Typy danych

Aby zapisać w komputerze jakąś daną, trzeba zapamiętać trzy jej podstawowe cechy:

- miejsce przechowywania informacji,
- przechowywaną wartość,
- rodzaj przechowywanej wartości.

Typ użyty w deklaracji zmiennej decyduje o rodzaju informacji, a nazwa zmiennej symbolicznie opisuje wartość.

Typy danych

Przykład

```
int stopień;
stopień = 5;
```

- Instrukcje te mówią, że przechowywana jest liczba całkowita (ang. *integer*), *stopień* to nazwa wartości, w tym wypadku: 5.
- Program znajduje dostatecznie duży fragment pamięci, aby zapisać liczbę całkowitą, zapamiętuje jego położenie, przypisuje temu miejscu etykietę *stopień* i wstawia w to miejsce wartość 5.
- Instrukcje te nie mówią, gdzie w pamięci znajduje się wartość, ale program pamięta i to. Za pomocą operatora **&** można pobrać adres zmiennej *stopień*.

Typy danych

Operator **&** (czyt. *ampersand*) pobiera adres zmiennej.

```
#include <iostream>
using namespace std;
int main()
{
   int stopien;
   stopien = 5;
   int s = 2;
   cout << "Zmienna stopien o wartości: " << stopien;
   cout << " i adresie: " << &stopien << endl;
   cout << "Zmienna s o wartości: " << s;
   cout << " i adresie: " << &s;
}
```

```
Zmienna stopien o wartości: 5 i adresie: 0x6ffe3c
Zmienna s o wartości: 2 i adresie: 0x6ffe38
```

Typy danych

Zasady tworzenia nazw zmiennych:

- Jedyne znaki dopuszczalne w nazwach to litery, cyfry oraz podkreślenie (_).
- Pierwszy znak nie może być cyfrą.
- Wielkie litery są odróżniane od małych liter.
- Nazwa nie może być słowem kluczowym języka C++.
- Nazwy zaczynające się od dwóch podkreśleń lub od podkreślenia i za nim wielkiej litery są zarezerwowane dla implementacji, to znaczy dla kompilatora i wykorzystywanych przez niego zasobów.
- Nazwy zaczynające się od pojedynczego podkreślenia zarezerwowane są jako identyfikatory globalne implementacji.
- Długość nazwy zmiennej nie jest ograniczona i wszystkie znaki są brane pod uwagę.

Typy danych

Przykłady poprawnych i niepoprawnych nazw zmiennych.

int delta;	poprawna
int Delta;	poprawna, inna niż delta
int DELTA;	poprawna, inna niż dwie poprzednie
Int liczba;	niepoprawna, zamiast Int powinno być **int**
int myszka_miki;	poprawna
int myszka-miki;	niepoprawna, nie można używać minusów
int _delta2;	poprawna, ale zarezerwowana
int ___delta;	poprawna, ale zarezerwowana
int 2delta;	niepoprawna, zaczyna się od cyfry
int **double**;	niepoprawna, **double** jest słowem kluczowym
int begin;	poprawna, begin jest słowem kluczowym ale Pascala
int w_szczebrzeszynie_chrzaszcz_brzmi_w_trzcinie;	poprawna

Systematyka typów w C++

Typy języka C++ można podzielić dwojako:

- **Pierwszy podział**:
  - typy fundamentalne
    np: int, short, long, long long, char, float, double, long double
  - typy pochodne, które powstają na bazie typów fundamentalnych
    np: tablice, funkcje, wskaźniki, struktury
- **Drugi podział**:
  - typy wbudowane, czyli takie, w które język C++ jest wyposażony
  - typy zdefiniowane przez użytkownika

Typy całkowitoliczbowe

- Różne typy całkowite C++ różnią się ilością miejsca przeznaczonego na wartość.
  Większe zużycie pamięci oznacza możliwość zapisywania większych wartości.
- Niektóre typy pozwalają zapisywać wartości dodatne i ujemne (typy ze znakiem, **signed**), inne uniemożliwiają zapisanie wartości ujemnych (typy bez znaku, **unsigned**).
- Określając ilość pamięci zajmowanej przez typy całkowitoliczbowe, mówi się zwykle o *szerokości*. Im więcej pamięci dana wartość zajmuje, tym jest szersza.
- Podstawowe typy całkowitoliczbowe C++, ułożone według rosnącej szerokości: **char**, **short**, **int**, **long**, **long long**.

