Week 8 Lecture 2

Type Conversion;

Enumerations;

Macros

Type Conversion

Type Conversion

- C converts between basic types:
  - Implicitly
  - Explicitly

Implicit Conversion

- The compiler implicitly converts when provided an unexpected type.
  - Conversion during assignments:

```
char c='a';
int i;
i=c;    /* i  is assigned the ASCII code of ’a’ */
```

- The compiler implicitly converts if two operands of a binary operator are different types:

```
int i=5 , j=1;
float x=1.0 , y;
y = x / i;          /* y = 1.0 /  5.0  */
y = j / i;          /* y = 1 / 5  so  y = 0 */
```

TYPE CONFLICTS IN C

- An arithmetic operation between an integer and integer yields an integer result.
- An operation between a real and real yields a real result.
- An operation between an integer and real yields a real result.
  - The integer is promoted to a real, before the operation.

.

Operation	Result	Operation	Result
5/2	2	2/5	0
5.0/2	2.5	2.0/5	0.4
5/2.0	2.5	2.0/5	0.4
5.0/2.0	2.5	2.0/5.0	0.4

Explicit Conversion

Explicit conversion uses the cast operator.

- Example :

```
int x=10;
float y, z=3.14;
y=(float) x;     /* y=10.0 */
x=(int) z;       /* x=3    */
```

- Example :

```
int i=5 , j=1;
float x=1.0 , y;
y = (float) j / i;  /* y = 1.0 / 5  explicit*/
/* The cast operator has a higher precedence */
```

Style

- Always use explicit conversion
  - It indicates the intention to do the conversion
  - Relaxed when the conversion is expected such as float division

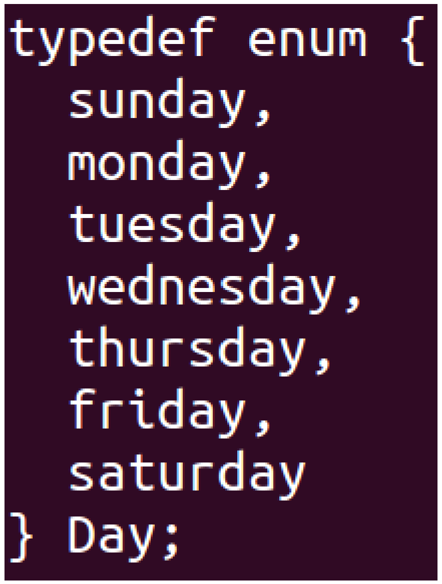
Enumeration

Enumeration

- The enumeration allows you to define a set of similar elements.
  - E.g., Sunday, Monday, Tuesday, Wednesday, …
- These variables represent exactly one of the elements.
- Enumerations can add to the clarity of your program.

