# PROGRAMMING

CT103

Week 6b

# Lecture Content

- Last lecture (Week 6a):
  - Length of a string
  - Insert data in a string
  - String functions
  - Example C program
- Today's lecture (Week 6b):
  - Constants
  - Puts
  - Gets
  - Sscanf_s
  - Example C program

# Bank Holiday

- Please note the **bank holiday** on 30 October 2023.
- There will be **no lectures** on **Monday 30 October 2023**.
- Labs with new lab assignment will still take place on Tuesday 31 October 2023.

# CONSTANTS

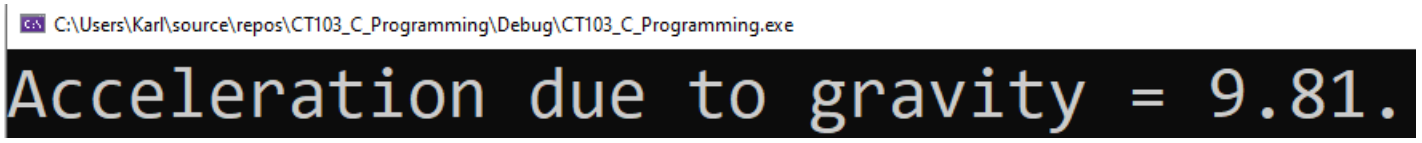
# Constants

- We talked a lot about variables already and know what they are.
- E.g. `int age = 62;`
- We can also create **constants**.
- **Constants** refer to fixed values that the program cannot change during its execution. These are also often called literals.

# Constants

- You would create a constant in C as follows:

```
const float gravity = 9.81;
printf("Acceleration due to gravity = %0.2f.\n\n", gravity);
```



# Constants

- As the name suggests, you cannot change the value of a constant.

```
const float gravity = 9.81;
printf("Acceleration due to gravity = %0.2f.\n\n", gravity);
gravity = gravity + 1;
```

Can’t do this!

# #Define in C

- You can also declare constants using #define.
- These need to be created outside of main.
- Using #define creates what is called a **macro**.

# Macro

- What is a **macro**?

- A macro is a fragment of code which has been given a name. Whenever the name is used it is replaced by the contents of the macro.

- There are two types of macros:
  - Object like macros.
  - Function like macros (we will ignore these for now).

# #Define in C

- What does #define look like in C?

- You can create an object like macro in C using the following:

```
#define PI 3.14
```

# #Define in C Example

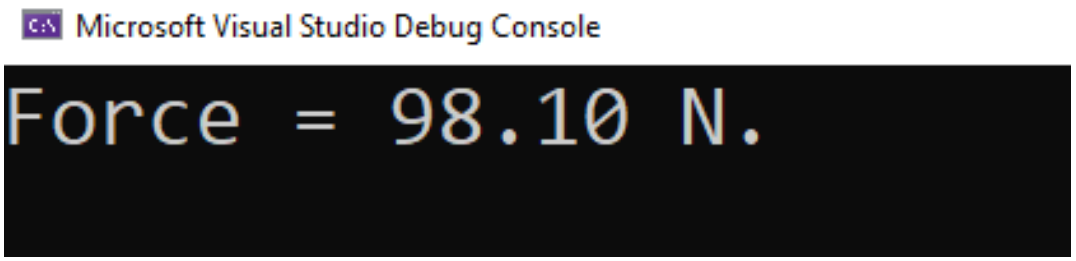
```c
#include <stdio.h>
#include "string.h"

#define g 9.81
void main() {
    float mass = 10;
    float F;
    F = mass * g;
    printf("Force = %0.2f N.\n",F);
}
```

# #Define in C Example

```c
#include <stdio.h>
#include "string.h"

#define g 9.81
void main() {
    float mass = 10;
    float F;
    F = mass * g;
    printf("Force = %0.2f N.\n",F);
}
```



# PUTS

# Puts

- What is puts?
- Puts is a function for printing strings to the screen.
- You need to include the <stdio.h> library to call puts.

