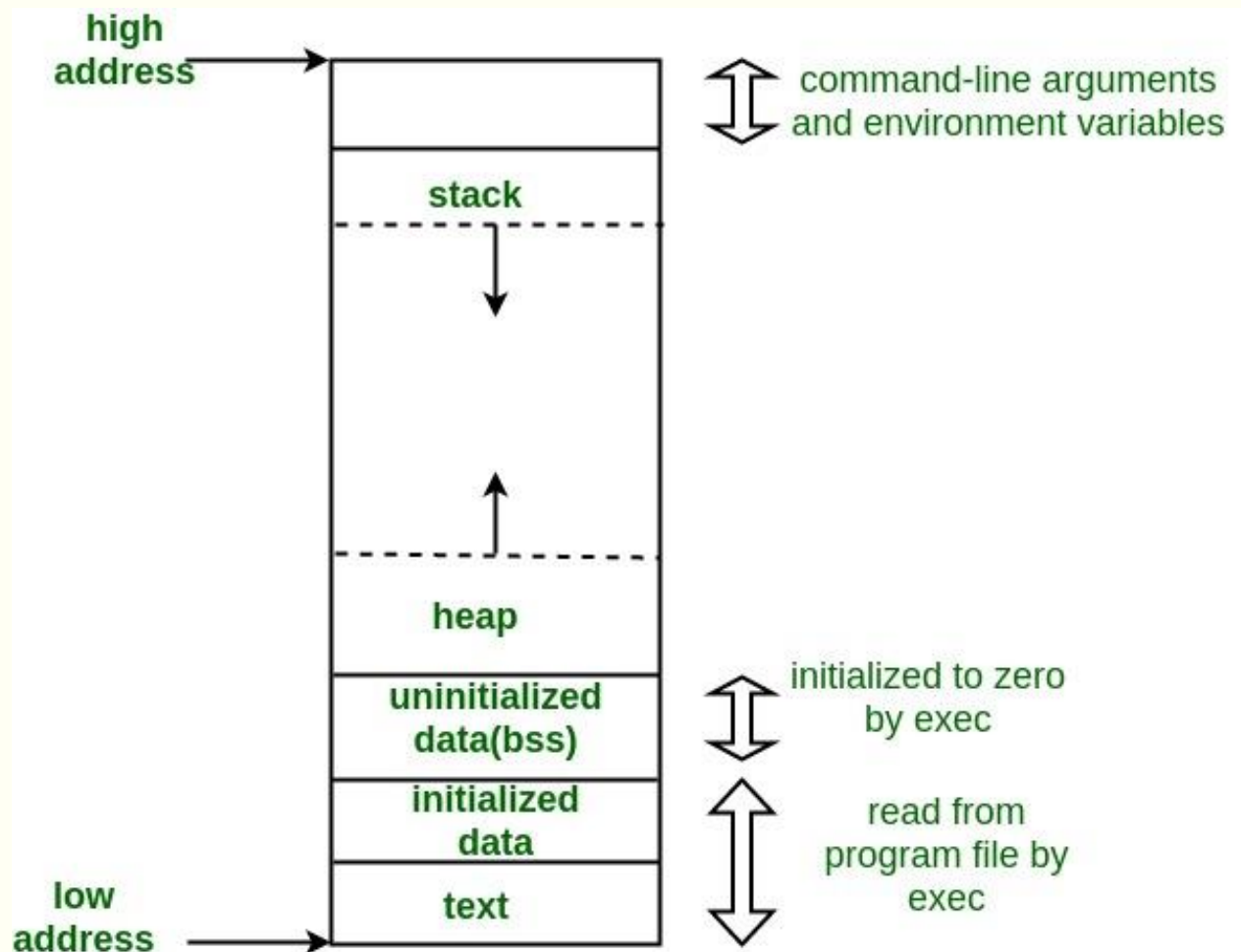
C:\Program Files (x86)\Dev-Cpp\MinGW64\bin>size testExe/test_segments_2.exe
  text    data     bss     dec      hex filename
  6836    1488     1108    9432    24d8 testExe/test_segments_2.exe