Example

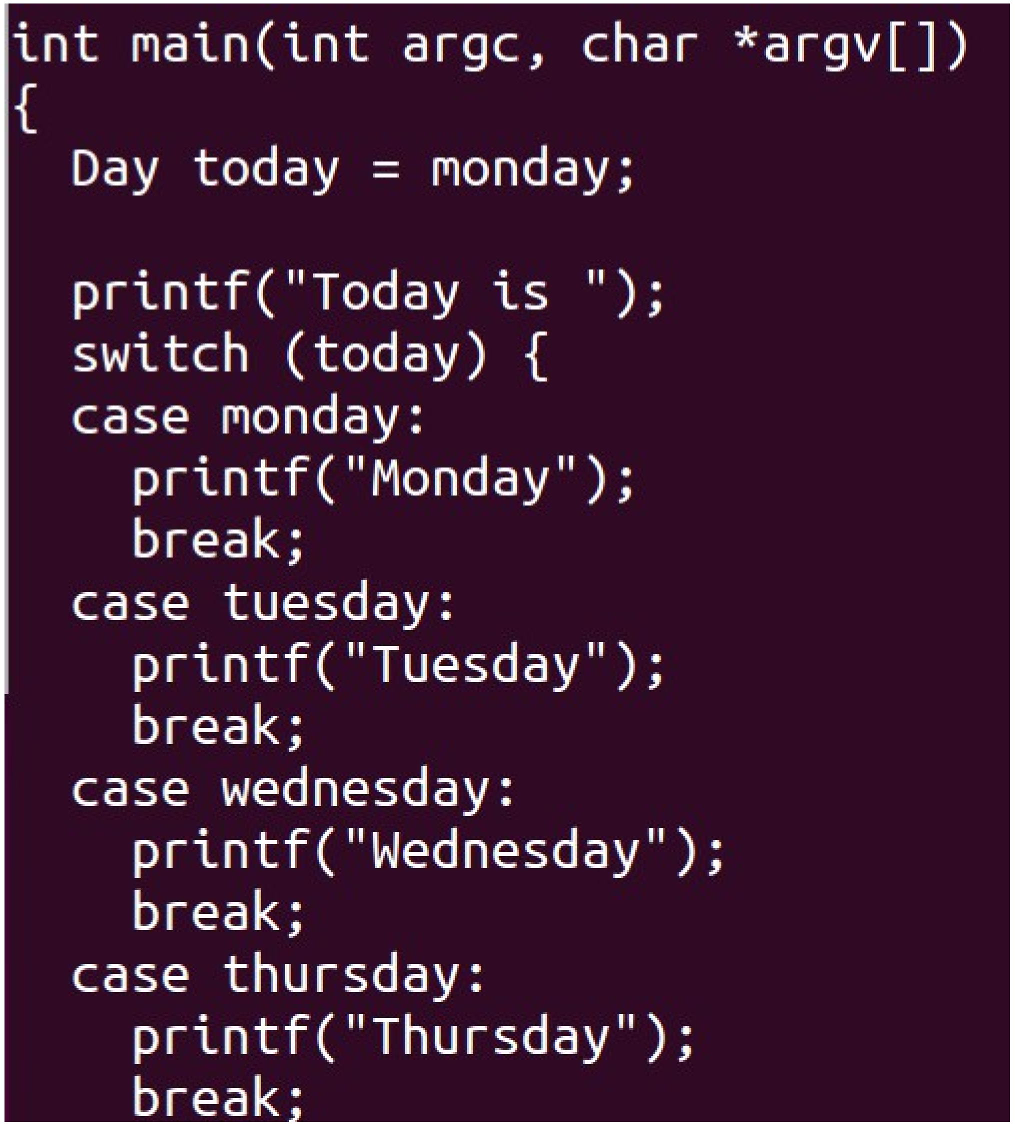
- Enumerations are often used as types
  - E.g. define a variable that represents a day of the week



```
typedef enum {
  sunday,
  monday,
  tuesday,
  wednesday,
  thursday,
  friday,
  saturday
} Day;
```

Use

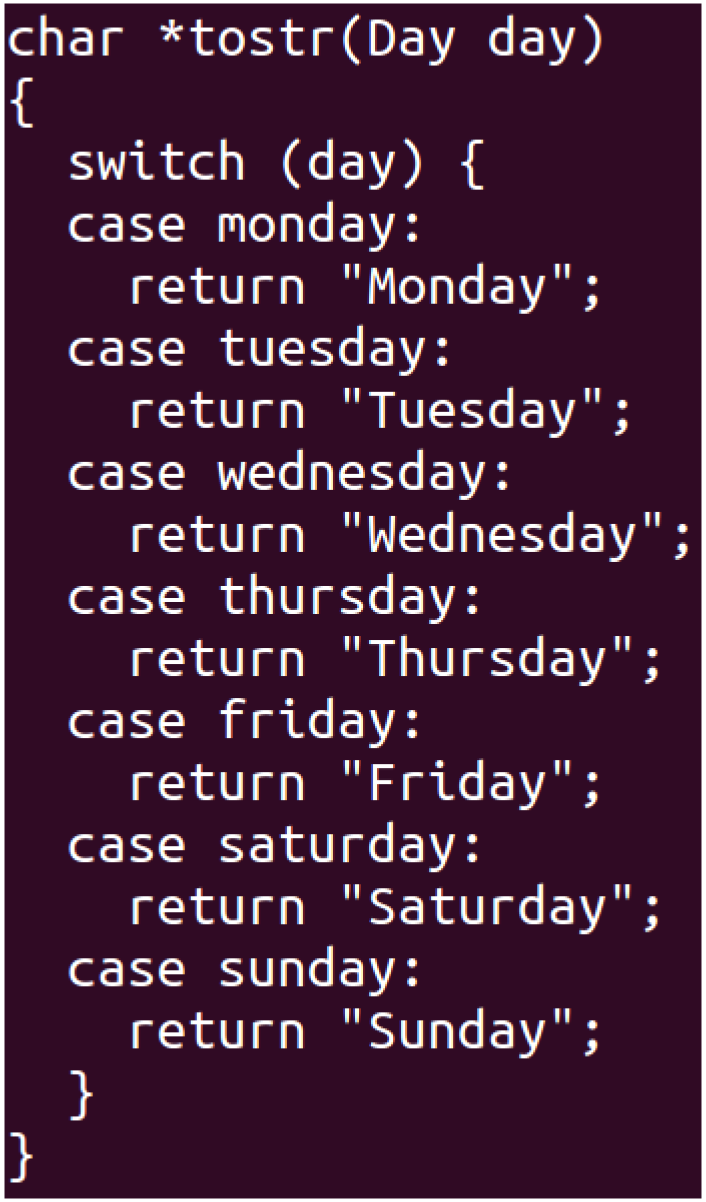
- Create a variable
- Assign a value
- Use the variable



```
int main(int argc, char *argv[])
{
  Day today = monday;

  printf("Today is ");
  switch (today) {
  case monday:
    printf("Monday");
    break;
  case tuesday:
    printf("Tuesday");
    break;
  case wednesday:
    printf("Wednesday");
    break;
  case thursday:
    printf("Thursday");
    break;
```

