Data type

Arijit Mondal
Dept. of Computer Science & Engineering
Indian Institute of Technology Patna
`arijit@iitp.ac.in`

More data types

- Some of the basic data types can be augmented by using certain data type qualifiers:
    - `short`
    - `long`
    - `signed`
    - `unsigned`
- Typical examples:

  `short int` (usually 2 bytes)
  `long int` (usually 4 bytes)
  `unsigned int` (usually 4 bytes, but no way to store + or -)

Typical sizes

Data type	bit size	Minimum value	Maximum value
`char`	8	$-2^7 = -128$	$2^7 - 1 = 127$
`short int`	16	$-2^{15} = -32768$	$2^{15} - 1 = 32767$
`int`	32	$-2^{32} = -2147483648$	$2^15 - 1 = 2147483647$
`long int`	32	$-2^{32} = -2147483648$	$2^{32} - 1 = 2147483647$
`long long int`	64	-2^{64}	$2^{64} - 1$
`unsigned char`	8	0	$2^8 - 1$
`unsigned short int`	16	0	$2^{16} - 1$
`unsigned int`	32	0	$2^{32} - 1$
`unsigned long int`	32	0	$2^{32} - 1$
`unsigned long long int`	64	0	$2^{64} - 1$

`char` data type

- Is actually an integer type internally
- Each character has an integer code associated with it (ASCII code value)
- Internally, storing a character means storing its integer code
- All operators that are allowed on int are allowed on char
  - `32 + 'a'` will evaluate to `32 + 97` (the integer ascii code of the character `'a'`) $= 129$
  - Same for other operators
- Can switch on chars constants in `switch`, as they are integer constants

Examples

```
int  a;
a='c'*3+5;
printf("%d",a);
```

- It will print 296 (97*3+5)

```
char  c='A';
printf("%c=%d",c,c);
```

- It will print A=65

- Assigning `char` to `int` is fine. But other way round is **dangerous**, as size of int is larger

ASCII Code

- Each character is assigned a unique integer value (code) between 32 and 127
- The code of a character is represented by an 8-bit unit. Since an 8-bit unit can hold a total of $2^8 = 256$ values and the computer character set is much smaller than that, some values of this 8-bit unit do not correspond to visible characters

Decimal	Hex	Binary	Character
32	20	00100000	SPACE
48	30	00110000	0

Switching with `char` type

```
char letter
scanf("%c",&letter);
switch(letter){
  case 'A':
    printf("First letter\n");
    break;
  case 'Z':
    printf("Last letter\n");
    break;
  default:
    printf("Middle letter\n");
}
```

Switching with `char` type: example

```
switch(choice=getchar()){
  case 'r':
  case 'R':
    printf("Red\n");
    break;
  case 'b':
  case 'B':
    printf("Blue\n");
    break;
  case 'g':
  case 'G':
    printf("Green\n");
    break;
  default:
    printf("Black\n");
}
```

Evaluating expression

```
void main(){
  int op1, op2;
  int result=0;
  char op;
  scanf("%d",&op1);
  scanf("%c",&op);
  scanf("%d",&op2);
  switch(op){
    case’+’:
      result=op1+op2;
      break;
    case’-’:
      result=op1-op2;
      break;
    case’*’:
      result=op1*op2;
      break;
    case’/’:
      result=op1/op2;
      break;
    default:
      printf("Invalid operation\n");
  }
}
```

`bool` type

- Used to store boolean variables, like flags to check if a condition is true or false
- Can take only two values, true and false

```
bool negative = false;
int n;
scanf("%d", &n);
if (n < 0) negative = true;
```

- Internally, false is represented by 0, true is usually represented by 1 but can be different (print a bool variable with %d to see what you get)
- More compact storage internally