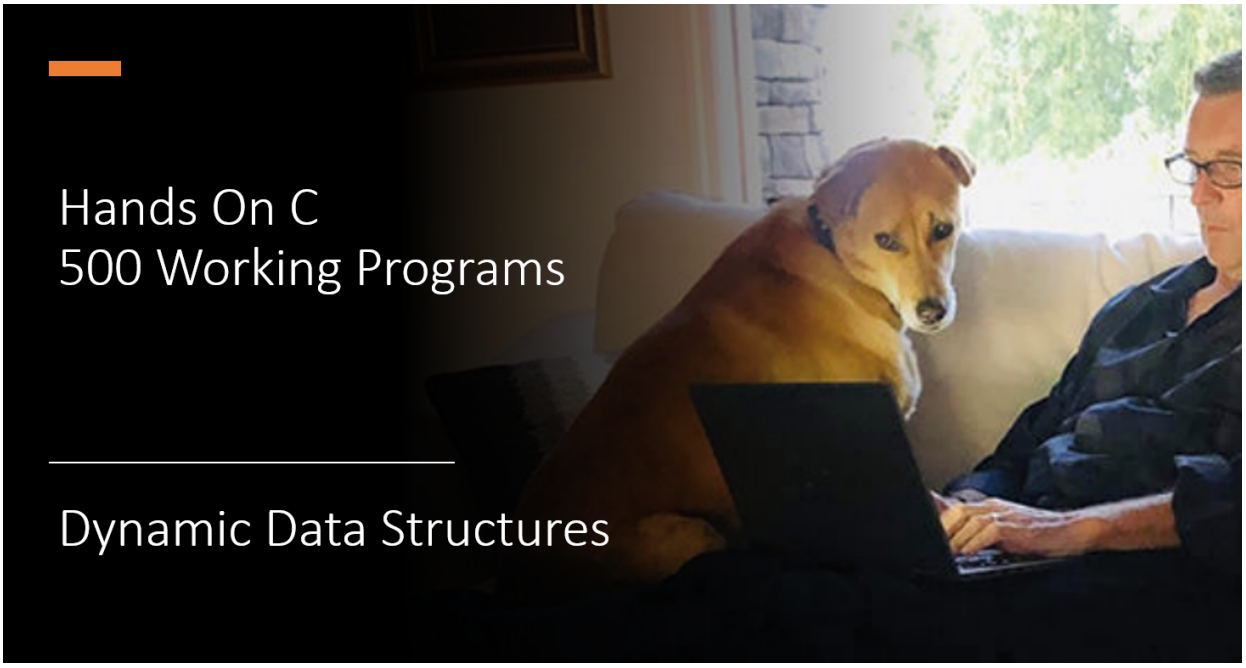


Allocating Dynamic Memory with malloc

In [1]:
```
#include <stdio.h>
#include <stdlib.h>

void main(void)
 {
   char *string;
   int *int_values;
   float *float_values;

   if ((string = (char *) malloc(512)))
     {
        printf("Successfully allocated a 512 byte string\n");
        int i;
        char letter;

        for (i = 0, letter = 'A'; letter <= 'Z'; letter++, i++)
          string[i] = letter;

        string[i] = '\0';

        printf("%s\n", string);
        free(string);
     }
   else
     printf("Error allocating string\n");

   if ((int_values = (int *) malloc(1000 * sizeof(int))) != NULL)
     {
       printf("Successfully allocated int_values[1000]\n");
       free(int_values);
     }
   else
     printf("Error allocating int_values[1000]\n");

   if ((float_values = (float *) malloc(25 * sizeof(float))) != NULL)
     {
       printf("Successfully allocated float_values[25]\n");
       free(float_values);
     }
   else
     printf("Error allocating float_values[25]\n");
 }
```

```
Successfully allocated a 512 byte string
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Successfully allocated int_values[1000]
Successfully allocated float_values[25]
```