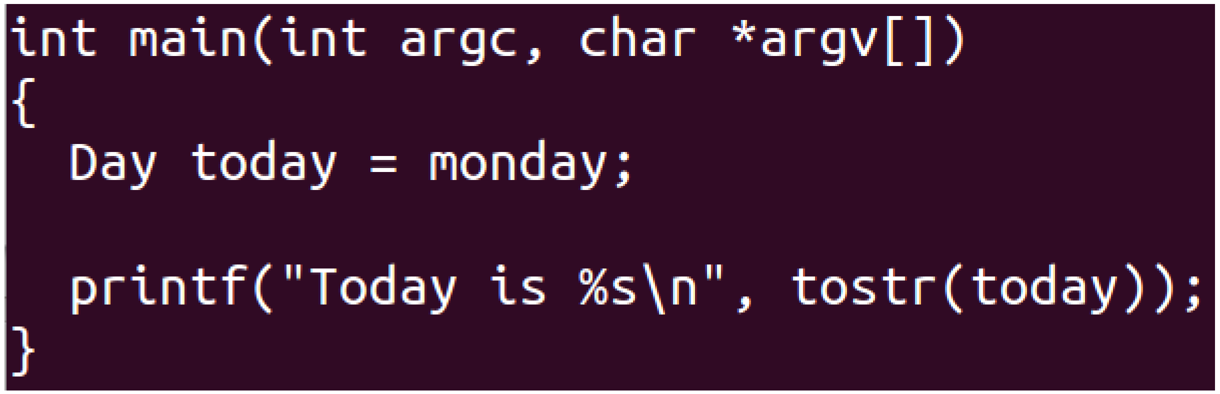
tostr

- Enumeration need string definition.
  - The value of the variable is a sequence of bits.
    - Usually interpretable as an int.
      - monday: 0
      - tuesday: 1
      - ...



```
char *tostr(Day day)
{
  switch (day) {
  case monday:
    return "Monday";
  case tuesday:
    return "Tuesday";
  case wednesday:
    return "Wednesday";
  case thursday:
    return "Thursday";
  case friday:
    return "Friday";
  case saturday:
    return "Saturday";
  case sunday:
    return "Sunday";
  }
}
```

Printing



```
int main(int argc, char *argv[])
{
  Day today = monday;

  printf("Today is %s\n", tostr(today));
}
```

- Now we can print the enumeration

How enumerations work

- An enumeration is implemented as an int
- Constants are declared for each of the possible values of the int
- Only those constants can be assigned to the variable.
- E.g.
  - enum boolean {false, true} assigned false one value and true another
    - Usually starting with 0
    - You may assign values e.g. enum boolean {false = 0, true}

