# Организация памяти программы. Стек вызовов и принцип его работы.

## Организация памяти программы на языке Си

Программа на языке С при выполнении использует несколько областей памяти, каждая из которых имеет своё назначение. Основные сегменты памяти:

1. **Сегмент кода (text segment)**
   - Здесь хранится исполняемый машинный код программы.
   - Этот сегмент, как правило, доступен только для чтения (чтобы предотвратить изменение инструкций во время выполнения).
   - Размер фиксируется на этапе компиляции
2. **Сегмент данных (data segment)**
   - **Инициализированные данные (initialized data):** переменные глобального и статического типа, которые были явно инициализированы.
   - **Неинициализированные данные (BSS-сегмент):** глобальные и статические переменные без инициализации. Они автоматически инициализируются нулями при запуске программы.
   - Существует в течение всего времени выполнения программы
3. **Сегмент кучи (heap)**
   - Используется для динамического выделения памяти во время выполнения с помощью функций `malloc`, `calloc`, `realloc` и освобождается через `free`.
   - Управляется вручную программистом.
   - Расширяется вверх (в сторону увеличения адресов).
4. **Стек (stack)**
   - Используется для хранения данных временного характера, таких как:
     - локальные переменные функций,
     - параметры функций,
     - адрес возврата после вызова функции,
     - сохранённые регистры.
   - Работает по принципу **LIFO (Last In, First Out)**.
   - Стек растёт вниз (в сторону уменьшения адресов памяти).

## Стек вызовов (Call Stack)

**Стек вызовов** — это часть стека, используемая для управления функциями во время выполнения программы.

### Как работает стек вызовов:

1. **Вызов функции:**
   - Когда вызывается функция, создаётся **кадр стека (stack frame)**.
   - В этот кадр помещаются:
     - параметры функции,
     - локальные переменные,
     - адрес возврата (указание, куда вернуться после выполнения функции),
     - значение регистра `bp` или `rbp` (указатель базы кадра стека, сохраняется для восстановления контекста).
2. **Выполнение функции:**
   - Функция работает с данными в своём кадре стека.
   - Каждый вызов функции создаёт новый кадр.
3. **Завершение функции:**
   - Когда функция завершает выполнение (встречает `return`), её кадр стека уничтожается.
   - Управление передаётся по сохранённому адресу возврата в предыдущую функцию.
   - Локальные переменные исчезают (их память освобождается автоматически).

Пример:

```
void funcB() {
    int b = 20;
}

void funcA() {
    int a = 10;
    funcB();
}

int main() {
    funcA();
    return 0;
}
```

Порядок работы стека:

- `main()` вызывается первым — создаётся кадр `main`.
- Внутри `main()` вызывается `funcA()` — создаётся новый кадр для `funcA`.
- Затем вызывается `funcB()` — создаётся третий кадр для `funcB`.
- После завершения `funcB()` кадр удаляется, и управление возвращается в `funcA`.
- Аналогично после завершения `funcA()` кадр удаляется, и управление возвращается в `main`.

## Особенности и проблемы, связанные со стеком

- **Ограниченный размер:** стек имеет фиксированный объём, и при превышении возникает **переполнение стека (stack overflow)**. Например, при бесконечной рекурсии.
- **Автоматическое управление памятью:** освобождение памяти стека происходит автоматически при выходе из функции.
- **Безопасность:** перезапись данных в стеке может привести к уязвимостям (например, **stack buffer overflow**), которые часто используются в атаках на программы.