Bity i bajty*

- Podstawową jednostką pamięci komputerowej jest **bit**.
  O bicie można myśleć jako o przełączniku elektronicznym, który może być albo włączony, albo wyłączony.
  Wyłączenie oznacza wartość 0, włączenie: 1.
- 8-bitowy fragment pamięci może przyjmować 256 różnych wartości: skoro mamy 8 bitów, z których każdy daje 2 możliwości, to razem mamy $2^8 = 256$ możliwości.
  Zatem w 8-bitowej jednostce pamięci możemy zapisać na przykład liczby od 0 do 255 lub od -128 do 127.
- Każdy dodatkowy bit podwaja liczbę dostępnych kombinacji, zatem na 16 bitach można zapisać 65 536 różnych wartości, a na 32 bitach - 4 294 672 296.

Bity i bajty*

Przez **bajt** zwykle rozumie się 8-bitowy fragment pamięci. Tak rozumiany bajt jest jednostką miary pamięci komputera, gdzie kilobajt = 1024 bajty, a megabajt = 1024 kilobajty.

Jednak w C++ bajt zdefiniowano inaczej.

- Bajt (byte) składa się z przynajmniej tylu sąsiadujących bitów, ilu trzeba na zapisanie podstawowego zestawu znaków używanego w danej implementacji.
- Wobec tego liczba możliwych wartości bajta musi być przynajmniej równa liczbie różnych znaków.
- Zwykle bazuje się na zestawach znaków ASCII mających 8 bitów na znak. Dlatego w C++ bajt ma przeważnie 8 bitów.
- Jednak w przypadku programowania w środowisku bazującym na większych zestawach znaków, jak Unicode, bajt może mieć 16, a nawet 32 bity.

Bity i bajty*

Korzystając z różnej liczby bitów na wartość, typy **short**, **int** i **long** pozwalają zapisywać liczby całkowite rożnej szerokości. Dobrze byłoby, gdyby każdy z tych typów zawsze miał tę samą szerokość na wszystkich systemach - na przykład typ **short** 16 bitów, typ **int** 32 bity itd. Jednak żadna wielkość nie jest odpowiednia jednocześnie dla wszystkich możliwych konstrukcji komputerów. C++ jest elastyczny i określa tylko gwarantowaną wielkość minimalną.

- Liczby typu **short** muszą mieć przynajmniej 16 bitów.
- Liczby typu **int** muszą być nie mniejsze od liczb typu **short**.
- Liczby typu **long** muszą mieć przynajmniej 32 bity i być nie mniejsze od liczb typu **int**.

Aby poznać ograniczenia liczb całkowitych w używanym systemie, można skorzystać z narzędzi dostępnych w samym C++. Operator **sizeof** zwraca wyrażoną w bajtach wielkość typu lub zmiennej.

Ograniczenia liczb całkowitych*

```
#include <iostream>
using namespace std;
int main()
{
   int n_int = INT_MAX;
   short n_short = SHRT_MAX;
   long n_long = LONG_MAX;
   long long n_llong = LLONG_MAX;
   cout << "int ma " << sizeof (int) << " bajty." << endl;
   cout << "short ma " << sizeof n_short << " bajty." << endl;
   cout << "long ma " << sizeof n_long << " bajty." << endl;
   cout << "long long ma " << sizeof (long long) << " bajtów.";
   cout << endl << endl << "Wartości maksymalne:" << endl;
   cout << "int: " << n_int << endl;
   cout << "short: " << n_short << endl;
   cout << "long: " << n_long << endl;
   cout << "long long: " << n_llong << endl << endl;
   cout << "Minimalna wartość typu int = " << INT_MIN << endl;
   cout << "Bitów na bajt = " << CHAR_BIT << endl;
}
```

Ograniczenia liczb całkowitych*

Przykładowy wynik programu (Dev C++, Windows 10)

```
int ma 4 bajty.
short ma 2 bajty.
long ma 4 bajty.
long long ma 8 bajtów.

Wartości maksymalne:
int: 2147483647
short: 32767
long: 2147483647
long long: 9223372036854775807

Minimalna wartość typu int = -2147483648
Bitów na bajt = 8
```

Inicjalizacja zmiennych

Inicjalizacja stanowi połączenie deklaracji i przypisania.

```
int a = 5;
int b = a;              // b=5
int c = a + b + 3;      // c=13
```