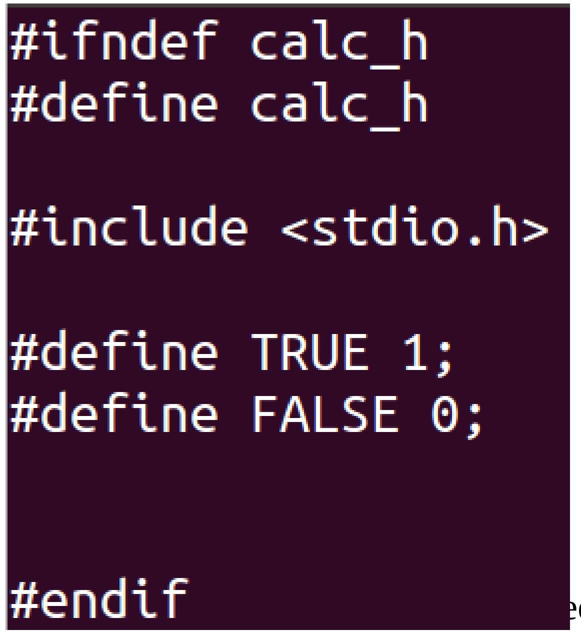
Macros

Macros

- Macros replace text with other text before compilation.
- Macros provide the powerful ability to change the syntax of the language.
  - But they make the resulting programs harder to read.
- Macros should be used with care

Legitimate use of Macros

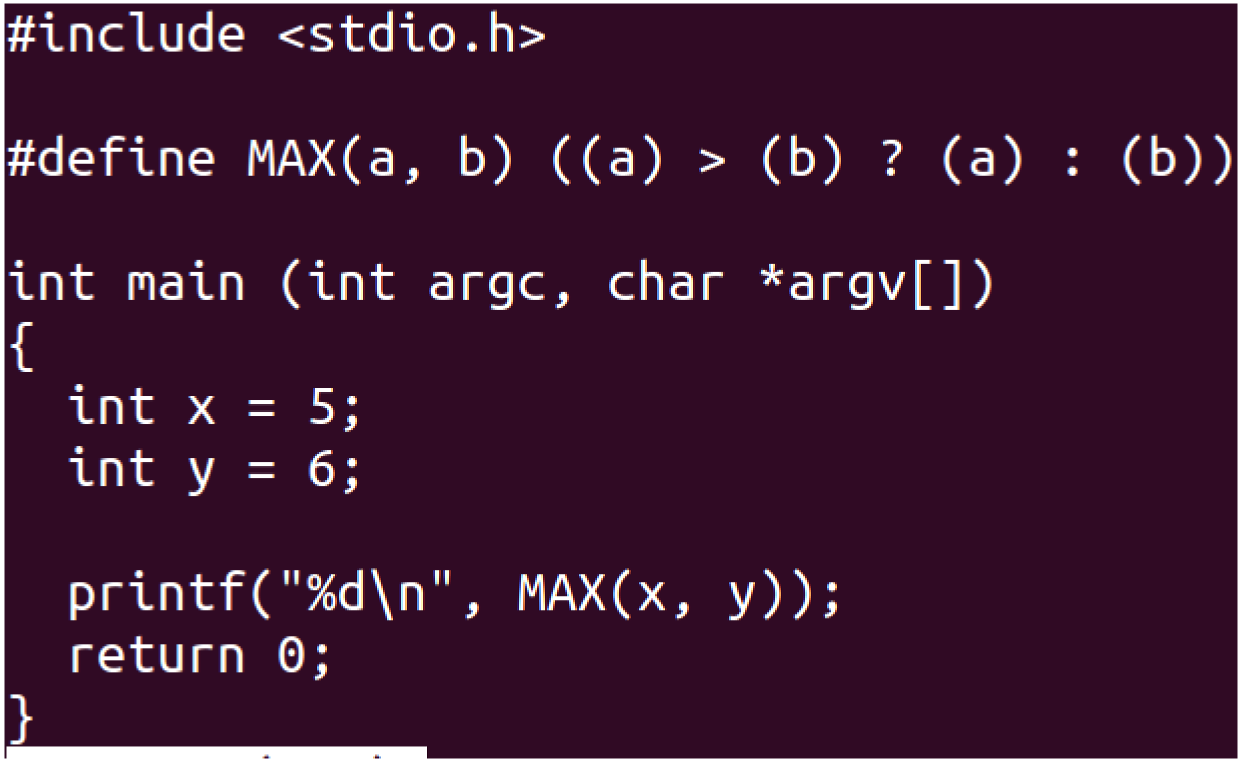
- Define a constant
  - e.g. #define PI 1.416
  - Type constants in UPPERCASE
- Protect .h files from multiple inclusion
  - e.g.



