- C++ has features that we can use to create an *alias* for a *constant* or *type*; for example:
  (**NOTE**: For *constant*s, it's conventional to have the aliases in *all-uppercase*.)
  - Creating aliases **`PI`** (π) and **`AVG_BIRTH_RATE`** for *constant*s **`3.1416`** and **`0.1758`**:
    ```
    const double PI = 3.1416;
    const double AVG_BIRTH_RATE = 0.1758;
    ```
  - Creating alias **`ADJ_AMOUNT`** for class-level *constant* **`5`**:
    ```
    static const int ADJ_AMOUNT = 5;
    ```
  - Creating aliases **`life_t`** (type for tracking *amount of life left*) and **`credit_t`** (type for tracking *credit points scored*) for *type* **`unsigned int`**:
    ```
    typedef unsigned int life_t;
    typedef unsigned int credit_t;
    ```
- Motivational factors for creating aliases may include the following:
  (**NOTE**: Examples of the previous section are continued here.)
  - The alias is *more meaningful* and may help in *avoiding usage mix-ups*.
    - Most probably can figure out from the context that **`3.1416`** represents π but will be baffled by **`0.1758`**.
    - An attempt to store data of type **`life_t`** into a **`credit_t`** variable has greater likelihood of *raising a flag* (intuitively) than does an attempt to store data of type **`unsigned int`** into an **`unsigned int`** variable.
  - The alias is *easier and less error-prone to type* (because it's shorter, *etc.*).
    - **`3.1416`** is not too bad but what if there's need to represent π more precisely as **`3.1415926535898`**?
  - **(More Importantly)** The alias *makes code modification easier* (*i.e.*, *improves modifiability*).
    - To modify code so that π is represented more precisely as **`3.1415926535898`** (instead of **`3.1416`**):
      - ***Not using alias***: search for all relevant **`3.1416`** and replace each with **`3.1415926535898`**.
      - ***Using alias***: replace **`3.1416`** with **`3.1415926535898`** in the *alias-creating statement* (assuming the alias is *brought to bear*, *i.e.*, **`PI`** and not **`3.1416`** has been used in every computation involving π).
    - To modify code so that credit points scored is allowed to become negative (*i.e.*, want to use **`int`** for it):
      - ***Not using alias***: search for all relevant **`unsigned int`** and replace each with **`int`**.
      - ***Using alias***: replace **`unsigned int`** with **`int`** in the *alias-creating statement* (assuming the alias is *brought to bear*, *i.e.*, **`credit_t`** and not **`unsigned int`** has been used in every type reference involving credit points scored).
- When developing (in non-**`template`** fashion) a *container class* that one wants to use (in different programs) for manipulating objects of different types, the utility of *type aliasing* (using **`typedef`**) comes in handy as follows:
  - Create a type alias (*e.g.*: **`value_type`**) for the object type the container class is presently for; for example:
    ```
    typedef int value_type; // presently for containers of int objects
    ```
  - Use the type alias (**`value_type`** and not **`int`** for the above example) to refer to the object type thereafter in the class' specification and implementation.
    (**NOTE**: This is *bringing the alias to bear* and is ***essential*** for the utility of type aliasing to materialize.)

  This way, if we want to use the same type of container but for objects of a different type in another program, we only need to appropriately change the *alias-creating statement*; for example, changing it to
  ```
  typedef char value_type; // presently for containers of char objects
  ```
  if **`char`** is the purported type.