Jeśli zmienne zdefiniowane w funkcji nie zostaną zainicjalizowane, wartość zmiennej będzie nieokreślona. Oznacza to, że wartość ta będzie przypadkowa, zależna od tego, co znajdowało się akurat w pamięci w chwili tworzenia danej zmiennej.

```
#include <iostream>
using namespace std;
int main()
{
   int a = 5;
   int b, c, d;
   cout << a << ", " << b << ", " << c << ", " << d;
}
```

```
5, 1, 0, 44372880
```

Typy całkowitoliczbowe - **char**

- Typ **char** przechowuje znaki takie jak litery i cyfry.
- O ile przechowywanie w komputerze liczb nie stanowi problemu, to z przechowywaniem znaków nie jest już tak łatwo.
- Języki programowania zamiast liter przechowują ich kody, dlatego typ char jest kolejnym typem liczbowym.

Typy całkowitoliczbowe - **char**

```
#include <iostream>
using namespace std;
int main()
{
   char c = 'B';
   int i = c;
   cout << "Kod ASCII znaku " << c << " to " << i << endl;
   cout << "Dodajemy do kodu znaku jedynkę:" << endl;
   c = c + 1;
   i = c;
   cout << "Kod ASCII znaku " << c << " to " << i << endl;
}
```

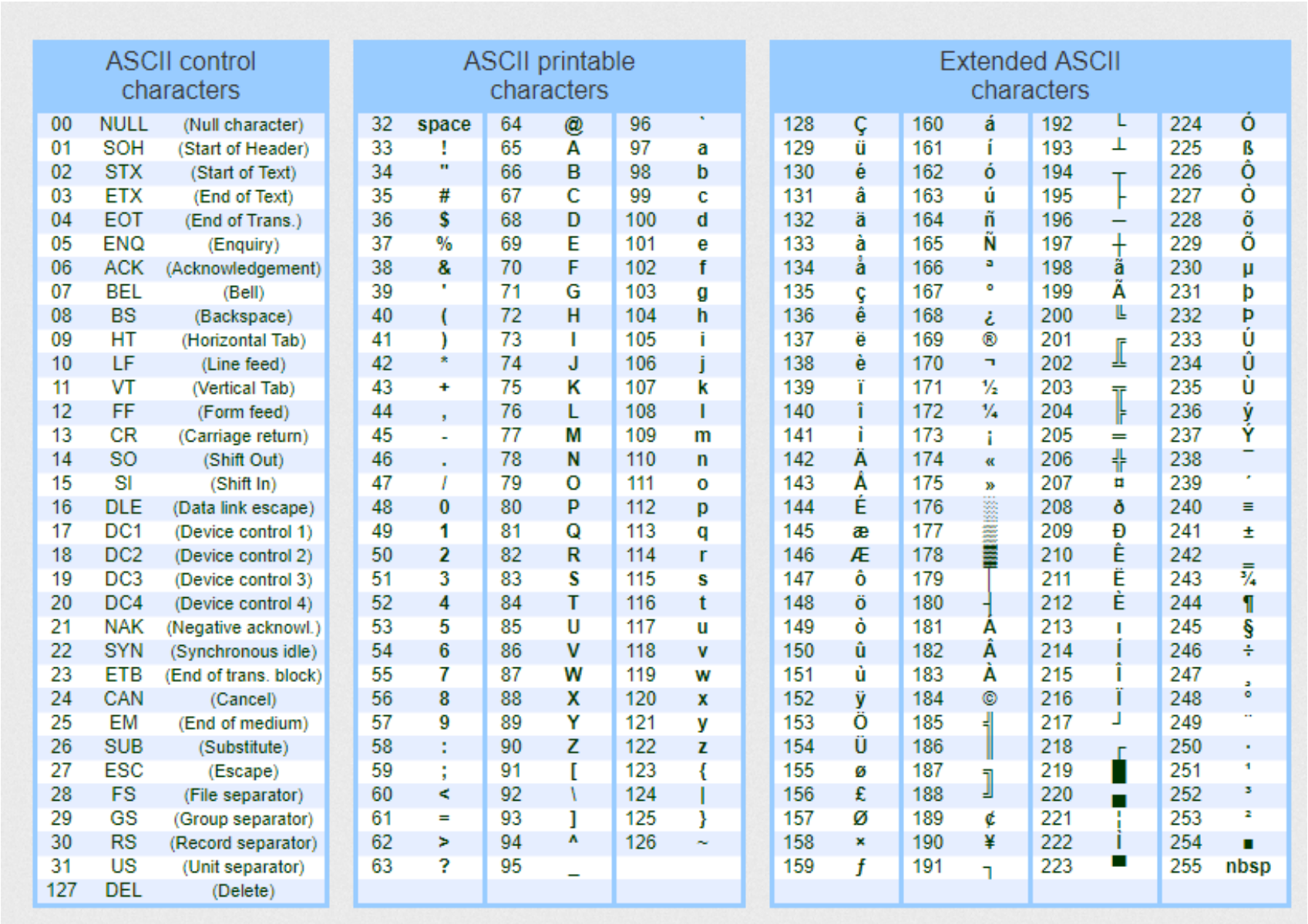
```
Kod ASCII znaku B to 66
Dodajemy do kodu znaku jedynkę:
Kod ASCII znaku C to 67
```

```
int main()
{
  char c = 'B';
  int i = c;
  cout << "Kod ASCII znaku " << c << " to " << i << endl;
  cout << "Dodajemy do kodu znaku jedynkę:" << endl;
  c = c + 1;
  i = c;
  cout << "Kod ASCII znaku " << c << " to " << i << endl;
}
```