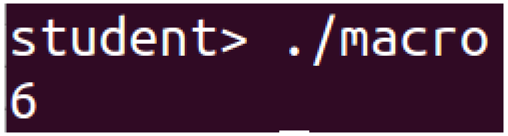
```
#ifndef calc_h
#define calc_h

#include <stdio.h>

#define TRUE 1;
#define FALSE 0;


#endif
```

Advanced Macros

- Macros can act like functions
- e.g.



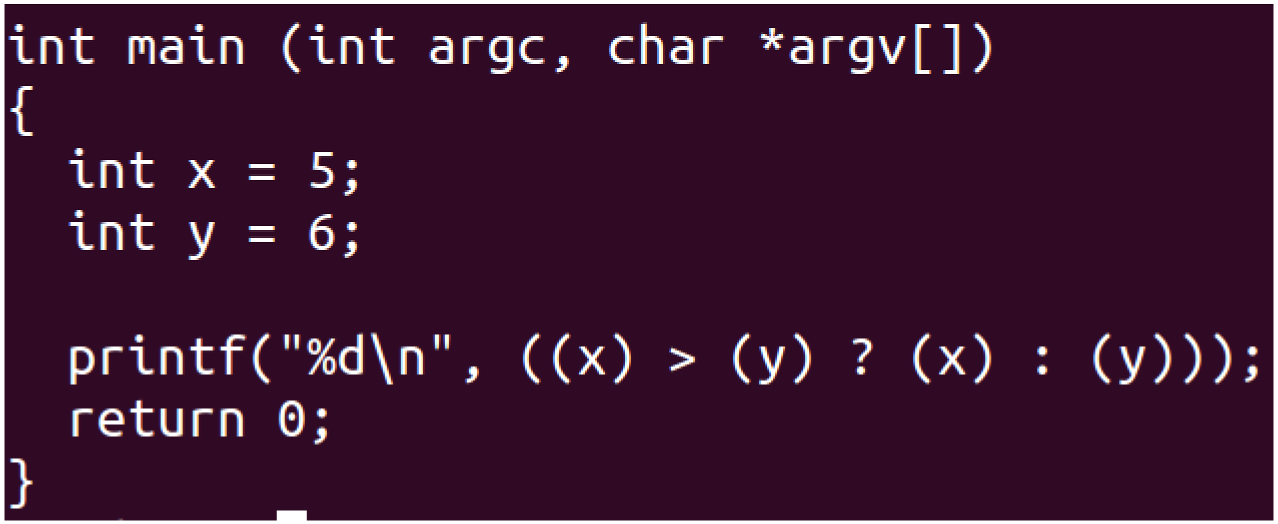
```
#include <stdio.h>

#define MAX(a, b) ((a) > (b) ? (a) : (b))

int main (int argc, char *argv[])
{
  int x = 5;
  int y = 6;

  printf("%d\n", MAX(x, y));
  return 0;
}
```



```
student> ./macro
6
```