Releasing Memory No Longer Needed

In [2]:
```
#include <stdio.h>
#include <stdlib.h>

int main(void)
 {
   char *string;

   if ((string = (char *) malloc(512)) == NULL)
     printf("Error allocating string\n");
   else
   {
     int i;
     char letter;

     for (i = 0, letter = 'A'; letter <= 'Z'; letter++, i++)
       string[i] = letter;

     string[i] = '\0';

     printf("%s\n", string);

     free(string);
   }
 }
```

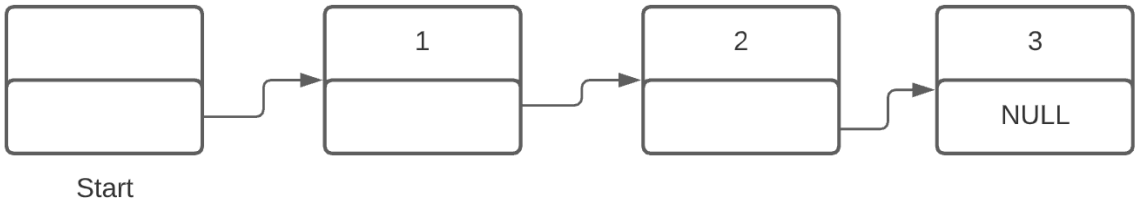
```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Allocating Dynamic Memory with calloc

In [3]:
```
#include <stdio.h>
#include <stdlib.h>

void main(void)
 {
   char *string;
   int *int_values;
   float *float_values;

   if ((string = (char *) calloc(512, sizeof(char))))
     printf("Successfully allocated a 512 byte string\n");
   else
     printf("Error allocating string\n");

   if ((int_values = (int *) calloc(1000, sizeof(int))) != NULL)
     printf("Successfully allocated int_values[1000]\n");
   else
     printf("Error allocating int_values[100]\n");

   if ((float_values = (float *) calloc(25, sizeof(float))) != NULL)
     printf("Successfully allocated float_values[25]\n");
   else
     printf("Error allocating float_values[25]\n");

    free(string);
    free(int_values);
    free(float_values);
 }
```

```
Successfully allocated a 512 byte string
Successfully allocated int_values[1000]
Successfully allocated float_values[25]
```

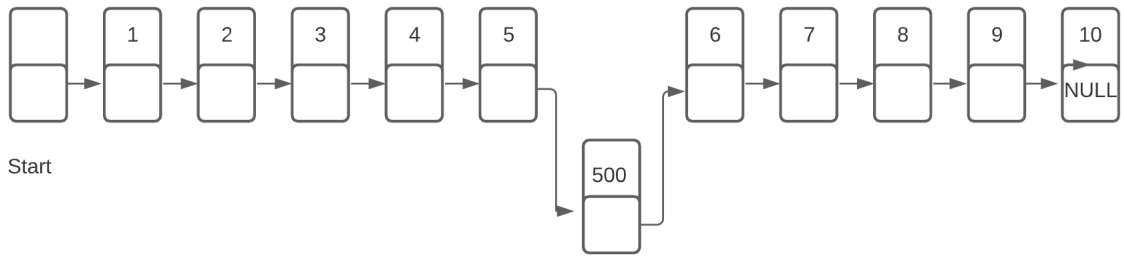
Creating a Singly-Linked List



In [4]:
```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int value;
    struct Node *next;
} ;

int main(void)
{
    struct Node *start = (struct Node *) calloc(1, sizeof(struct Node));
    start->next = NULL;
    struct Node *current_node = start;

    for (int i = 1; i < 10; ++i)
    {
      current_node->next = (struct Node *) calloc(1, sizeof(struct Node));
      current_node = current_node->next;
      current_node->next = NULL;
      current_node->value = i;
    }

    current_node = start->next;
    while (current_node)
    {
        printf("%d ", current_node->value);
        current_node = current_node->next;
    }
}
```