- Zapis **'B'** oznacza kod liczbowy znaku **B**, więc inicjalizacja zmiennej **c** typu **char** wartością **'B'** powoduje ustawienie **c** na 66.
- Następnie program przypisuje tę samą wartość zmiennej **i** typu **int** tak, że **c** i **i** mają wartość 66.
- Następnie **cout** wyświetla wartość zmiennej **c** jako **B**, a zmiennej **i** jako 66.
  To typ zmiennej decyduje o sposobie jej wyświetlenia przez **cout**.
- Zmienna **c** zawiera tak naprawdę liczbę całkowitą, można stosować do niej operacje arytmetyczne, np. dodanie jedynki. Powoduje to zmianę wartości **c** na 67.

Tabela rozszerzona ASCII

(czyt. aski; ang. American Standard Code for Information Interchange)



ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Typy całkowitoliczbowe - **bool**

- Nazwa tego typu pochodzi od nazwiska angielskiego matematyka George'a Boole'a, który stworzył matematyczny zapis praw logiki.
- W obliczeniach zmienna typu **bool** (nazywana też zmienną logiczną) może mieć jedną z dwóch wartości: **true** (prawda) lub **false** (fałsz).
- Faktycznie jednak stan prawda przechowywany jest za pomocą wartości 1, natomiast stan fałsz za pomocą wartości 1.

```
int x = 8;
bool b;
b = false;
b = 6;
b = 0;
b = (x > 5);
```

Typy całkowitoliczbowe - **bool**

```
#include <iostream>
using namespace std;
int main()
{
   int n, i;
   bool pierwsza = true;
   cout << "Podaj liczbę naturalną: ";
   cin >> n;
   i = 2;
   while (i < n)
   {
      if (n % i == 0)
         pierwsza = false;
      i++;
   }
   if (pierwsza)
      cout << "Liczba " << n << " jest liczbą pierwszą";
   else
      cout << "Liczba " << n << " nie jest liczbą pierwszą";
}
```

Typy zmiennoprzecinkowe

Typy reprezentujące liczby zmiennoprzecinkowe umożliwiają pracę na liczbach rzeczywistych z różną dokładnością:

- **float** - ang. floating point: zmienny przecinek
- **double** - ang. double precision: podwójna dokładność
- **long double** - ang. long: długi, ulepszona wersja typu o podwójnej precyzji, rozszerzona dokładność

Przykłady

12.34	2.52e+8 = 252000000
939001.32	2.52e8 = 252000000
0.00023	8.33E-4 = 0.000833
5.0	-16.32e-3 = -0.01632

Typy zmiennoprzecinkowe

Typy te opisuje się przez liczbę cyfr znaczących oraz dopuszczalny zakres wykładników.
Cyfry znaczące to cyfry, które mają wpływ na wielkość liczby.

Liczby 123, 1.23, 0.123, 0.000123 mają 3 cyfry znaczące.
Liczba 12.3000 ma 6 cyfr znaczących.