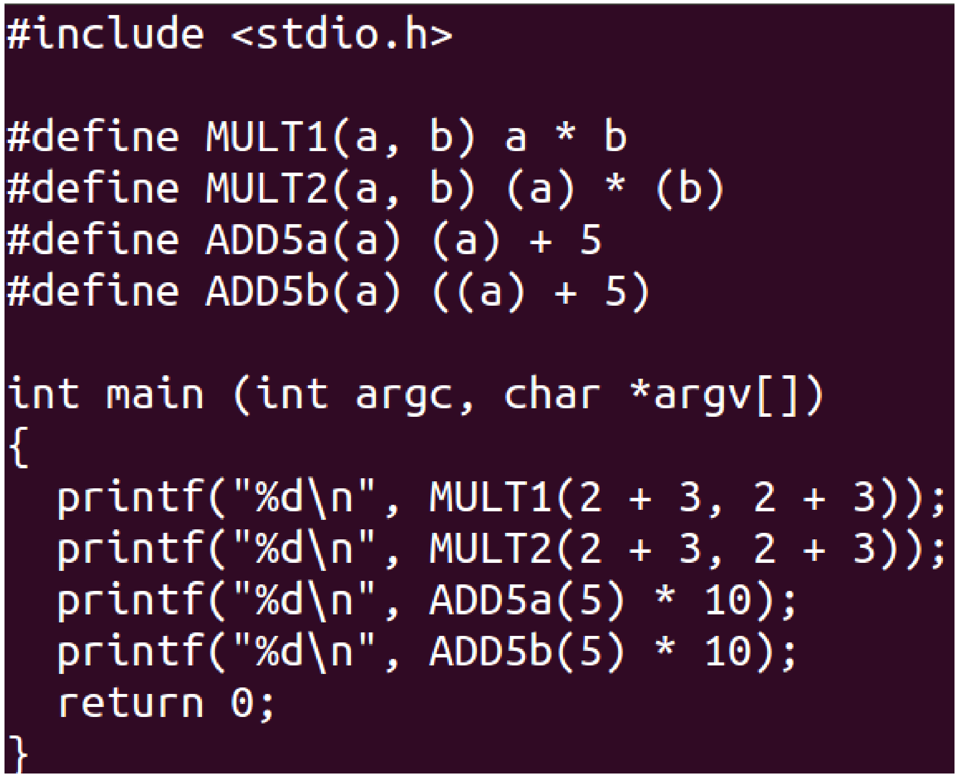
Macros are not Functions

- Translated into code using cpp
- e.g. cpp macro.c:

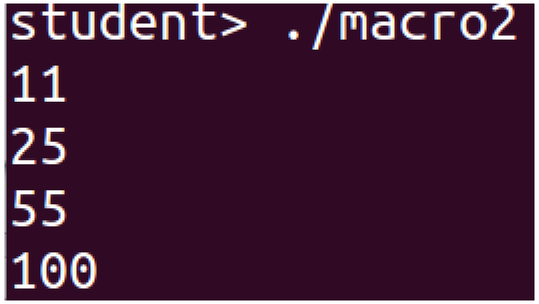


```
int main (int argc, char *argv[])
{
  int x = 5;
  int y = 6;

  printf("%d\n", ((x) > (y) ? (x) : (y)));
  return 0;
}
```

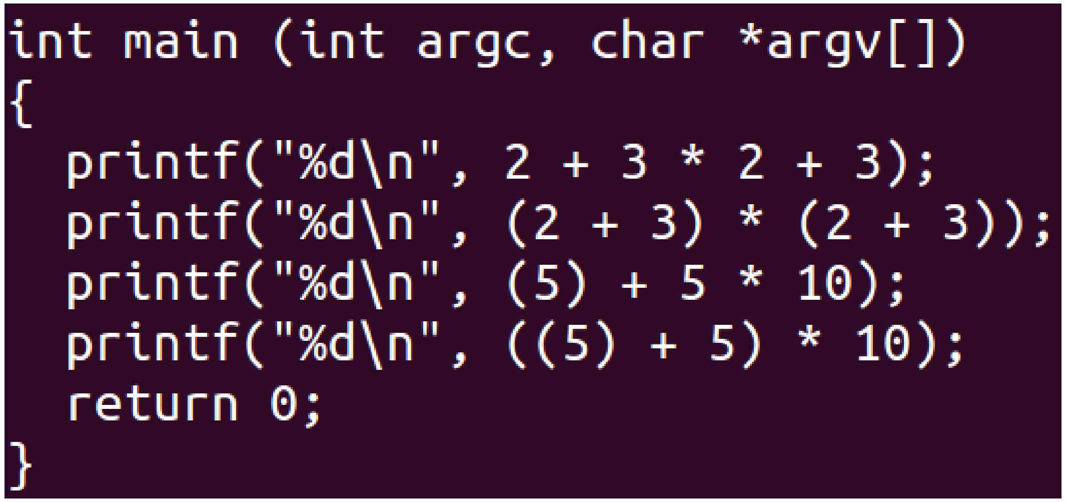
Why all the parentheses



```
#include <stdio.h>

#define MULT1(a, b) a * b
#define MULT2(a, b) (a) * (b)
#define ADD5a(a) (a) + 5
#define ADD5b(a) ((a) + 5)

int main (int argc, char *argv[])
{
  printf("%d\n", MULT1(2 + 3, 2 + 3));
  printf("%d\n", MULT2(2 + 3, 2 + 3));
  printf("%d\n", ADD5a(5) * 10);
  printf("%d\n", ADD5b(5) * 10);
  return 0;
}
```



```
student> ./macro2
11
25
55
100
```



```
int main (int argc, char *argv[])
{
  printf("%d\n", 2 + 3 * 2 + 3);
  printf("%d\n", (2 + 3) * (2 + 3));
  printf("%d\n", (5) + 5 * 10);
  printf("%d\n", ((5) + 5) * 10);
  return 0;
}
```

Even more Advanced Macros

- The macro facility is complicated.
  - You will need to explore further if you end up programming in C.