```
1 2 3 4 5 6 7 8 9
```

Inserting a Node into a Singly-Linked List



In [5]:
```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int value;
    struct Node *next;
} ;

int main(void)
{
    struct Node *start = (struct Node *) calloc(1, sizeof(struct Node));
    start->next = NULL;
    struct Node *current_node = start;

    for (int i = 1; i < 10; ++i)
    {
      current_node->next = (struct Node *) calloc(1, sizeof(struct Node));
      current_node = current_node->next;
      current_node->next = NULL;
      current_node->value = i;
    }

    // insert value 500 after node with value 5
    current_node = start->next;
    while (current_node->value != 5)
        current_node = current_node->next;

    struct Node *new_node = (struct Node *) calloc(1, sizeof(struct Node));
    new_node->next = current_node->next;
    new_node->value = 500;
    current_node->next = new_node;

    current_node = start->next;
    while (current_node)
    {
        printf("%d ", current_node->value);
        current_node = current_node->next;
    }

}
```

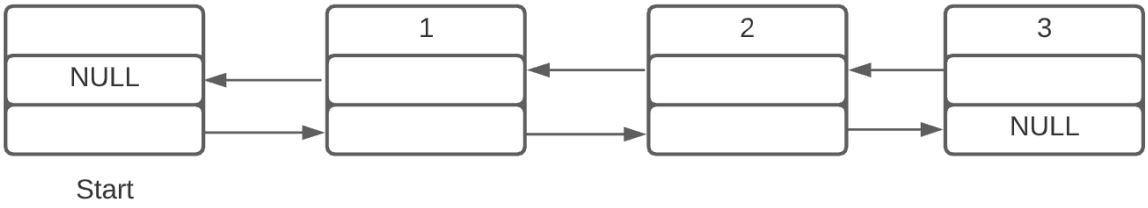
```
1 2 3 4 5 500 6 7 8 9
```

Deleting a Node from a Singly-Linked List

In [6]:
```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int value;
    struct Node *next;
} ;

int main(void)
{
    struct Node *start = (struct Node *) calloc(1, sizeof(struct Node));
    start->next = NULL;
    struct Node *current_node = start;

    for (int i = 1; i < 10; ++i)
    {
      current_node->next = (struct Node *) calloc(1, sizeof(struct Node));
      current_node = current_node->next;
      current_node->next = NULL;
      current_node->value = i;
    }

    // Delete node with value 5
    current_node = start->next;
    struct Node *previous_node = start;

    while (current_node->value != 5)
    {
        previous_node = current_node;
        current_node = current_node->next;
    }

    previous_node->next = current_node->next;
    free(current_node);

    current_node = start->next;
    while (current_node)
    {
        printf("%d ", current_node->value);
        current_node = current_node->next;
    }

}
```

```
1 2 3 4 6 7 8 9
```

Using a Doubly-Linked List



In [7]:
```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int value;
    struct Node *next;
    struct Node *previous;
} ;

int main(void)
{
    struct Node *start = (struct Node *) calloc(1, sizeof(struct Node));
    start->next = NULL;
    start->previous = NULL;
    struct Node *current_node = start;

    for (int i = 1; i < 10; ++i)
    {
      current_node->next = (struct Node *) calloc(1, sizeof(struct Node));
      current_node->next->previous = current_node;
      current_node = current_node->next;
      current_node->next = NULL;
      current_node->value = i;
    }