# Why Puts Over Printf?

- Puts is simple.
- Puts is less expensive than printf.
- Puts is more secure.
  - E.g. if there is a “%” in your string, printf may behave strangely, puts won’t.

# Puts in C

- You would use Puts as follows in C:

```c
#include <stdio.h>
void main()
{
    char myString[] = "Here is my string.";
    puts(myString);
}
```

# Puts in C

- This gives the following output:

```c
#include <stdio.h>
void main()
{
    char myString[] = "Here is my string.";
    puts(myString);
}
```

# GETS

# Gets

- What is gets?
- Gets is a function for reading input from the keyboard.
- You also need to include the <stdio.h> library to call gets.

# Gets vs Scanf?

- Gets is only used for strings.
- Gets will not stop reading characters, even with whitespace, until it reaches a newline.
- Gets is easy to use.

# Gets in C

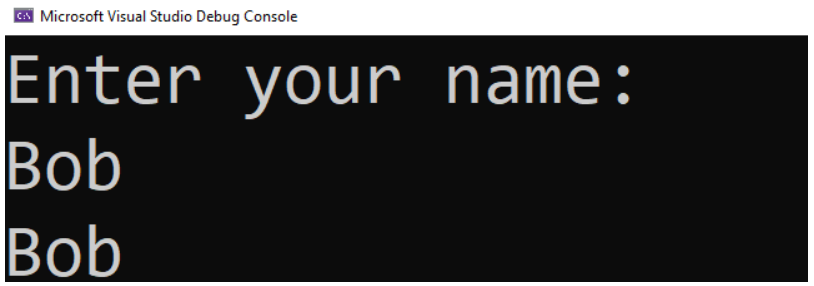
- You would use Gets as follows in C:

```c
#include <stdio.h>
void main()
{
    char myString[10] = "Temp";
    puts("Enter your name:");
    gets(myString);
    puts(myString);
}
```

# Gets in C

- This gives the following output:

```
#include <stdio.h>
void main()
{
    char myString[10] = "Temp";
    puts("Enter your name:");
    gets(myString);
    puts(myString);
}
```



# USING PUTS AND GETS

# Puts and Gets

```c
#include <stdio.h>
#include "string.h"

void main()
{
    char firstName[15], secondName[15];
    char fullName[35] = "";

    puts("what is your first name?: ");
    gets(firstName);
    puts("What is your surname?: ");
    gets(secondName);

    strcat_s(fullName, 35 , firstName);
    strcat_s(fullName, 35, " ");
    strcat_s(fullName, 35, secondName);

    puts("\nYour full name is:");
    puts(fullName);
}
```

# Puts and Gets

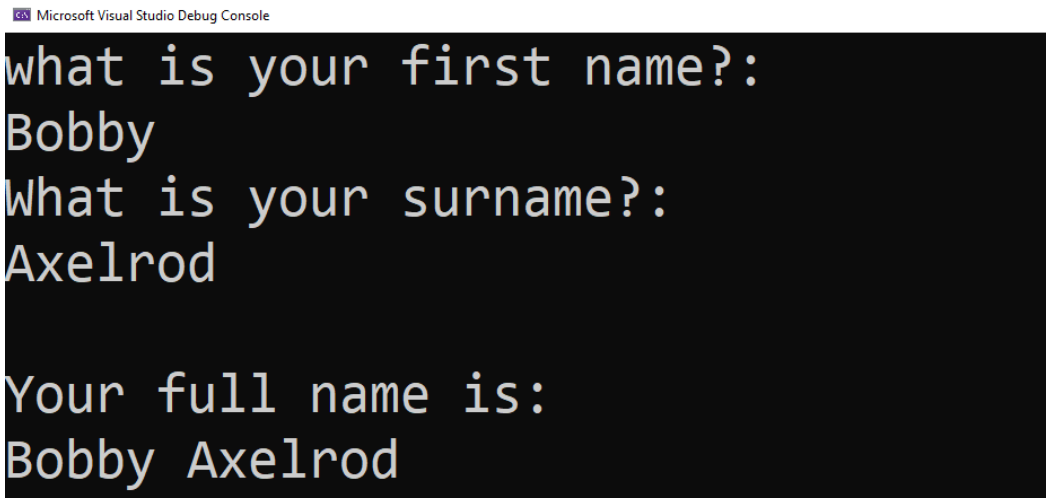
```c
#include <stdio.h>
#include "string.h"

void main()
{
    char firstName[15], secondName[15];
    char fullName[35] = "";

    puts("what is your first name?: ");
    gets(firstName);
    puts("What is your surname?: ");
    gets(secondName);

    strcat_s(fullName, 35 , firstName);
    strcat_s(fullName, 35, " ");
    strcat_s(fullName, 35, secondName);

    puts("\nYour full name is:");
    puts(fullName);
}
```