C:\Program Files (x86)\Dev-Cpp\MinGW64\bin>size testExe/test_segments_3.exe
  text    data     bss     dec      hex filename
  6836    1488     1108    9432    24d8 testExe/test_segments_3.exe

C:\Program Files (x86)\Dev-Cpp\MinGW64\bin>size testExe/test_segments_4.exe
  text    data     bss     dec      hex filename
  6836    1492     1108    9436    24dc testExe/test_segments_4.exe

C:\Program Files (x86)\Dev-Cpp\MinGW64\bin>size testExe/test_segments_5.exe
  text    data     bss     dec      hex filename
  6836    1496     1104    9436    24dc testExe/test_segments_5.exe
```

Представлення програми мовою С в пам'яті

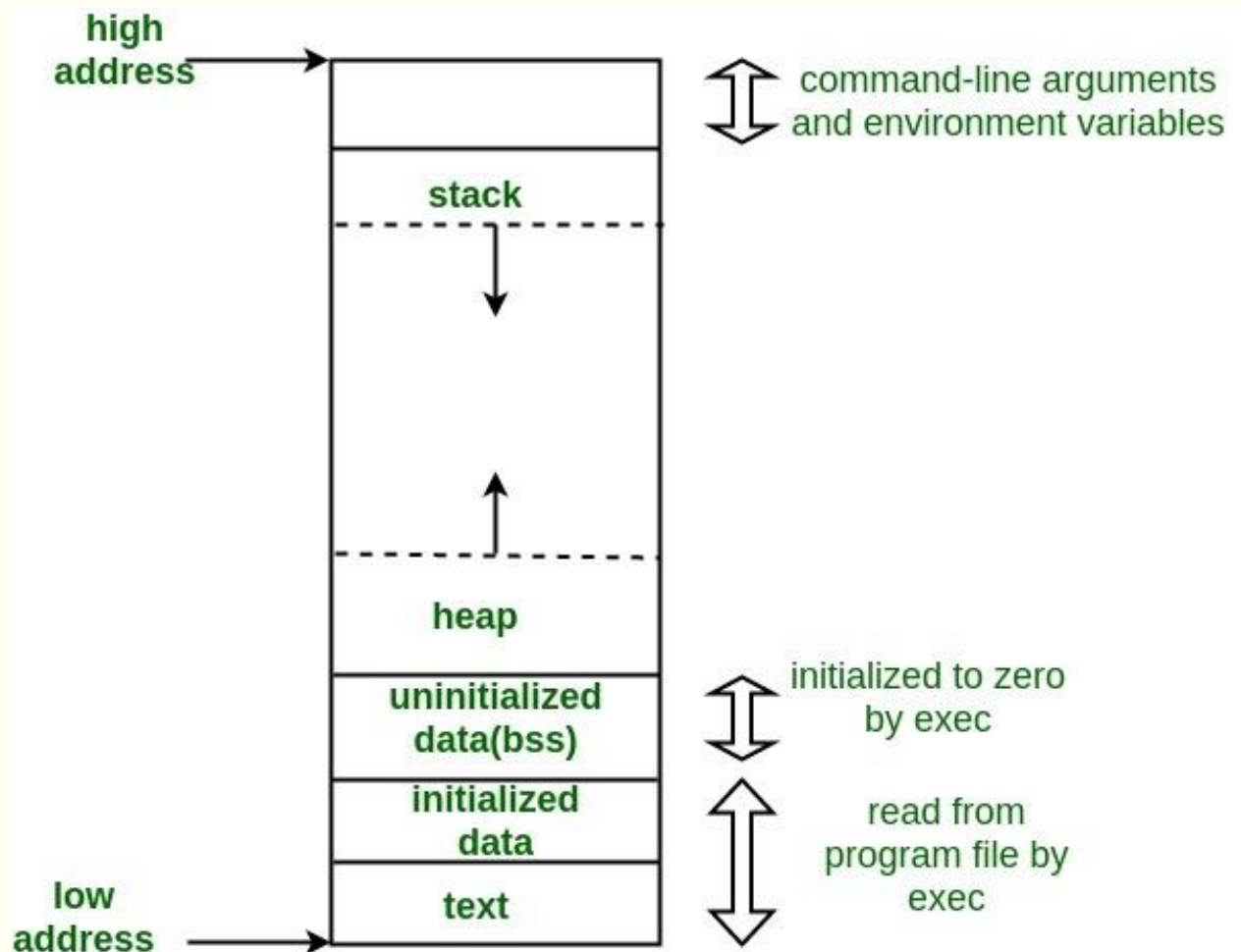


4. Куча (Heap):

Сегмент, у якому зазвичай відбувається динамічне виділення пам'яті (dynamic memory allocation).

- Починається після BSS сегменту.
- Управляється функціями malloc, realloc та free.