    // insert value 500 after node with value 5
    current_node = start->next;
    while (current_node->value != 5)
        current_node = current_node->next;

    struct Node *new_node = (struct Node *) calloc(1, sizeof(struct Node));
    new_node->next = current_node->next;
    new_node->previous = current_node;
    new_node->value = 500;
    current_node->next = new_node;

    current_node = start->next;
    while (current_node)
    {
        printf("%d ", current_node->value);
        current_node = current_node->next;
    }
}
```

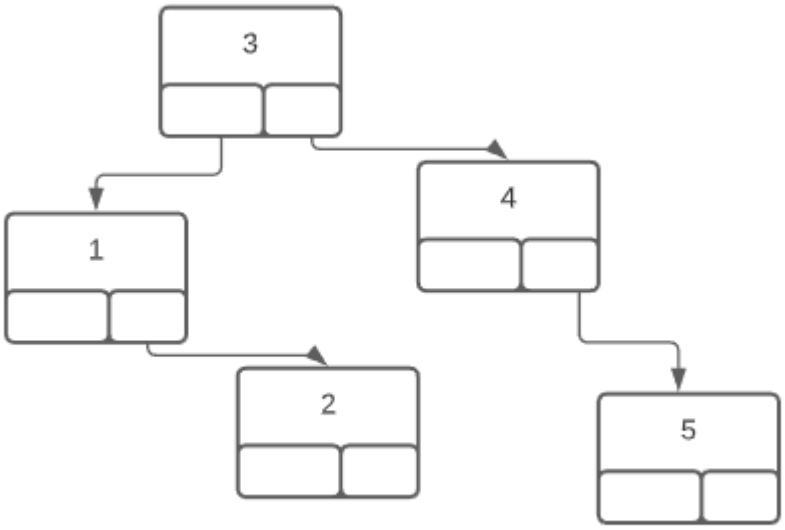
```
1 2 3 4 5 500 6 7 8 9
```

Deleting a Value from a Doubly-Linked List

In [8]:
```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int value;
    struct Node *next;
    struct Node *previous;
} ;

int main(void)
{
    struct Node *start = (struct Node *) calloc(1, sizeof(struct Node));
    start->next = NULL;
    start->previous = NULL;
    struct Node *current_node = start;

    for (int i = 1; i < 10; ++i)
    {
      current_node->next = (struct Node *) calloc(1, sizeof(struct Node));
      current_node->next->previous = current_node;
      current_node = current_node->next;
      current_node->next = NULL;
      current_node->value = i;
    }

    // delete the value 5
    current_node = start->next;
    while (current_node->value != 5)
        current_node = current_node->next;

    if (current_node->next != NULL)
      current_node->next->previous = current_node->previous;
    if (current_node->previous != NULL )
      current_node->previous->next = current_node->next;
    free(current_node);

    current_node = start->next;
    while (current_node)
    {
        printf("%d ", current_node->value);
        current_node = current_node->next;
    }
}
```

```
1 2 3 4 6 7 8 9
```

Creating a Binary Tree



In [9]:
```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int value;
    struct Node *right;
    struct Node *left;
} ;

void insert_value(struct Node **node, int value)
{
    struct Node *new_node = NULL;

    if (*node == NULL)
    {
        new_node =  (struct Node *) calloc(1, sizeof(struct Node));
        new_node->value = value;
        new_node->left = NULL;
        new_node->right = NULL;
        *node = new_node;
        return;
    }

    if (value < (*node)->value)
      insert_value(&(*node)->left, value);
    else
      insert_value(&(*node)->right, value);
}

void display_tree(struct Node *node)
{
   if (node)
   {
       display_tree(node->left);
       printf("%d\n", node->value);
       display_tree(node->right);
   }
}

int main(void)
{
    struct Node *root = NULL;

    insert_value(&root, 3);
    insert_value(&root, 1);
    insert_value(&root, 4);
    insert_value(&root, 2);
    insert_value(&root, 5);

    display_tree(root);
}
```

```
1
2
3
```

```
4
5
```

What To Learn Next

```
Congratulations on learning C programming. I always say learning your first
language is time consuming, learning the second language takes half that amount
of time and learning your third takes half of that.

You now have a great foundation to learn and more importantly appreciate C++
programming. I recommend that you should take my C++ course followed by my C#
course.

Good luck with your programming endeavors.
```