There are multiple valid ways to use the malloc function with one of them being considered the best practice. Often, the non-best practice way is taught first since it's a bit easier to understand. Below, you will see a demonstration of the two methods, and why one is considered better. The demonstration includes using malloc for both primitive types and structures.

```
typedef struct s1 {
    int data1;
} S1;

typedef struct s2 {
    int data1;
    int data2;
} S2;
```

Create an array of 10 integers and create an S1 structure. Later, change the types.

Non-best practice way

- Typecast the return value of malloc
- Explicitly write in the type in the sizeof operator

```
int* a = (int*)malloc(sizeof(int) * 10);
S1* b = (S1*)malloc(sizeof(S1));

// You decide to change "a' to an array of doubles, and "b" to an S2 structure
double* a = (int*)malloc(sizeof(int) * 10);          // invalid
S2* b = (S1*)malloc(sizeof(S1));                     // invalid

double* a = (double*)malloc(sizeof(double) * 10);    // valid after additional changes
S2* b = (S2*)malloc(sizeof(S2));                     // valid after additional changes
```

Best practice way

- Don't typecast the return value of malloc
- Don't explicitly write in the type in the sizeof operator

```
int* a = malloc(sizeof(*a) * 10);
S1* b = malloc(sizeof(*b));

// You decide to change "a' to an array of doubles, and "b" to an S2 structure
double* a = malloc(sizeof(*a) * 10);          // valid without additional changes
S2* b = malloc(sizeof(*b));                   // valid without additional changes
```