- Куча спільна для всіх спільних бібліотек та динамічно завантажуваних модулів всередині процесу.

Представлення програми мовою С в пам'яті

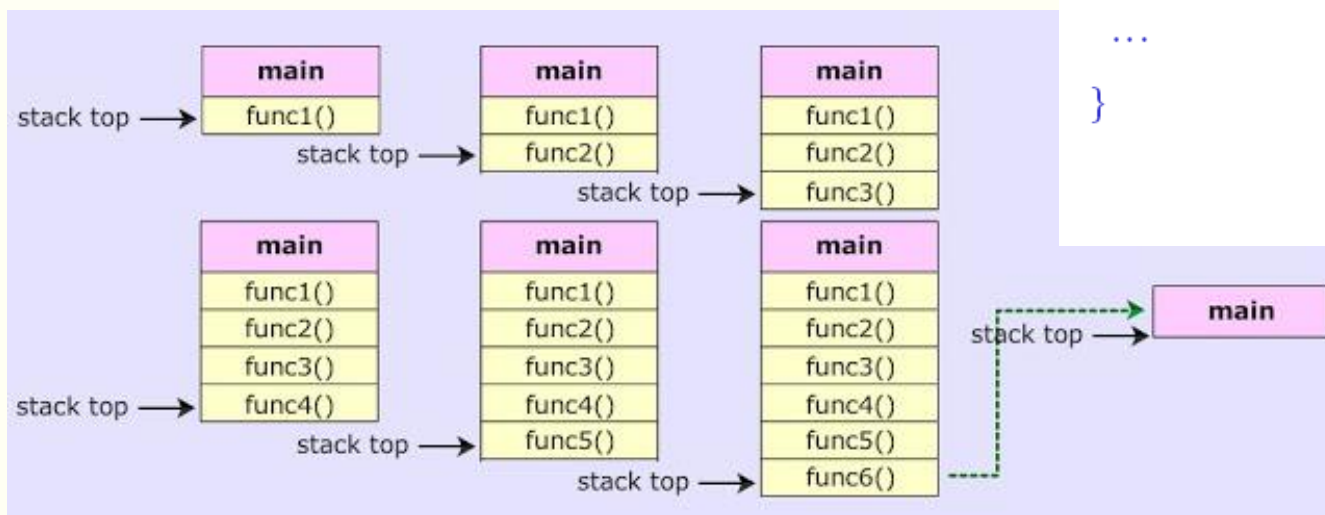


5. Стек (Stack):

- Традиційно суміжна область в пам'яті з кучою, адреси зростають у зворотному напрямку;
- Коли вказівник стеку зустрічається із вказівником кучі, вільна пам'ять закінчується.
- Сегмент стеку містить програмний стек.
- Регістр "stack pointer" відстежує вершину стеку;
- Набір значень, доданих (push) у стек протягом одного виклику функції, називають **стековим кадром** (stack frame);

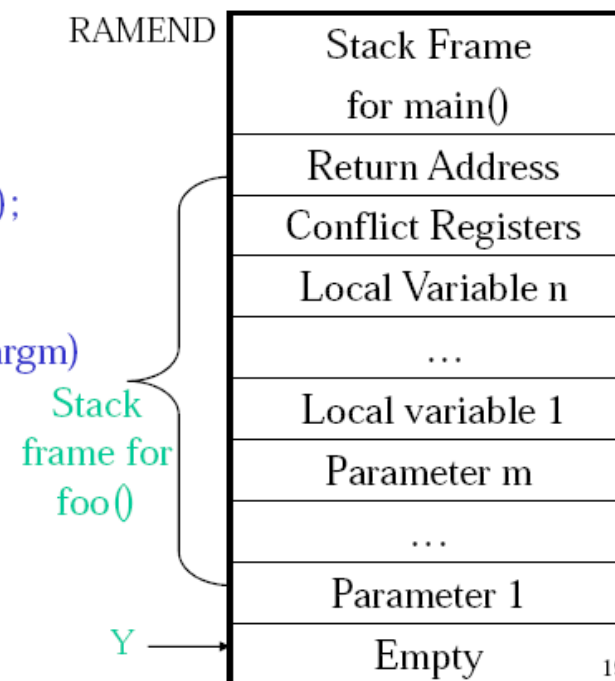
Представлення функцій у стековому кадрі

- У стеку зберігаються *автоматичні* змінні та інформація від кожного виклику функції.
- При виклику функції виділяється ділянка пам'яті для стеку з аргументами, локальними змінними та значенням, що повертатиметься. Тому працює рекурсія.