Cyfry znaczące w C++ w typie:

- **float** - muszą być zapisywane przynajmniej na 32 bitach
- **double** - przynajmniej na 48 bitach i nie mniej niż w typie float
- **long double** - wymaga się przynajmniej tylu bitów, ile jest w typie double

Jednak zwykle typ float ma 32 bity, double 64, a long double 80, 96 lub 128 bitów.

Zakres *wykładników* dla wszystkich trzech typów nie może być mniejszy niż od -37 do $+37$.

Typy zmiennoprzecinkowe

Najczęsciej spotykane szerokości i zakresy wartości typów zmiennoprzecinkowych dla kompilatorów 32 bitowych.

float	32 bity	3.4e-38	...	3.4e38
double	64 bity	1.7e-308	...	1.7e308
long double	80 bitów	3.4e-4932	...	1.1e4932

Konwersje typów podczas przypisania

```
#include <iostream>
using namespace std;
int main()
{
   cout.setf(ios::showpoint);
   float f = 3;        // konwersja int na float
   int i = 5.986;      // konwersja float na int
   cout << "f = " << f << endl;
   cout << "i = " << i;
}
```

```
f = 3.00000
i = 5
```

- Polecenie **cout.setf(ios::showpoint);** służy do wyświetlenia kropki i zer nieznaczących w części dziesiętnej
- Zmienna *f* otrzymała wartość zmiennoprzecinkową 3.0.
- Przypisanie wartości 5.986 zmiennej *i* typu **int** spowodowało obcięcie tej wartości do 5; (C++ nie zaokrągla, tylko obcina).

Rzutowanie typów

Składnia

nazwa_typu (wartość)

```
#include <iostream>
using namespace std;
int main()
{
   int a, b;
   a = 19.99 + 11.99;                  // 31.98 -> 31
   b = int(19.99) + int(11.99);        // 19+11
   cout << "a= " << a << endl;
   cout << "b= " << b << endl;
   char c = 'A';
   cout << "Kod znaku " << c << " to " << int(c);
}
```

```
a= 31
b= 30
Kod znakou A to 65
```

Kwalifikator **const**

```
const int MIESIACE = 12;
```

- MIESIACE to stała symboliczna równa 12.
- Teraz zamiast liczby 12 możemy używać identyfikatora MIESIĄCE (samo 12 w programie może oznaczać liczbę cali w stopie albo liczbę butelek w kartonie; nazwa MIESIACE od razu mówi nam, co mamy na myśli).
- Powszechną praktyką jest zapisywanie nazw stałych wielkimi literami, aby w programie od razu było widać, że to właśnie stałe. Nie jest to konwencja obowiązująca, ale ułatwia pracę z programem.
- Słowo kluczowe **const** to kwalifikator, bo kwalifikuje deklarację, w tym wypadku jako deklarację stałej.

Kwalifikator **const**

- Po inicjalizacji stałej MIESIACE wartość ta jest ustalona i kompilator nie pozwoli już jej zmieniać.
- Zauważmy, że w deklaracji **const** od razu występuje inicjalizacja. Poniższy kod jest nieprawidłowy:

```
const int MIESIACE;
MIESIACE = 12;
```

- Jeśli nie podamy wartości stałej w deklaracji, będziemy mieli wartość nieokreśloną, której nie możemy zmienić.

Instrukcja **typedef**

Pozwala na nadanie dodatkowej nazwy już istniejącemu typowi.

```
typedef int cena;
cena x;          // odpowiada: int x;
cena a, b, c;    // odpowiada: int a, b, c;
```

Taka możliwość jest przydatna. Wyobraźmy sobie program, w którym wielokrotnie występują zmienne **int**. Niektóre z nich mają określać cenę, więc zastosowaliśmy instrukcję **typedef**. Jeżeli zdecydujemy, że dokładność liczb całkowitych nas nie zadowala i chcielibyśmy, aby ceny reprezentowane były typem **double**, wystarczy, że odszukamy w programie instrukcję **typedef** i zamienimy ją na następującą

```
typedef double cena;
```

Tym sposobem wszystkie miejsca w programie, gdzie używamy nazwy typu (np. deklarujemy zmienne typu cena) za jednym zamachem zmieniają się we właściwy sposób:

```
cena x;          // odpowiada: double x;
cena a, b, c;    // odpowiada: double a, b, c;
```