Write a Python function: fast_copy(source, count) that:

● Allocates a destination structure large enough to hold count elements.

● Determines how many complete transfer groups of eight elements are required.

● Performs any necessary initial element transfers using structured control flow rather than iteration.

● Completes the remaining transfers in a repetitive block that assigns exactly eight elements per iteration.

● Returns the destination structure containing the copied elements.

Example:

We have one fruit box contains 12 fruits(Source). We want to make another fruit box(Dest). We want copy fruits from source to destination like

Making a machine called fast_copy.

Small Copy first $\longrightarrow$ Big Copy together $\longrightarrow$ Finished

Count -----→ how many want.

I.e. It copies the reminder first(copies the a few items…) then copy many items… done

Why we go for C:

- Python: I ask my robot to copy
- C loop: I copy box by box
- C fast copy: I grab 8 boxes at once
- memcpy: I teleport boxes

Hindu Mythological Story:

In Mahabharatha, once Arjunan asked kannan(Krishna), Even though Dharmar is considered the greatest in righteousness, Karnan is remembered as the greatest in charity. In the Mahabharata, when Arjuna asked Krishna why Karnan was considered superior in charity, Krishna decided to demonstrate it. He gave two mountains of gold to Arjuna and asked him to donate them. Despite his strength and good intentions, Arjuna struggled and could not even donate half of a mountain because the process required effort, deliberation, and careful action. Krishna then gave the same two mountains to Karnan and asked him to donate them. Without hesitation, Karnan instantly donated both mountains to two people. In the same way,

Python represents safety, rules, and careful step-by-step execution, where copying happens one element at a time with many checks, similar to Arjuna’s effortful charity. C’s memcpy, on the other hand, is like Karnan’s charity — it performs the task in one decisive action, moving entire blocks of memory at once without hesitation or overhead. Thus, while Python excels in safety and correctness like Dharmar, raw performance and speed belong to C’s memcpy, just as supreme charity belongs to Karnan.

Story character	Programming
Arjunan	Python loop
Dharmar	Python safety checks
Karnan	C memcpy
Kannan	CPU / Compiler
Mountain of gold	Memory block
Dhaanam	Copy

<u>Simple Loop in C:</u>

```
for(int i=0; i<10; i++)
{
dest[i] = source[i];
}
```

- Copies one by one
- Copies faster

<u>memcpy(dest, src, size);</u>

- No hesitation
- No checking
- No counting
- It copies whole blocks.

PYTHON (fast_copy)
C
VS
LITTLE BY LITTLE
ARJUNA
Copies one element at a time
PYTHON
FOLLOWS RULES
Copies one element at a time Carefully and slowly
EVERYTHING AT ONCE
KARNA
Instantly copies whole block
C KARNA
IGNORES RISK
Instantly copies whole block. No checks, no limits

```
dest[i] == source[i]
dest[i+1] = source [i+1]

dest [i+7] = source [i+7]
...
if rem == 1:
  dest[0] = source [0]
elif rem == 2: dest[0] = -1] - source[1]
```

```
memcpy(dest, src, size);
```

Python Code:

```
def fast_copy(source, count):
    if count <= 0:
        return []
    if len(source) < count:
        raise ValueError("source does not contain enough elements for count")
    dest = [None] * count
    groups = count // 8
    rem = count % 8
    i = 0
    if rem == 0:
        pass
    elif rem == 1:
        dest[0] = source[0]; i = 1
    elif rem == 2:
        dest[0] = source[0]; dest[1] = source[1]; i = 2
    elif rem == 3:
        dest[0] = source[0]; dest[1] = source[1]; dest[2] = source[2]; i = 3
    elif rem == 4:
        dest[0] = source[0]; dest[1] = source[1]; dest[2] = source[2]; dest[3] = source[3]; i = 4
    elif rem == 5:
        dest[0] = source[0]; dest[1] = source[1]; dest[2] = source[2]; dest[3] = source[3]; dest[4]
= source[4]; i = 5
    elif rem == 6:
        dest[0] = source[0]; dest[1] = source[1]; dest[2] = source[2]; dest[3] = source[3]; dest[4]
= source[4]; dest[5] = source[5]; i = 6
    elif rem == 7:
        dest[0] = source[0]; dest[1] = source[1]; dest[2] = source[2]; dest[3] = source[3]; dest[4]
= source[4]; dest[5] = source[5]; dest[6] = source[6]; i = 7
    for _ in range(groups):
        dest[i    ] = source[i    ]
        dest[i + 1] = source[i + 1]
        dest[i + 2] = source[i + 2]
        dest[i + 3] = source[i + 3]
        dest[i + 4] = source[i + 4]
        dest[i + 5] = source[i + 5]
        dest[i + 6] = source[i + 6]
        dest[i + 7] = source[i + 7]
        i += 8
    return dest
```

Submitted by

Ms.S.Saranya

Assistant Professor/CSBS