```
int main(void)
{ ...
  foo(arg1, arg2, ..., argm);
}

void foo(arg1, arg2, ..., argm)
{ int var1, var2, ..., varn;
  ...
}
```





Стек vs куча

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 void check(int depth) {
5     char c;
6     char *ptr = malloc(1);
7     printf("stack at %p, heap at %p\n", &c, ptr);
8     if (depth <= 0) return;
9     check(depth-1);
10 }
11
12 int main() {
13     check(10);
14     return 0;
15 }
```

```
E:\GDISK\College\{фэютш яёуёрьетрэз Ер рыуюёшёь|ўэ| ьютш|ҫхър 03. 4юї|фэ| ёшяш фрэші ё ьют| ҫ\Code\stack_heap.exe
stack at 000000000062FE17, heap at 0000000000742370
stack at 000000000062FDD7, heap at 0000000000742390
stack at 000000000062FD97, heap at 0000000000746FE0
stack at 000000000062FD57, heap at 0000000000747000
stack at 000000000062FD17, heap at 0000000000747020
stack at 000000000062FCD7, heap at 0000000000747040
stack at 000000000062FC97, heap at 0000000000747060
stack at 000000000062FC57, heap at 0000000000747080
stack at 000000000062FC17, heap at 00000000007470A0
stack at 000000000062FBD7, heap at 00000000007470C0
stack at 000000000062FB97, heap at 00000000007470E0
```

■ Stack

- Дуже швидкий доступ
- Не потрібно явно очищати (de-allocate) змінні
- Простір ефективно управляється процесором, пам'ять не фрагментується
- Містить тільки локальні змінні
- Має обмежений розмір залежно від ОС (зазвичай від 1 до 8 Мб)
- Змінні не можуть змінювати свій розмір

■ Heap

- Доступ до змінних може бути глобальним
- Немає обмежень на розмір пам'яті
- (відносно) повільніший доступ
- Ефективність використання пам'яті не гарантується, пам'ять може фрагментуватись з часом через операції аллокації/вивільнення
- Необхідно управляти пам'яттю (явно виділяти та вивільняти)
- Змінні можуть змінювати розмір за допомогою realloc()

Резюме щодо класів пам'яті (storage class) у мові C

Storage Specifier	Storage	Initial value	Scope	Life
auto	stack	Garbage	Within block	End of block
extern	Data segment	Zero	global Multiple files	Till end of program
static	Data segment	Zero	Within block	Till end of program
register	CPU Register	Garbage	Within block	End of block



ДЯКУЮ ЗА УВАГУ!

Тема доповіді



Тема доповіді: рандомізація адресного простору та її ефективність



Тема доповіді: багатопоточне програмування в мові C