# SSCANF_S

# Sscanf_s

- You may get strings from anywhere, such as files or databases.
- These may then need to be parsed to extract data.
- What does **parse** mean?
  - Transforming a steam of text into some other form of information.

# Sscanf_s

- Sscanf_s is useful for **scanning formatted data from a string**.

- **IF** you know the exact format of the string (and it won't be changed), you can read it the same way as you would read the console input using scanf.

# Sscanf_s Example

```c
#include <stdio.h>
#include "string.h"

void main()
{
    char string[100]="hi";
    char firstName[20], surname[20];
    char fullName[40];
    int dd, mm, yyyy;

    puts("Enter 'firstName' 'Surname' 'dd/mm/yyyy'");

    gets(string);

    sscanf_s(string, "%s %s %d/%d/%d", firstName, 20, surname, 20, &dd, &mm, &yyyy);
    strcpy_s(fullName, 40, firstName);
    strcat_s(fullName, 40, " ");
    strcat_s(fullName, 40, surname);

    printf("\n%s was born on %d-%d-%d\n\n", fullName, dd, mm, yyyy);
}
```

# Sscanf_s Example

```c
#include <stdio.h>
#include "string.h"

void main()
{
    char string[100]="hi";
    char firstName[20], surname[20];
    char fullName[40];
    int dd, mm, yyyy;

    puts("Enter 'firstName' 'Surname' 'dd/mm/yyyy'");

    gets(string);

    sscanf_s(string, "%s %s %d/%d/%d", firstName, 20, surname, 20, &dd, &mm, &yyyy);
    strcpy_s(fullName, 40, firstName);
    strcat_s(fullName, 40, " ");
    strcat_s(fullName, 40, surname);

    printf("\n%s was born on %d-%d-%d\n\n", fullName, dd, mm, yyyy);
}
```

Microsoft Visual Studio Debug Console

```
Enter 'firstName' 'Surname' 'dd/mm/yyyy'
Bobby Axelrod 1/1/1960


Bobby Axelrod was born on 1-1-1960
```

# EXAMPLE PROBLEMS

# Physics Energy Calculator

- You are writing software to calculate physics equations. Write a program that:
  - Defines acceleration due to gravity as a constant.
  - Reads in 4 measurements of the following:
    - Mass (m), velocity (v) and height (h).
  - Calculate the kinetic (KE) and potential (PE) energy of the 4 objects.

  - **Note**:
    - KE = 0.5 m $v^2$
    - PE = m g h
    - g = 9.81 $m/s^2$

# Physics Energy Calculator

- Go to C program solution.

# Physics Energy Calculator

```c
#include <stdio.h>
#include "string.h"

#define g 9.81
void main() {
    float mass;
    float v;
    float h;
    float F;
    float kEnergy;
    float pEnergy;
    char dataIn[100] = "temp";

    for (int i = 0; i < 4; i++) {
        puts("Enter the mass, velocity and height as: 'mass' 'vel' 'height'");
        gets(dataIn);
        sscanf_s(dataIn, "%f %f %f", &mass, &v, &h);
        kEnergy = 0.5 * mass * v * v;
        pEnergy = mass * g * h;
        printf("Kinetic Energy = %0.2f J.\n", kEnergy);
        printf("Potential Energy = %0.2f J.\n", pEnergy);
    }
}
```

# Physics Energy Calculator

- C Program Output: